UNIVERSITY OF MIAMI

A FRAMEWORK FOR EFFICIENT IMPLEMENTATION AND EFFECTIVE VISUALIZATION OF DEMPSTER-SHAFER BELIEF THEORETIC COMPUTATIONS FOR REASONING UNDER UNCERTAINTY

By

Lalintha G. Polpitiya

A DISSERTATION PROPOSAL

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida

June 2018

UNIVERSITY OF MIAMI

A dissertation proposal submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

A FRAMEWORK FOR EFFICIENT IMPLEMENTATION AND EFFECTIVE VISUALIZATION OF DEMPSTER-SHAFER BELIEF THEORETIC COMPUTATIONS FOR REASONING UNDER UNCERTAINTY

Lalintha G. Polpitiya

Approved:

Kamal Premaratne, Ph.D. Professor of Electrical and Computer Engineering	Manohar N. Murthi, Ph.D. Associate Professor of Electrical and Computer Engineering
Stephen J. Murrell, Ph.D. Lecturer of Electrical and Computer Engineering	Jie Xu, Ph.D. Assistant Professor of Electrical and Computer Engineering
Dilip Sarkar, Ph.D. Associate Professor of Computer Science	Guillermo J. Prado, Ph.D. Dean of the Graduate School

POLPITIYA, LALINTHA G. (Ph.D. Candidate, Electrical and Computer Engineering)

A Framework for Efficient Implementation and Effective Visualization of Dempster-Shafer Belief Theoretic Computations for Reasoning Under Uncertainty (June 2018)

Abstract of a dissertation proposal at the University of Miami.

Dissertation proposal supervised by Professor Kamal Premaratne.
No. of pages in text. (117)

The convenience, intuitiveness, and versatility of Dempster-Shafer (DS) theoretic models of data uncertainty make DS evidence theory an ideal framework for reasoning under uncertainty and decision making in Artificial Intelligence (AI) applications. However, a major criticism cast towards DS theoretic (DST) evidential reasoning is the heavy computational burden it entails. If the advantages offered by DS theory are to be fully realized, it is essential that one explores efficient data structures and algorithms that can be used for DST operations and computations.

We present a novel generalized computational framework for exactly this purpose. We develop three representations — *DS-Vector, DS-Matrix,* and *DS-Tree* — which allow DST computation to be performed in significantly less time. These three representations can also be utilized as tools for visualizing DST models. A new strategy, which we refer to as *REGAP,* which allows *REcursive Generation of and Access to Propositions*, is introduced and harnessed in the development of this framework and computational algorithms. A new computational library, which we refer to as *BCL (Belief Computation Library)* is developed and utilized in the simulations. We provide a discussion and experimental validation of the utility, efficiency, and implementation of the proposed data structures and algorithms.

As in Bayesian probability theory, the conditional operation plays a pivotal role in DS theoretic (DST) strategies for evidence updating and fusion. Again, a major limitation in applying DST techniques for reasoning under uncertainty is the absence of a feasible computational framework to overcome the prohibitive computational burden this conditional operation entails. This is a known problem with non-deterministic polynomial-time hardness (NP-hard). We address this critical challenge via two novel generalized conditional computational models — *DS-Conditional-One* and *DS-Conditional-All* — which allow significantly less computational and space complexity when computing conditional mass and belief. They provide deeper insight into the DST conditional itself and so act as a valuable visualization tool during conditional computation. We provide a thorough analysis and experimental validation of the proposed data structures and algorithms for computing both the *Dempster's conditional* and the *Fagin-Halpern conditional,* the two most widely utilized DST conditional strategies. Two new computational libraries, which we refer to as *CCL (Conditional Computation Library)*, and *DS-CONAC (DS-Conditional-One and DS-Conditional-All in C++)* are developed and harnessed in our work.

The current work that is being carried out involves developing efficient algorithms for DST fusion strategies, including *Dempster's rule of combination*, *Conditional Update Equation (CUE)*, *Conditional Fusion Equation (CFE)* and *pignistic transformation.* Our research will include developing data structures to work with general low density bodies of evidence, as well as special categories like *Dirichlet belief functions* and *consonant belief functions.* We are working on computations involving potentially dynamic frames. A collection of effective visualization tools and computational libraries complementing our work will be made available.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

"As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality" - Albert Einstein

1.1 Reasoning Under Uncertainty: The Role of Dempster-Shafer (DS) Belief Theory

The recent momentum in Artificial Intelligence (AI) is a culmination of well constructed algorithms, unprecedented processing power of computers, and effective use of large flows of data. Today expert systems beat champion players at Chess and Go [1, 2], allow AI assistants to book appointments over the phone [3], diagnose and predict disease in health care [4, 5], and drive autonomous vehicles [6]. However uncertainty is unavoidable in many real-world domains, and expert systems are still prone to collapse due to the difficulty in replicating complex environments [7, 8]. To accommodate uncertainty and data imperfections intelligently, we need to have effective models to capture them and then we must propagate and reason with these data uncertainties.

The Dempster-Shafer (DS) belief theory, also referred to as DS evidence theory, is a powerful and convenient framework that can handle a wide variety of uncertainty and data imperfections [9–11]. Dempster-Shafer (DS) theory was first introduced by Dempster [12, 13] in the context of statistical inference. It was later developed by Shafer [14] into a general framework for uncertainty modeling. The name "Dempster-Shafer theory" was coined in [15], which also introduced the theory to a wider audience in the AI community. The early contributions laid the foundation for many important developments, including the transferable belief model (TBM) [16] and the theory of hints [17]. DS theoretic (DST) approach offers a convenient framework for reasoning when prior information is lacking. Consequently, DST methods are finding increased utilization in numerous application scenarios and have generated an active research field [11, 18–21].

1.2 Motivation

While DS theory offers greater expressiveness and flexibility in evidential reasoning under uncertainty [22], these advantages come at a cost: DST operations involve an additional cost in terms of higher computational complexity, especially when compared to methods based on classical probability theory. Computing the DST conditionals, a key operation in evidence updating and fusion, and even the computation of DST belief functions, are non-deterministic polynomial-time hardness (NP-hard) problems [23, 24]. This computational difficulty is the main criticism that the DS formalism has drawn since its very inception [11].

A major challenge for harnessing the advantages of DS theory in practice is to overcome this computational complexity challenge, especially when working with large 'frames of discernment'. The problem is further exacerbated by the absence of a flexible and scalable platform for visualizing the complex operations involved DS theory.

The development of an efficient computational framework is of critical importance if we are to harness the strengths of DS theory and make it more widely applicable in practice and in real-world scenarios, and this problem has been identified as an issue that requires increased attention [25].

1.3 Challenges

1.3.1 Making Exact Computations Feasible

In order to address the high computational complexity of DST belief computations, several approximation methods have been discussed [26–32]. Most of these approximation algorithms provide lower bounds which are obtained by removing some of the focal elements with or without redistributing the corresponding belief potentials [26–30]. More sophisticated methods produce lower and upper bounds, thus improving the quality of the approximation [31, 32]. These methods provide approximate and not exact belief potentials. Many of these approaches dramatically reduce the quality of the approximation when applied to the conditional and fusion operations, and some lack the ability to be extended for DST conditional computations.

The quality of evidence updating and fusion strategies depend directly on the precision of the underlying computations. Thus, making exact (or precise) computations feasible is of utmost importance for purposes of reasoning under uncertainty. A fast Möbius transform, which is analogous to the fast Fourier transform (FFT), has also been developed toward efficient DST computations [33–35]. It is worth pointing out that Shafer has stated, *“It remains to be seen how useful the fast Möbius transform will be in practice. It is clear, however, that it is not enough to make arbitrary belief function computations feasible.”* [22, p.348].

1.3.2 Developing a Feasible and Scalable Computational Framework

There is no computationally feasible and scalable generalized framework to represent DST models and carry out DST operations. We do not believe that the present literature has given due attention to this issue [9, 11]. A thoughtful discussion about data structures and algorithms for efficient DST computations is still lacking. The development of an efficient and scalable computational framework, which includes a wide range of data structures and well constructed algorithms, is essential in order to utilize the strengths of DS methods in practical applications.

1.3.3 Handling Large Frames of Discernment

DST implementations in current use are limited to computations on smaller frames of discernment, and they lack the ability to handle larger frames mainly due to the prohibitive computational complexity they engender. A review of current implementations and applications [9, 11, 36] confirms that work is needed to overcome these computational limitations.

1.3.4 Efficient Computation of DST Conditionals

As in the Bayesian methods [37], the conditional operation plays a fundamental role in DST strategies for evidence updating and fusion and in general, in reasoning under uncertainty. There are numerous strategies of DST conditioning that have appeared in the literature [14, 38–42]. The selection of the appropriate conditioning strategy must always be given due consideration.

Among these various conditional notions that have been proposed over the years, perhaps the most extensively utilized DST conditional notion is the *Dempster's conditional* [14, 43–46]. On the other hand, the *Fagin-Halpern (FH) conditional* can be considered the most natural generalization of the probabilistic conditional notion because of its close connection with the inner and outer conditional probability measures [40]. Recent work on the DST conditional approach [47, 48], which is based on the FH conditional, demonstrate how both soft and hard evidence can be incorporated into the reasoning process with DST methods.

A review of existing studies on precise belief computations and their applications [33–36,43,46,49–57] reveals that more work is needed to overcome these computational limitations associated with DST conditionals. The fast Möbius transform which has been utilized for efficient and precise computation of DST belief functions [33–35, 50] has not been employed for the computation of DST conditionals.

As for the Dempster's conditional, perhaps the most thorough discussions for carrying out its precise computation appears in [43, 46]. It provides a matrix calculus based algorithm to compute Dempster's conditional masses. However, this approach is feasible only on smaller frames because of the expensive matrix operations involved with a certain specialization matrix which is a $2^{|\Theta|} \times 2^{|\Theta|}$ stochastic matrix ($|\Theta|$ is cardinality of the frame of discernment). For example, the theoretical computational time for this method is more than 1800 CPU (Central Processing Unit) years for a frame size of 30 and more than 15 CPU hours for a frame size of 20 (assuming 10 million computational iterations per second). Computational complexity and space complexity are both $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$ for this approach. Thus, the computation is infeasible for larger frames.

This method is not applicable for computing the FH conditional. As for FH conditional computation, the work in [58, 59] presents the conditional core theorem (CCT) to identify propositions that retain non-zero support after FH conditioning. However, this approach does not address conditional computation of these propositions. The computational complexity of the CCT becomes $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$ for high density bodies of evidence, thus the conditional computation becomes prohibitive for larger frames.

1.3.5 Visualization and Analysis of Complex DST Operations

Many scientific disciplines involve the analysis of large collections of numeric data. Scientific visualization involves transforming numeric data into more visual forms so that scientists can understand and gain insights from this data readily [60]. Visualization has become an effective technology [61], especially in the analysis of complex situations, and in reasoning of both abstract and concrete ideas. Similarly, the ability to visualize complex DST computations and simulations is absolutely essential to ensure the integrity of representation and reasoning, to provoke insights, and to carefully improve the computational performance.

1.4 Contributions

The work being proposed is an attempt to fill the void between what DS theory can offer and its practical implementation. We now summarize the main contribution of this research work.

1.4.1 Scalable Generalized Computational Framework

We introduce a novel generalized computational framework where we develop three different representations — *DS-Vector, DS-Matrix,* and *DS-Tree* — that offer significantly greater computational capability for representation of DST models and DST operations. They also act as simple tools for visualization of DST models and the complex nature of the computations involved. A strategy, which we refer to as *REGAP (REcursive Generation of and Access to Propositions),* is developed and used in our development of this computational framework. Relevant data structures and generalized algorithms to work with the framework are discussed and compared in Chapter 3 [56,62].

1.4.2 Implicit Index Calculation Mechanism

Popular approaches to represent a focal element (i.e., a proposition which receives DST 'support') are the use of a bit-string [33,35,51] or an integer [63]. Binary representation has been used in combination, marginalization, and projection of multivariate belief functions [53,54]. Binary representation has also been used to express the belief computation formulas using matrix calculus [46] and in fusion algorithm implementations [64]. In Chapter 3, we introduce an implicit index calculation mechanism to represent a focal element, which reduces memory usage and significantly improves computational performance [56].

1.4.3 Efficient Computation of DST Operations

Using the REGAP strategy, we introduce a new approach to identify propositions that are relevant to a belief potential computation. This technique is useful for calculating arbitrary belief, plausibility, and commonality potentials from a minimum number of operations. Implementation of the relevant belief computation algorithms based on the proposed framework is discussed and compared with alternative approaches in Chapter 3 [56].

1.4.4 Efficient Computation of DST Conditionals

The main contribution of Chapter 4 and Chapter 5 is a novel scalable, generalized computational framework for computing DST *conditionals.* This framework includes two conditional computational models — *DS-Conditional-One* and *DS-Conditional-All* — which offer significantly greater flexibility and computational capability for implementation of DST conditional strategies.

- DS-Conditional-One model can be employed to compute both the FH and Dempster's conditional beliefs of an *arbitrary* proposition, as well as to compute Dempster's conditional masses of an *arbitrary* proposition. The difficulty in arbitrary computations is exactly the challenge that Shafer refers to in [22, p.348], viz., *"It remains to be seen how useful the fast Möbius transform will be in practice. It is clear, however, that it is not enough to make arbitrary belief function computations feasible."*

- DS-Conditional-All model can be utilized for efficient computation of all conditional beliefs. Computation of *all* FH and Dempster's conditional beliefs are discussed using this model and it can also be used to compute Dempster's conditional masses of *all* propositions.

This computational framework is a significantly better approach to conditional computations in contrast to the current available strategies. By carefully reducing the number of operations being executed, the proposed approach takes significantly less computational and space complexity when compared to other approaches for conditional computation. For example, our experimental results demonstrate that the average computational time taken to compute the Dempster's conditional mass of an arbitrary proposition by the proposed approach is less than 1.3 (μs) for a frame of size 10 ($\sim$1000 focal elements), 77.2 (μs) for a frame of size 20 ($\sim$1 million focal elements) and less than 111.3 (ms) for a frame of size 30 ($\sim$1 billion focal elements). Also the average computational time taken to compute the Dempster's or Fagin-Halpern conditional belief of an arbitrary proposition by the proposed approach is less than 1.7 (μs) for a frame of size 10 ($\sim$1000 focal elements), 472.5 (μs) for a frame of size 20 ($\sim$1 million focal elements) and less than 1.6 (sec) for a frame of size 30 ($\sim$1 billion focal elements). This computational framework can also be utilized as a visualization tool for conditional computations and in analyzing characteristics of conditioning and updating operations. The computational implementations in available literature are limited only to one particular conditional strategy, but the conditional computational models we present in Chapter 4 and Chapter 5 can be utilized as a common platform to carry out both Dempster's and FH conditional computations. We believe that this computational framework and the associated implementations constitute a significant step toward closing the gap between what the DST framework can offer for reasoning under uncertainty and its utility in practical application scenarios [57, 65–67].

1.4.5 Computational Libraries

As an outcome of this research work, three computational libraries which include all software routines that we have developed are being made available [62, 65, 67].

The DST data structures and algorithms for DST operations in Chapter 3 are being made available in a new computational library, which we refer to as *BCL (Belief Computation Library)* [62].

The data structures and algorithms in Chapter 4 and Chapter 5 for computing the Dempster's conditional and the FH conditional are being made available in two additional computational libraries, which we refer to as *CCL (Conditional Computation Library)* [65] and *DS-CONAC (DS-Conditional-One and DS-Conditional-All in C++)* [67].

1.4.6 Effective Visualization Tools

It is a challenging task for the user to quantitatively examine non-deterministic polynomial-time hardness (NP-hard) problems associated with large frames. Simple numerical formats and models are not the best way to assist the human brain to interpret large flows of data associated with such frames. Visualization can assist the scientist in several ways. Changes in parameters can be readily grasped, patterns and classifications become more visible, and anomalies in the data can be easily detected. Therefore, the ability to visualize complex DST computations and simulations is absolutely essential to reinforce cognition, decision making, and reasoning [60, 61].

We have developed a sequence of scalable graphical illustrations for visualizing DST computations. We have utilized these visualizations to gain a better understanding of DST computations and to develop efficient algorithms. In Chapter 3 we first visualize the DST operations using DS-Vector, DS-Matrix and DS-Tree data structures. We then visualize the arbitrary conditional computations using DS-Conditional-One model in Chapter 4 [57, 65], and all conditional computations using DS-Conditional-All model in Chapter 5 [66, 67].

1.5 Organization of the Dissertation Proposal

The organization of the dissertation proposal is as follows:

1.5.1 Chapter 2 Preliminaries

Chapter 2 provides a review of those notions related to DS theory that are essential for the work presented in this dissertation proposal.

1.5.2 Chapter 3 A Framework for Efficient Computation of Belief Theoretic Operations

Chapter 3 presents a novel generalized computational framework for efficient computation of belief theoretic operations. We develop three representations — *DS-Vector, DS-Matrix,* and *DS-Tree* — which allow DST computation to be performed in significantly less time. These three representations can also be utilized as tools for visualizing DST models. A new strategy, which we refer to as *REGAP,* which allows *RE*cursive *G*eneration of and *A*ccess to *P*ropositions is introduced and harnessed in the development of this framework and computational algorithms. A new computational library, which we refer to as *BCL (Belief Computation Library)* [62] is developed and utilized in the simulations. We provide a discussion and experimental validation of the utility, efficiency, and implementation of the proposed data structures and algorithms [56].

1.5.3 Chapter 4 DS-Conditional-One: Efficient and Exact Computation of Arbitrary Conditionals

Chapter 4 addresses the critical challenge of computing DST conditionals via a novel generalized conditional computational model — *DS-Conditional-One* — which allows the conditional to be computed in significantly less computational and space complexity. This computational model also provides valuable insight into the DS theoretic conditional itself and can be utilized as a tool for visualizing the conditional computation. We provide a thorough analysis and experimental validation of the utility, efficiency, and implementation of the proposed data structures and algorithms for carrying out both the Dempster's conditional and FH conditional. A new computational library, which we refer to as *CCL (Conditional Computation Library)* [65], is developed and harnessed in the simulations [57].

1.5.4 Chapter 5 DS-Conditional-All: Efficient and Exact Computation of All Conditionals

Chapter 5 provides a novel generalized conditional computational model — *DS-Conditional-All* — which allows significantly less computational and space complexity when computing all conditional masses and beliefs. This computational model also provides deeper insight into the DST conditional itself and can be utilized as a tool for visualizing the conditional computation. We provide a thorough analysis of the proposed data structures and algorithms for computing both the Dempster's conditional and FH conditional. A new computational library, which we refer to as *DS-CONAC (DS-Conditional-One and DS-Conditional-All in C++)* [67] is developed and harnessed in the simulations [66].

1.5.5 Chapter 6 Operations on Dynamic Frames

Chapter 6 presents algorithms for operations on dynamic frames. Removing one singleton from a frame removes half the propositions that need to be considered. Thus, from a computational perspective, the ability to add, remove, and update singletons in a frame is highly important. Adding and removing operations are discussed in this chapter using the data structures DS-Vector, DS-Matrix, and DS-Tree.

1.5.6 Chapter 7 Future Work

Chapter 7 provides a discussion of our future work. Based on the proposed computational framework, we are working on developing efficient algorithms for DST fusion strategies, including *Dempster's rule of combination* [14], *Conditional Update Equation (CUE)* [47], *Conditional Fusion Equation (CFE)* [68], and *pignistic transformation* [16,69,70].

The current work that is being carried out involves developing efficient algorithms and data structures to work with general low density bodies of evidence, as well as special categories like *Dirichlet belief functions* [71,72] and *consonant belief functions* [14]. We are also conducting a wide range of experiments with dynamic operations to improve and conclude the work discussed in Chapter 6. This would be of immense value for enhanced resource utilization.

Synthetic aperture radar (SAR) interferometry (InSAR) [73] is an important technique that can measure terrain deformation with high precision. It has applications for geophysical monitoring including earthquakes, volcanic eruptions, landslides, and hydrological subsidence [74–79]. Our contributions will include a new selection criterion to identify the best baseline network, which is a primary component in InSAR processing [73]. This is still an open challenge. We have conducted two research studies: network selection using centrality concepts [80]; and identifying good networks applying deep learning [81] techniques. Both these initial steps yielded promising results. The ongoing challenging task is to adapt uncertainty reasoning capabilities and develop a robust application for InSAR processing.

Geometrical exposition is an important consideration in designing effective data structures and algorithms. Two novel DST visualization tools are being devised in order to scrutinize complex fusion strategies and to augment computational optimization. We call these tools DS-LASIC (DS-Layered Symmetric Clustering) diagram and DS-TRISEV (DS-Three Dimensional Spring Electrical Visualization) model. Based on these tools, we are developing a new class of Venn diagrams [82] for computational analysis which are flexible and expandable. In addition to the possible advantages we can gain in DST implementations, perhaps this will be of importance to the fields of discrete and computational geometry, and combinatorics.

A collection of open source computational libraries will be made available as an outcome of this research.

CHAPTER 2

Preliminaries

2.1 DST Basic Notions

In DS theory, the *frame of discernment (FoD)* refers to the set of all possible mutually exclusive and exhaustive propositions [14]. We consider the case where the FoD is finite and we denote it as $\Theta = \{\theta_0, \theta_1, \ldots, \theta_{n-1}\}$. Note that, for computational ease, we use the indices 0 and $n-1$ for the first and the last elements, respectively. Proposition $\{\theta_i\}$, which is referred to as a *singleton,* represents the lowest level of discernible information. The power set of Θ, denoted by 2^Θ, form all the propositions of interest in DS theory. A proposition that is not a singleton is referred to as a *composite.* The set $A \backslash B$ denotes all singletons in $A \subseteq \Theta$ that are not included in $B \subseteq \Theta$, i.e., $A \backslash B = \{\theta_i \in \Theta \mid \theta_i \in A, \, \theta_i \notin B\}$. We use $\overline{A}$ to denote $\Theta \backslash A$ and $|A|$ to denote the cardinality of A.

2.1.1 Basic Belief Assignment (BBA) or Masses

In DS theory, the *basic belief assignment (BBA)* or *mass* is used to represent the 'support' that is strictly allocated to a given proposition.

Definition 1 (Basic Belief Assignment (BBA) or Masses) *The mapping* $m : 2^{\Theta} \mapsto [0,1]$ *is said to be a* basic belief assignment (BBA) *or a* mass assignment *if*

$$m(\emptyset) = 0 \text{ and } \sum_{A \subseteq \Theta} m(A) = 1. \qquad \blacksquare$$

The mass of a composite proposition (a general focal element) is free to move into its subsets and specially into individual singletons, which allows one to model the notion of *ignorance.* Complete ignorance can be modeled via the *vacuous BBA:*

$$m(A) = 1_{\Theta} \equiv \begin{cases} 1, & \text{for } A = \Theta; \\ 0, & \text{for } A \subset \Theta. \end{cases} \tag{2.1}$$

Propositions that possess nonzero mass are referred to as *focal elements;* the set of all focal elements in an FoD is referred to as its *core* $\mathfrak{F}$, i.e., $\mathfrak{F} = \{A \subseteq \Theta \mid m(A) > 0\}$. Note that $|\mathfrak{F}|$ is the number of focal elements. $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ is referred to as the *body of evidence (BoE).*

2.1.2 Belief

The *belief* assigned to a proposition takes into account the support for all of its subsets.

Definition 2 (Belief) *Given a BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$, *the* belief *assigned to* $A \subseteq \Theta$ *is* $Bl : 2^{\Theta} \mapsto [0,1]$ *where*

$$Bl(A) = \sum_{B \subseteq A} m(B). \qquad \blacksquare$$

Propositions that possess nonzero belief are denoted by $\widehat{\mathfrak{F}}$, i.e., $\widehat{\mathfrak{F}} = \{A \subseteq \Theta \mid Bl(A) > 0\}$.

Also, given a valid belief function $Bl : 2^{\Theta} \mapsto [0, 1]$, one may generate the corresponding BBA $m : 2^{\Theta} \mapsto [0, 1]$ via the Möbius transform [14]

$$m(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|} Bl(B), \ \forall A \subseteq \Theta. \tag{2.2}$$

2.1.3 Plausibility

The *plausibility* measures the extent to which a proposition is plausible, i.e., the amount of belief not strictly supporting the complement of the proposition.

Definition 3 (Plausibility) *Given a BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$, *the* plausibility *assigned to* $A \subseteq \Theta$ *is* $Pl : 2^{\Theta} \mapsto [0, 1]$ *where*

$$Pl(A) = 1 - Bl(\overline{A}). \qquad \blacksquare$$

It is easy to see that, for all $A \subseteq \Theta$,

$$Pl(A) = \sum_{\substack{B \subseteq \Theta \\ B \cap A \neq \emptyset}} m(B) = 1 - Bl(\overline{A}) \geq Bl(A), \forall A \subseteq \Theta. \tag{2.3}$$

The *uncertainty* $Un(A)$ associated with the proposition $A \subseteq \Theta$ is taken as the interval $Un(A) = [Bl(A), Pl(A)]$.

2.1.4 Commonality

The *commonality* quantifies the support for those propositions that imply a given proposition.

Definition 4 (Commonality) *Given a BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$, *the commonality function of* $A \subseteq \Theta$ *is* $Q : 2^{\Theta} \mapsto [0, 1]$ *where*

$$Q(A) = \sum_{A \subseteq B \subseteq \Theta} m(B). \qquad \blacksquare$$

2.2 Dempster's Conditional

Dempster's conditional is perhaps the most widely employed DST conditional notion [14].

Definition 5 (Dempster's Conditional [14]) *Consider the BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ *and* $A \subseteq \Theta$ *s.t.* $Bl(\overline{A}) \neq 1$, *or equivalently,* $Pl(A) \neq 0$. *The* conditional belief $Bl(B\|A) : 2^{\Theta} \mapsto [0,1]$ *of* B *given the conditioning event* A *is*

$$Bl(B\|A) = \frac{Bl(\overline{A} \cup B) - Bl(\overline{A})}{1 - Bl(\overline{A})};$$

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

The conditional mass $m(B\|A) : 2^{\Theta} \mapsto [0,1]$ *of* B *given the conditioning event* A *is*

$$m(B\|A) = \begin{cases} \dfrac{\sum\limits_{C \subseteq \overline{A}} m(B \cup C)}{1 - Bl(\overline{A})}, & \text{for } \emptyset \neq B \subseteq A; \\ 0, & \text{otherwise.} \end{cases} \quad \blacksquare$$

One may compute the corresponding conditional mass $m(B\|A)$ and $Pl(B\|A)$ from $Bl(B\|A)$. Dempster's conditioning annuls masses of all those propositions that 'straddle' the conditioning proposition A and its complement $\overline{A}$. So, w.l.o.g., for Dempster's conditioning, one may consider only those propositions $B \subseteq A$.

2.3 Fagin-Halpern (FH) Conditional

FH conditional can be considered the most natural generalization of the probabilistic conditional notion because of its close connection with the inner and outer conditional probability measures in probability theory [40].

Definition 6 (Fagin-Halpern (FH) Conditional [40]) *Consider the BoE $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ and $A \in \widehat{\mathfrak{F}}$. The* conditional belief $Bl(B|A) : 2^{\Theta} \mapsto [0,1]$ *of B given the conditioning event A is*

$$Bl(B|A) = \frac{Bl(A \cap B)}{Bl(A \cap B) + Pl(A \cap \overline{B})};$$

$$Pl(B|A) = \frac{Pl(A \cap B)}{Pl(A \cap B) + Bl(A \cap \overline{B})}. \qquad \blacksquare$$

The conditional plausibility $Pl(B|A)$ of B given A is computed as $Pl(B|A) = 1 - Bl(\overline{B}|A)$. Of course, once the conditional beliefs of all the propositions are computed, one may obtain the corresponding conditional BBA via a Möbius transform of the type in (2.2).

Suppose the BoE $\{\Theta, \mathfrak{F}, m(\cdot)\}$ is being conditioned w.r.t. the proposition $A \in \widehat{\mathfrak{F}}$. The propositions that retain a nonzero mass after conditioning are referred to as the *conditional focal elements;* the set of all such conditional focal elements is referred to as the *conditional core* $\mathfrak{F}_A$, i.e., $\mathfrak{F}_A = \{B \subseteq A \in \widehat{\mathfrak{F}} \mid m(B|A) > 0\}$.

Similar to Dempster's conditioning, FH conditioning annuls masses of all those propositions that 'straddle' the conditioning proposition A and its complement $\overline{A}$. So, w.l.o.g., for FH conditioning, one may consider only those propositions $B \subseteq A$.

CHAPTER 3

A Framework for Efficient Computation of Belief Theoretic Operations

The work in this chapter is an attempt to fill the void between what DS theory can offer and its practical implementation. For this purpose, we introduce a novel generalized computational framework where we develop three different representations — *DS-Vector, DS-Matrix,* and *DS-Tree* — which offer significantly greater computational capability for representation of DST models and DST operations. They also act as simple tools for visualization of DST models and the complex nature of the computations involved. A strategy, which we refer to as *REGAP (REcursive Generation of and Access to Propositions),* is developed and used in development of this computational framework. We also introduce an implicit index calculation mechanism to represent a focal element, which reduces memory usage and significantly improves computational performance.

We introduce a new approach to identify propositions that are relevant to a belief potential computation, using the REGAP strategy. This technique is useful for calculating arbitrary belief, plausibility, and commonality potentials from a minimum number of operations. Implementation of the relevant belief computation algorithms based on the proposed framework is discussed and compared with alternative approaches. A new computational library, which we refer to as *BCL (Belief Computation Library)* [62] is developed and utilized in the simulations [56].

This chapter is organized as follows: Section 3.1 introduces our generalized computational framework; Section 3.2 contains efficient algorithms for belief potential computation of arbitrary propositions; Section 3.3 contains comparisons and experimental results; Section 3.4 provides the concluding remarks.

3.1 A General Computational Framework

Henceforth, for convenience, we will use the notation N to denote 2^n, where $n = |\Theta|$. Note that, $N = 2^n$ is the maximum number of focal elements that a BoE could possess. Actually, the maximum number of focal elements is $N - 1$, but this difference is immaterial especially when working with a large FoD. Also we will refer to the subsets of A and A itself as *subset propositions* of A and use the notation M to denote 2^m, where $m = |A|$.

A lookup table named *power* is used to enhance the computational efficiency. It contains 2 to the power of singleton indexes in increasing order and was implemented using a dynamic array that replaces run-time computation of power values with a simpler array indexing operation. $Power[i]$ is referred to as 2 to the power of i, which is the i'th entry in the *power* table.

Computational complexity of DST computations directly depends on the access speed of focal elements and corresponding belief values. In recent literature, the commonly used approach for such computations is the utilization of list structures for both focal elements and respective belief values, or hard coding relevant values [36,53,63]. Representation as a list or a set of pairs of focal elements and relevant belief values are also used. Accessing a focal element in list implementation requires cycling through the lists, and complexity of this operation is $\mathcal{O}(|\mathfrak{F}|)$. In the novel generalized computational framework that is being proposed in this chapter, propositions are identified via an implicit index. Therefore, there is no overhead on storing focal elements as a bit-string, an integer, or any other method; only belief potentials require to be stored. Due to the efficient access operations, the proposed method can be utilized to gain a significant advantage on computational efficiency when dealing with large BoEs. List structures become attractive when considering the memory usage aspect of static BoEs, with a fewer number of focal elements.

The proposed belief computation strategy provides generated indices. Therefore, computational complexity of an element access is $\mathcal{O}(1)$ (i.e., constant) during belief computations. In the provided general access algorithms which include index generation, the computational complexity is $\mathcal{O}(m)$.

3.1.1 REGAP: REcursive Generation of and Access to Propositions

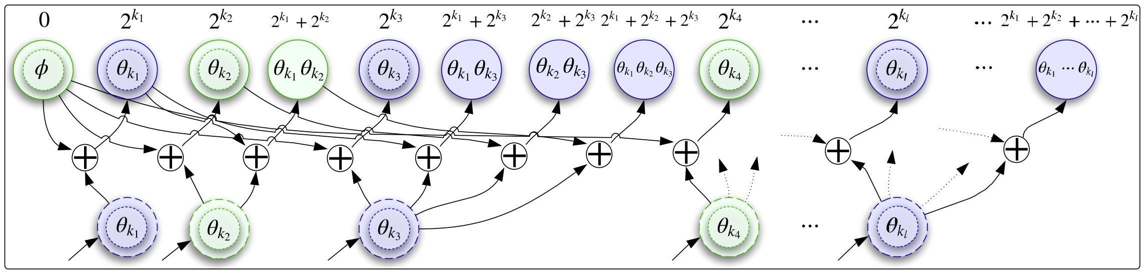

0
2^{k_1}
2^{k_2}
$2^{k_1}+2^{k_2}$
2^{k_3}
$2^{k_1}+2^{k_3}$
$2^{k_2}+2^{k_3}$
$2^{k_1}+2^{k_2}+2^{k_3}$
2^{k_4}
$\cdots$
2^{k_l}
$\cdots\ 2^{k_1}+2^{k_2}+\cdots+2^{k_l}$
ϕ
θ_{k_1}
θ_{k_2}
$\theta_{k_1}\theta_{k_2}$
θ_{k_3}
$\theta_{k_1}\theta_{k_3}$
$\theta_{k_2}\theta_{k_3}$
$\theta_{k_1}\theta_{k_2}\theta_{k_3}$
θ_{k_4}
θ_{k_l}
$\theta_{k_1}\cdots\theta_{k_l}$

Figure 3.1: **REGAP:** REcursive Generation of and Access to Propositions.

Consider the FoD $\Theta = \{\theta_0, \theta_1, \ldots, \theta_{n-1}\}$. Suppose we desire to determine the belief potential $Bl(A)$ associated with $A = \{\theta_{k_0}, \theta_{k_1}, \ldots, \theta_{k_{|A|-1}}\} \subseteq \Theta$. Then, $REGAP(A)$ recursively generates all the $2^{|A|}-1$ propositions whose masses are required to compute

$Bl(A)$, viz., all subsets of A (including A itself). We will refer to the subsets of A and A itself as *subset propositions* of A. $REGAP(A)$ is implemented in the following manner: Start with $\{\emptyset\}$. First insert the singleton $\{\theta_{k_0}\} \in A$. Only one proposition is associated with this singleton, viz., $\{\emptyset\} \cup \{\theta_{k_0}\} = \{\theta_{k_0}\}$ itself. Next insert another singleton $\{\theta_{k_1}\} \in A$. The new propositions that are associated with this singleton are $\{\emptyset\} \cup \{\theta_{k_1}\} = \{\theta_{k_1}\}$ and $\{\theta_{k_0}\} \cup \{\theta_{k_1}\} = \{\theta_{k_0}, \theta_{k_1}\}$. Inserting the next singleton $\{\theta_{k_2}\} \in A$ brings the new propositions $\{\emptyset\} \cup \{\theta_{k_2}\} = \{\theta_{k_2}\}$, $\{\theta_{k_0}\} \cup \{\theta_{k_2}\} = \{\theta_{k_0}, \theta_{k_2}\}$, $\{\theta_{k_1}\} \cup \{\theta_{k_2}\} = \{\theta_{k_1}, \theta_{k_2}\}$, and $\{\theta_{k_0}, \theta_{k_1}\} \cup \{\theta_{k_2}\} = \{\theta_{k_0}, \theta_{k_1}, \theta_{k_2}\}$. In essence, when a new singleton is added, new propositions associated with it can be recursively generated by adding the new singleton to each existing proposition. Of course, all propositions of interest within the FoD Θ can be generated by $REGAP(\Theta)$, i.e., when $A = \Theta$.

We refer to this recursive scheme as *REGAP,* which stands for *RE*cursive *G*eneration *of and A*ccess to *P*ropositions. It is illustrated in Fig. 3.1. These recursively generated propositions can be formulated as a vector, a matrix or a tree, and utilized to represent a dynamic BoE.

3.1.1.1 DS-Vector: Vector Representation of a Dynamic BoE

All propositions generated via REGAP can be represented using a dynamic vector, which we refer to as a *DS-Vector.* This is illustrated in Fig. 3.2. It can also be used as a visualization tool of a dynamic BoE.

From a computational point of view, a DS-Vector can be viewed as a dynamic array data structure [83]. Propositions are represented by implicit contiguous indexes, which are considered as implicit bit-strings or decimal integers. So, no memory al-

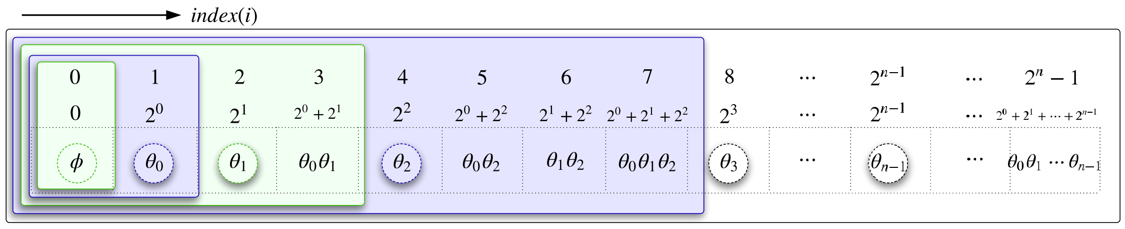


Figure 3.2: **DS-Vector:** Vector representation of a dynamic BoE.

location is needed to store a proposition. Memory allocation is needed only to store the required belief potentials (or, more generally, mass, belief, plausibility, or commonality potentials).

Algorithm 1 provides an implementation to access a belief potential in $\mathcal{O}(m)$ complexity. When the proposition index is available, this becomes a constant time (i.e., $\mathcal{O}(1)$) operation.

Algorithm 1 Access a belief potential in a DS-Vector

```
1: procedure AccessPotential(Singletons $A$)
2:     $index \leftarrow 0$
3:     for each $\theta_i$ in $A$ do
4:         $index \leftarrow index + power[i]$
5:     end for
6:     Return $potential[index]$
7: end procedure
```

The salient steps in the algorithm are as follows:

Line #1: The required proposition can be passed as a bit-string or an integer. If so, this segment has to be replaced by a single input parameter (a bit-string or an integer). In this algorithm, the input parameter is represented in a more general way and passed as A, which includes all the constituent singleton propositions of the proposition of interest A.

Lines #3-5: Implicit index of the proposition is calculated by adding 2 to the power of indexes ($power[i]$) relevant to all the singleton propositions in A.

Line #6: Propositions are represented by implicit indexes and belief potentials are stored in the contiguous memory segments of the DS-Vector. Thus, $potential[index]$ retrieves the respective belief potential.

3.1.1.2 DS-Matrix: Matrix Representation of a Dynamic BoE

All propositions generated via REGAP can also be represented using a dynamic matrix as illustrated in Fig. 3.3. We refer to this as a *DS-Matrix.*

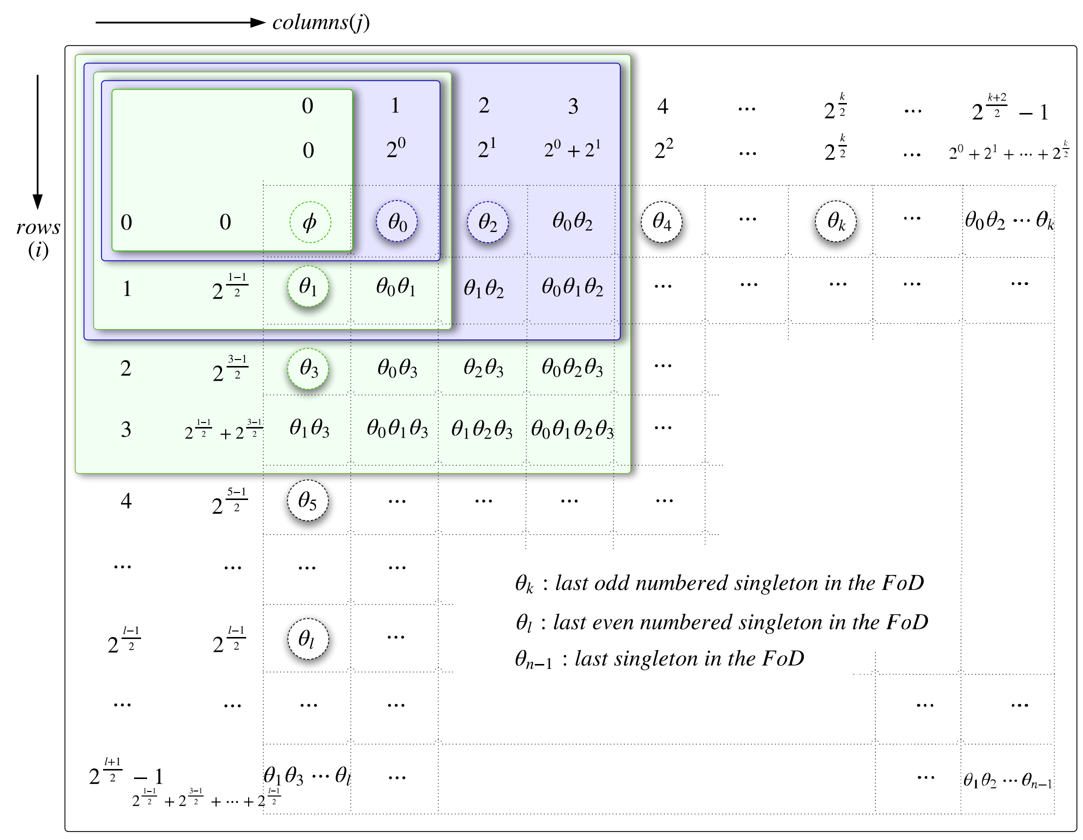

columns(j)
rows (i)
θ_k : last odd numbered singleton in the FoD
θ_l : last even numbered singleton in the FoD
θ_{n−1} : last singleton in the FoD

Figure 3.3: **DS-Matrix:** Matrix representation of a dynamic BoE.

From a computational point of view, the DS-Matrix can be implemented using a dynamic array of dynamic arrays [83]. Propositions are represented by two implicit contiguous indexes (i and j), which are considered as two implicit bit-strings or two decimal integers.

Algorithm 2 provides an implementation to access a belief potential in $\mathcal{O}(m)$ complexity.

Algorithm 2 Access a belief potential in a DS-Matrix

```
1:  procedure AccessPotential(EvenSingletons $A_e$, OddSingletons $A_o$)
2:      $row \leftarrow 0$
3:      $col \leftarrow 0$
4:      for each $\theta_i$ in $A_o$ do
5:          $row = row + power[i]$
6:      end for
7:      for each $\theta_i$ in $A_e$ do
8:          $col = col + power[i]$
9:      end for
10:     Return $potential[row][col]$
11: end procedure
```

The main steps in the algorithm are as follows:

Line #1: Input parameters are passed as A_e and A_o; A_e includes even numbered singletons and A_o includes odd singletons of the proposition of interest A ($A = A_e \cup A_o$).

Lines #4-6: Row index is computed by adding 2 to the power of existing odd singleton indexes ($power[i]$) in A_o.

Lines #7-9: Column index is calculated from the addition of 2 to the power of even singleton indexes ($power[i]$) in A_e.

Line #10: Belief potentials can be accessed by implicit row index and implicit column index; $potential[row][col]$ retrieves the respective belief potential.

3.1.1.3 DS-Tree: Perfectly Balanced Binary Tree Representation of a Dynamic BoE

Another way to represent all propositions generated via REGAP is a perfectly balanced binary tree [83] as illustrated in Fig. 3.4. We refer to this as a *DS-Tree.* When a new singleton is added, the singleton proposition itself stays as the root. Previous propositions stay on the left sub-tree. The new propositions relevant to the incoming singleton are generated by applying REGAP. Those elements stay on the right sub-tree.

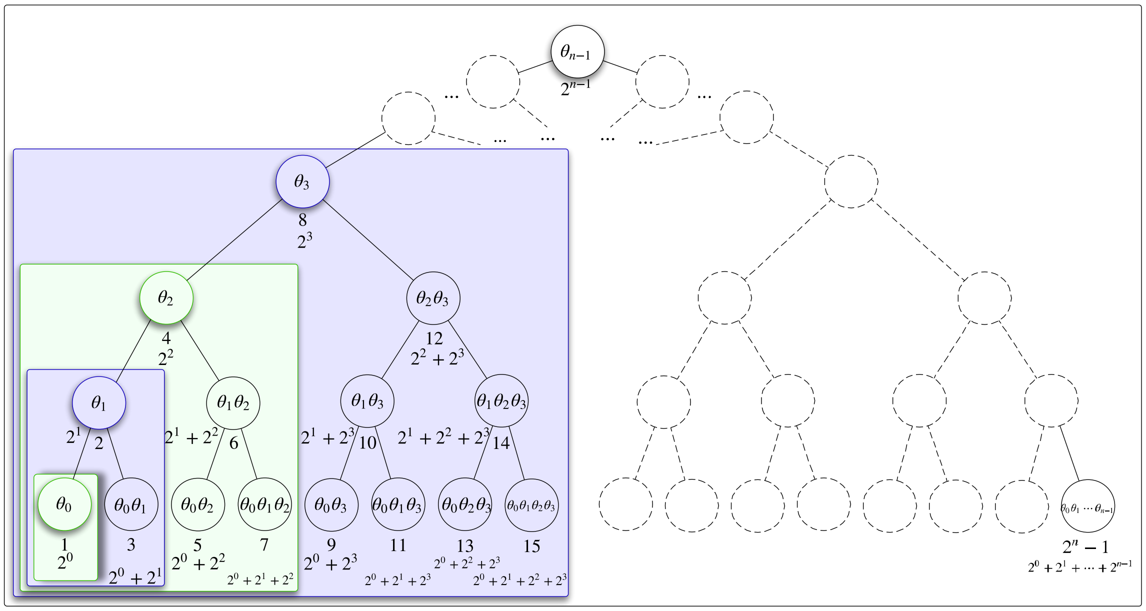


Figure 3.4: **DS-Tree:** Perfectly balanced binary tree representation of a dynamic BoE.

Propositions are represented by relative positions according to the illustration provided in Fig. 3.4. So focal elements can be interpreted as implicit bit-strings or decimal integers. Thus, no memory allocation is needed to store propositions. Only respective belief potentials are contained in the nodes.

Representation of propositions follow the perfectly balanced binary search tree properties [84]. Therefore, it can be implemented using a dynamic array and follows DS-Vector properties. When the DS-Tree complies with classical binary tree implementations, Algorithm 3 provides access to a belief potential in $\mathcal{O}(log(N))$ (or $\mathcal{O}(n)$).

Algorithm 3 Access a belief potential in a DS-Tree

```
1: procedure AccessPotential(FoD $\Theta$, Singletons $A$, DS-Tree $T$)
2:     $index \leftarrow 0$
3:     $level \leftarrow |\Theta| - 1$
4:     $node \leftarrow T.root$
5:     for each $\theta_i$ in $A$ do
6:         $index \leftarrow index + power[i]$
7:     end for
8:     $temp \leftarrow index$
9:     while $temp$ mod $power[level] > 0$ do
10:        if $temp/power[level] = 0$ then
11:            $node \leftarrow node.left$
12:        else if $temp/power[level] = 1$ then
13:            $temp \leftarrow temp - power[level]$
14:            if $temp = 0$ then
15:                break the loop
16:            end if
17:            $node \leftarrow node.right$
18:        end if
19:        $level \leftarrow level - 1$
20:    end while
21:    Return $node.potential$
22: end procedure
```

A belief potential of a proposition can be accessed by traversing through the implicit indexes of the binary tree. The important steps in the algorithm are as follows:

Line #1: Input parameters are the FoD Θ, constituent singleton propositions of the proposition of interest A, and the DS-Tree T.

Lines #5-7: Index of the proposition is computed by adding 2 to the power of singleton indexes ($power[i]$) in A.

Line #8: The binary tree is traversed down, starting from the root node, until the condition $index \bmod power[level] > 0$ is satisfied.

Lines #10-11: Left sub-tree is traversed if the *index* is less than the current implicit index.

Lines #12-18: Right sub-tree is traversed if the *index* is greater than the current implicit index.

Line #21: Required *node* relevant to the proposition is obtained at end of the traversal. So *node.potential* gives the relevant belief potential.

3.2 Efficient Algorithms for Arbitrary Belief Computations

Fast Möbius transform was developed towards addressing the high computational complexity of belief potential calculations [33–35]. However, it is inadequate to make arbitrary belief function computations feasible, especially when working with large FoDs [22]. Employing REGAP, we propose a new approach to identify propositions relevant to a given belief computation. This technique can be used to calculate arbitrary belief, plausibility, and commonality function values from a minimum number of operations.

3.2.1 Belief Calculation

List structure implementation is the commonly utilized approach to store mass potentials [36, 51, 53, 63]. Belief calculation requires cycling through the list structure to recognize whether each focal element should be included in the computation. Thus, computational complexity of this operation is $\mathcal{O}(|\mathfrak{F}|)$.

The REGAP strategy offers an alternative to generate the required propositions relevant to the computation of $Bl(A)$, where $A \subseteq \Theta$. In this method, belief computation is performed by accessing only the subset propositions. The maximum number of subset propositions that one would have to access is about $M = 2^m$, where $m = |A|$.

3.2.1.1 DS-Vector

Algorithm 4 provides an implementation to compute a belief potential in $\mathcal{O}(M)$ (or $\mathcal{O}(2^m)$) complexity.

Algorithm 4 Computing Belief in a DS-Vector

```
1: procedure ComputeBelief(Singletons $A$, Normalize $Nlz$)
2:     $belief \leftarrow 0$
3:     $count \leftarrow 0$
4:     for each $\theta_i$ in $A$ do
5:         $index[count] \leftarrow power[i]$
6:         $temp \leftarrow count$
7:         $count \leftarrow count + 1$
8:         for $j \leftarrow 0, temp - 1$ do
9:             $index[count] \leftarrow index[j] + power[i]$
10:            $count \leftarrow count + 1$
11:        end for
12:    end for
13:    for $i \leftarrow 0, power[|A|] - 2$ do
14:        $belief \leftarrow belief + potential[index[i]]$
15:    end for
16:    Return $belief/Nlz$
17: end procedure
```

The main steps in the algorithm are as follows:

Line #1: Input parameters are A and normalizing constant Nlz. Normalizing constant is the summation of all the mass potential values. mass potential are stored in raw values to improve the performance.

Lines #4-12: Subset propositions of A are generated by applying REGAP.

Lines #13-15: Belief is the summation of the potentials of relevant generated implicit indexes. Computational complexity of an iteration is $\mathcal{O}(1)$ (access operation).

Line #16: Normalized belief potential is the output parameter of the procedure.

3.2.1.2 DS-Matrix

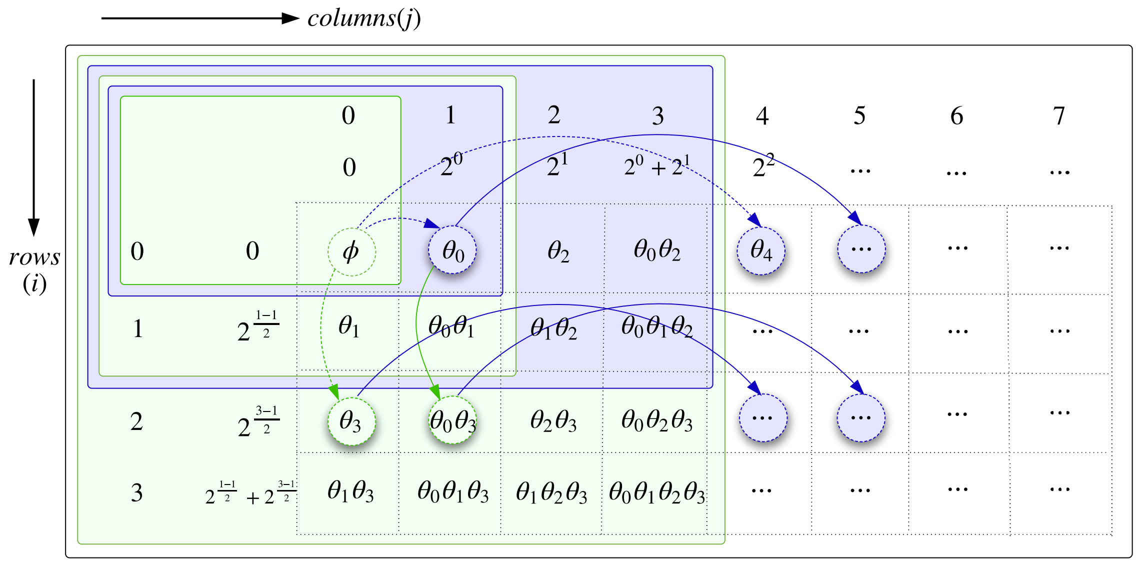


Figure 3.5: **Belief Calculation:** Propositions related to $Bl(A)$ computation when $A = \{\theta_0, \theta_3, \theta_4\}$, and $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_4, \theta_5, \}$.

Fig. 3.5 provides an illustration of the belief computation when $A = \{\theta_0, \theta_3, \theta_4\}$, and $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_4, \theta_5, \}$.

Algorithm 5 can be used to compute a belief potential in $\mathcal{O}(M)$ (or $\mathcal{O}(2^m)$) complexity.

Algorithm 5 Computing Belief in a DS-Matrix

```
1: procedure ComputeBelief(SingletonCoordinates $A_P$, Normalize $Nlz$)
2:     $belief \leftarrow 0$
3:     $count \leftarrow 0$
4:     for each pair $p$ in $A_P$ do
5:         $index[count].row \leftarrow p.row$
6:         $index[count].col \leftarrow p.col$
7:         $temp \leftarrow count$
8:         $count \leftarrow count + 1$
9:         for $j \leftarrow 0, temp - 1$ do
10:            $index[count].row \leftarrow index[j].row + p.row$
11:            $index[count].col \leftarrow index[j].col + p.col$
12:            $count \leftarrow count + 1$
13:        end for
14:    end for
15:    for $i \leftarrow 0$, $power[|A_P|] - 2$ do
16:        $belief \leftarrow belief$
17:        $+potential[index[i].row][index[i].col]$
18:    end for
19:    Return $belief/Nlz$
20: end procedure
```

The salient steps in the algorithm are as follows:

Line #1: Input parameters are A_P and normalizing constant Nlz. A_P contains row and column coordinate pairs of all the singleton propositions of the proposition of interest A.

Lines #4-14: Subset propositions of A are obtained by applying REGAP.

Lines #15-18: Belief is the summation of the potentials relevant to the generated implicit index pairs. Computational complexity of an iteration is $\mathcal{O}(1)$ (access operation).

Line #19: Procedure returns the normalized belief potential.

3.2.1.3 DS-Tree

Algorithm 6 can be used to compute a belief potential in $\mathcal{O}(M\log(N))$ (or $\mathcal{O}(2^m n)$) complexity when the DS-Tree is implemented using node structures. Dynamic array implementation of the DS-Tree complies with algorithm 4 and the belief computation complexity is $\mathcal{O}(M)$ (or $\mathcal{O}(2^m)$).

Algorithm 6 Computing Belief in a DS-Tree

```
1: **procedure** ComputeBelief(Singletons $A$, DSTree $T$, Normalize $Nlz$)
2:     $belief \leftarrow 0$
3:     $count \leftarrow 0$
4:     **for** each $\theta_i$ in $A$ **do**
5:         $index[count] \leftarrow power[i]$
6:         $temp \leftarrow count$
7:         $count \leftarrow count + 1$
8:         **for** $j \leftarrow 0, temp - 1$ **do**
9:             $index[count] \leftarrow index[j] + power[i]$
10:            $count \leftarrow count + 1$
11:        **end for**
12:    **end for**
13:    **for** $i \leftarrow 0, power[|A|] - 2$ **do**
14:        $level \leftarrow |\Theta| - 1$
15:        $leaf \leftarrow T.root$
16:        $index \leftarrow index[i]$
17:        **while** $index$ mod $power[level] > 0$ **do**
18:            **if** $index/power[level] = 0$ **then**
19:                $leaf \leftarrow leaf.left$
20:            **else if** $index/power[level] = 1$ **then**
21:                $index \leftarrow index - power[level]$
22:                **if** $index = 0$ **then**
23:                    break the loop
24:                **end if**
25:                $leaf \leftarrow leaf.right$
26:            **end if**
27:            $level \leftarrow level - 1$
28:        **end while**
29:        $belief \leftarrow belief + leaf.mass$
30:    **end for**
31:    Return $belief/Nlz$
32: **end procedure**
```

The salient steps in the algorithm are as follows:

Line #1: Input parameters are the proposition of interest A, DSTree T, and normalizing constant Nlz.

Lines #4-12: Subset propositions of A obtained by applying REGAP.

Lines #13-30: Belief is the summation of the potentials of generated implicit indexes. Computational complexity of a tree traversal iteration is $\mathcal{O}(\log(N))$ (or $\mathcal{O}(n)$) and follows the Algorithm 3.

Line #31: Normalized belief potential is the output parameter of the procedure.

3.2.2 Plausibility and Commonality Calculation

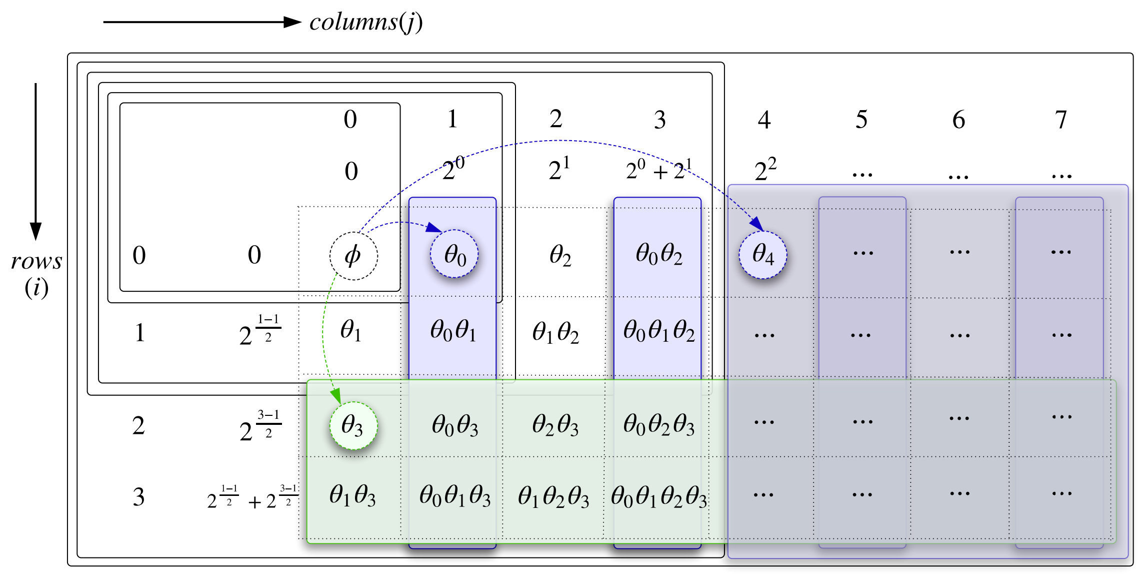


Figure 3.6: **Plausibility Calculation:** Propositions related to $Pl(A)$ computation when $A = \{\theta_0, \theta_3, \theta_4\}$, and $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_4, \theta_5, \}$.

Plausibility $Pl(A)$ can be computed by observing that $Pl(A) = 1 - Bl(\overline{A})$ and applying a belief computation algorithm to $\overline{A}$. Propositions relevant to commonality $Q(A)$ calculation can be generated by applying REGAP to $\overline{A}$ and adding proposition

A to all the generated propositions. In this way, computations can also be performed with minor modifications to Algorithms 4, 5, and 6. Computational complexity remains the same as for belief computation algorithms.

Fig. 3.6 illustrates the plausibility computation when $A = \{\theta_0, \theta_3, \theta_4\}$, and $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_4, \theta_5, \}$. This figure clarifies the complexity of plausibility computations over belief computations. There are 28 propositions for direct $Pl(A)$ computation. Computing $Bl(\overline{A})$ and using the equation $Pl(A) = 1 - Bl(\overline{A})$ to compute $Pl(A)$, accesses only 3 propositions: $m(\theta_1)$; $m(\theta_2)$; and $m(\theta_1\theta_2)$.

3.3 Experiments

We have developed a novel belief computation library, which we refer to as *BCL (Belief Computation Library)* using the C++ programming language [62]. This includes the implementation of data structures and algorithms for a generalized computational framework, in particular, the three representations *DS-Vector, DS-Matrix,* and *DS-Tree,* for representing DST models and carrying out DST operations. All experiments were simulated on a Macintosh desktop computer (iMac) running Mac OS X 10.11.3, with 2.9GHz Intel Core i5 processor and 8GB of 1600MHz DDR3 RAM.

Average computational times for accessing arbitrary propositions are listed in Table 3.1 for different implementations: DS-Vector in algorithm 1, DS-Matrix in algorithm 2, and common list structure implementation. Results were obtained by executing the algorithms for 100000 randomly chosen propositions from the FoD and noting the average CPU time. A random set of focal elements were generated in the core for each FoD size.

FoD Size	**Max. $\|\mathfrak{F}\|$**	**DS-Vector**	**DS-Matrix**	**List Struct.**
2	3	0.379	0.393	0.465
4	15	0.400	0.412	0.510
6	63	0.410	0.454	0.739
8	255	0.443	0.449	1.541
10	1023	0.433	0.496	4.632
12	4095	0.465	0.493	16.906
14	16383	0.465	0.527	67.242
16	65535	0.495	0.517	268.443
18	262143	0.529	0.560	1124.060
20	1048575	0.575	0.629	4609.370

Table 3.1: CPU time of accessing a proposition (μs)

From Table 3.1, the speed advantage of the three proposed implementations over the commonly used list structure implementation is quite clear. With increasing FoD size, the access times for DS-Vector and DS-Matrix almost remain the same when compared to the rapid growth of the computational times in list structure implementation.

Average computational times of a randomly chosen belief computation using the algorithm 4, algorithm 5, and list structures are given in Table 3.2. Results were obtained by executing the algorithms for 100000 randomly chosen propositions and noting the average CPU time. A random set of focal elements were generated in the core for each FoD size.

The proposed implementations offer better results when compared with the commonly used list structure implementation. With increasing FoD size, the average computational time of a randomly chosen belief function increases very slowly with DS-Vector and DS-Matrix in comparison to the list structure implementation. The DS-Tree provides the same experimental results as DS-Vector.

FoD Size	**Max. $\|\mathfrak{F}\|$**	**DS-Vector**	**DS-Matrix**	**List Struct.**
2	3	0.373	0.362	0.450
4	15	0.378	0.376	0.531
6	63	0.415	0.450	0.833
8	255	0.453	0.508	1.779
10	1023	0.525	0.663	5.529
12	4095	0.655	0.923	20.757
14	16383	0.884	1.314	81.196
16	65535	1.340	2.159	325.930
18	262143	2.107	3.510	1373.110
20	1048575	3.963	6.210	5448.170

Table 3.2: CPU time of a belief computation (μs)

3.4 Concluding Remarks

This chapter provides a general framework along with data structures and efficient algorithms for DST computations. The data structures presented can also serve as tools for BoE visualization. We believe that the proposed implementations constitute a significant step forward in harnessing the strengths of DS theory in practical application scenarios.

The implicit index calculation mechanism that we introduce for the purpose of representing a proposition serves to reduce the memory usage and to significantly improve computational efficiency. All the indexes are calculated according to relative positions in the data structures. Only belief potentials need to be stored. Memory usage efficiency is greatly improved since representing the proposition as a bit-string or an integer is unnecessary.

When the index of the proposition is available, the proposed algorithms for accessing a proposition take a constant time irrespective of the FoD's size. Therefore updating a belief potential (or a mass potential) in a BoE can be executed in significantly less time. The proposed REGAP strategy is invaluable in that it allows one to identify the exact subsets relevant to a given belief computation. Efficient algorithms for belief calculation of arbitrary propositions are also developed by ensuring that only a minimum possible number of operations are executed.

Representation of DS-Tree propositions follow the perfectly balanced binary search tree properties. Therefore, it can be implemented using a dynamic array and can gain the performance relevant to a DS-Vector. As an outcome of this research work, a belief computation library (BCL) in C++ which includes all the important operations to work with belief computations is made available [62]. This will be useful for a significant performance improvement for applications based on DST methods.

CHAPTER 4

DS-Conditional-One: Efficient and Exact Computation of Arbitrary Conditionals

The main contribution of this chapter is a new generalized model for computing DST *conditionals.* This conditional computational model — *DS-Conditional-One* — offers significantly greater flexibility and computational capability for implementation of DST conditional strategies. This model can be employed to compute both the FH and Dempster's conditional beliefs of an *arbitrary* proposition [57]. By reducing the number of operations being executed, the proposed approach takes significantly less computational and space complexity when compared with other approaches for conditional computation.

This chapter is organized as follows: Section 4.1 provides a review of essential DST notions and computational tools; Our DS-Conditional-One computational model and our algorithms for efficient computation of DST conditionals appear next in Sections 4.2 and 4.3, respectively; The experimental results are provided next in Section 4.4; Finally, Section 4.5 offers some concluding remarks.

4.1 Theoretical Foundation

The following notations will be useful for our work:

$$\mathcal{S}(A;B) = \sum_{\substack{\emptyset \neq C \subseteq A;\\ \emptyset \neq D \subseteq B}} m(C \cup D). \tag{4.1}$$

So, $\mathcal{S}(A;B)$ denotes the sum of all mass values of propositions that 'straddle' both $A \subseteq \Theta$ and $B \subseteq \Theta$.

$$\mathcal{T}(A;B) = \sum_{C \subseteq A} m(C \cup B). \tag{4.2}$$

$\mathcal{T}(A;B)$ denotes the sum of all mass values of $A \subseteq \Theta$ propositions that 'straddle' strictly proposition $B \subseteq \Theta$.

We utilize the important results of the following proposition for the development of computational models and for carrying out both the Dempster's and FH conditional computations.

Proposition 1 *Consider the BoE $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ and $A \subseteq \Theta$. For $B \subseteq \Theta$, consider the mappings $\Gamma_A : 2^{\Theta} \mapsto [0,1]$ and $\Pi_A : 2^{\Theta} \mapsto [0,1]$, where*

$$\Gamma_A(B) = \sum_{\emptyset \neq X \subseteq \overline{A}} m((A \cap B) \cup X); \; \Pi_A(B) = \sum_{Y \subseteq (A \cap B)} \Gamma_A(Y).$$

Then the following are true:

(i) $\Gamma_A(A \cap B) = \Gamma_A(B)$ *and* $\Pi_A(A \cap B) = \Pi_A(B)$*. So, w.l.o.g., we may assume that* $B \subseteq A$*.*

(ii) $\Gamma_A(\emptyset) = \Pi_A(\emptyset) = Bl(\overline{A})$*.*

(iii) $\Gamma_A(B) = \mathcal{T}(\overline{A}; A \cap B) - m(A \cap B)$*.*

(iv) $\Pi_A(B) = \Pi_A(\emptyset) + \mathcal{S}(\overline{A}; A \cap B)$*.* □

Proof: **(i)**, **(ii)** and **(iii)** follow by direct substitution. To show **(iv)**, note that

$$\Pi_A(B) = \sum_{\substack{Y \subseteq (A \cap B) \\ \emptyset \neq X \subseteq \overline{A}}} m((A \cap Y) \cup X).$$

When $Y = \emptyset$, the right-hand side yields $\Gamma_A(\emptyset) = \Pi_A(\emptyset) = Bl(\overline{A})$; else, it yields $\mathcal{S}(\overline{A}; A \cap B)$. This establishes **(iv)**. ■

In our work, we will exploit several previous results related to the conditional core [59, 85]. Of particular importance are the following results:

Lemma 1 ([85]) *Consider the BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ *and* $A \in \widehat{\mathfrak{F}}$. *Then, the following are true:*

(i) $m(B|A) = 0$ *whenever* $\overline{A} \cap B \neq \emptyset$, *and*

(ii) $Bl(B|A)$ *can be expressed as*

$$Bl(B|A) = \frac{Bl(A \cap B)}{Pl(A) - \mathcal{S}(\overline{A}; A \cap B)},\ B \subseteq A. \quad ■$$

Note that, **(i)** states that FH conditioning annuls those propositions that 'straddle' the conditioning proposition A and its complement $\overline{A}$. So, w.l.o.g., for FH conditioning, one may consider only those propositions $B \subseteq A$.

For our work, we will need the following alternate expression for the FH conditional:

Proposition 2 *Consider the BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ *and* $A \in \widehat{\mathfrak{F}}$. *Then, we may express* $Bl(B|A)$ *as*

$$Bl(B|A) = \frac{Bl(A \cap B)}{1 - Bl(\overline{A}) - \mathcal{S}(\overline{A}; A \cap B)},\ B \subseteq \Theta. \quad □$$

Proof: This follows directly from Lemma 1**(ii)** by using the fact that $Bl(A) = 1 - Pl(\overline{A})$. ■

For our work, we will need the following alternate expressions for the Dempster's conditional:

Proposition 3 *Consider the BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ *and* $A \subseteq \Theta$ *s.t.* $Bl(\overline{A}) \neq 1$. *Then,* $Bl(B\|A)$ *can be expressed as*

$$Bl(B\|A) = \frac{Bl(A \cap B) + \mathcal{S}(\overline{A}; A \cap B)}{1 - Bl(\overline{A})}, \; B \subseteq \Theta. \qquad \square$$

Proof: This follows directly from Definition 5 by using the fact that $Bl(\overline{A} \cup B) = Bl(\overline{A} \cup (A \cap B)) = Bl(\overline{A}) + Bl(A \cap B) + \mathcal{S}(\overline{A}; A \cap B)$. ■

Propositions 2 and 3 highlight an important fact: the three quantities $Bl(\overline{A})$, $Bl(A \cap B)$, and $\mathcal{S}(\overline{A}; A \cap B)$ fully determine both FH and Dempster's conditionals $Bl(B|A)$ and $Bl(B\|A)$, respectively. It is this fact that we exploit for computing the conditional belief of an arbitrary proposition.

Proposition 4 *Consider the BoE* $\mathcal{E} = \{\Theta, \mathfrak{F}, m(\cdot)\}$ *and* $A \subseteq \Theta$ *s.t.* $Bl(\overline{A}) \neq 1$. *Then,* $m(B\|A)$ *can be expressed as*

$$m(B\|A) = \begin{cases} \dfrac{\mathcal{T}(\overline{A}; A \cap B)}{1 - Bl(\overline{A})}, & \textit{for } \emptyset \neq B \subseteq A; \\ 0, & \textit{otherwise.} \end{cases}$$

Proof: This follows by direct substitution. ■

The propositions recursively generated via the REGAP property can be represented as a vector, DS-Vector, a matrix, DS-Matrix, or a tree, DS-Tree, and utilized to capture a BoE. We will utilize this REGAP property and the DS-Matrix structure in this work too.

4.2 DS-Conditional-One Computational Model

DS-Conditional-One is a computational model that enables one to compute the FH and Dempster's conditional beliefs of an *arbitrary* proposition. This model can also be used to compute Dempster's conditional masses of an *arbitrary* proposition. DS-Conditional-One model facilitates the representation, access, and efficient computation of the quantities that are needed to compute these conditionals (see Propositions 2, 3 and 4).

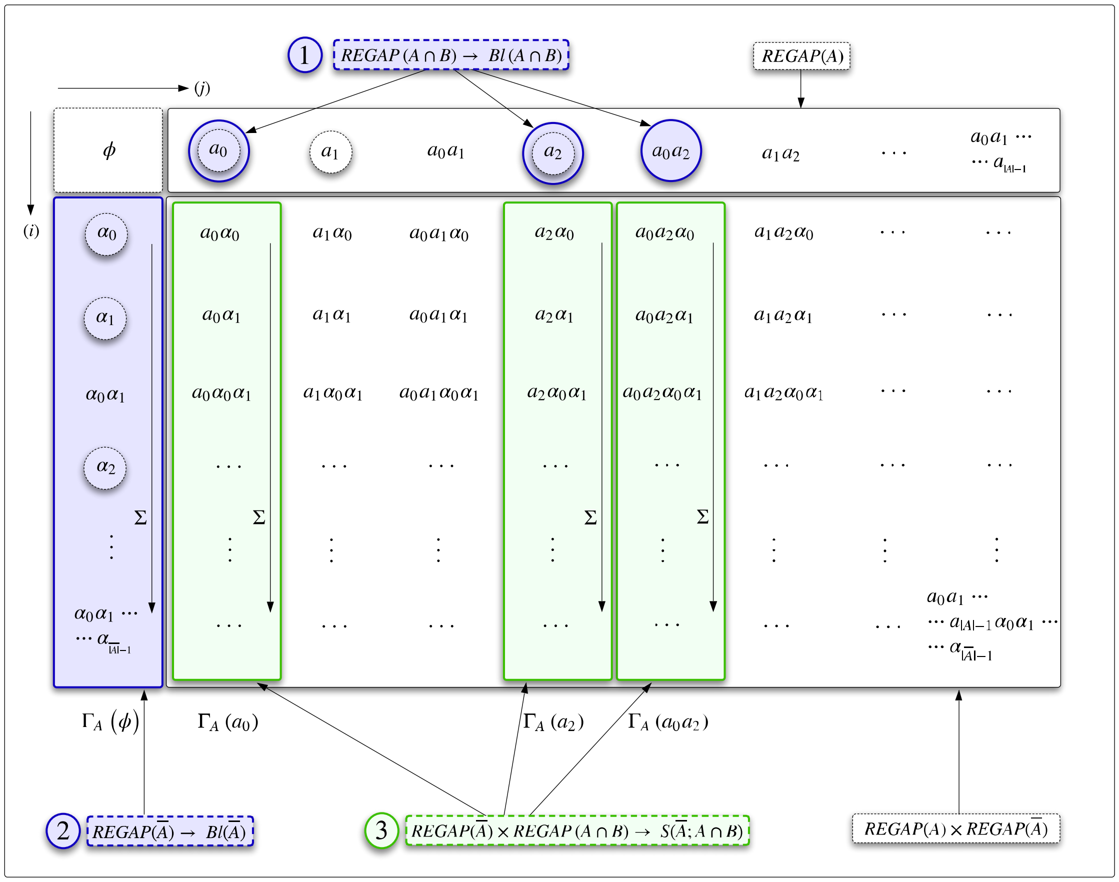

1
$REGAP(A \cap B) \rightarrow Bl(A \cap B)$
$REGAP(A)$
(j)
(i)
ϕ
a_0
a_1
a_0a_1
a_2
a_0a_2
a_1a_2
$a_0a_1 \cdots \cdots a_{|A|-1}$
α_0
α_1
$\alpha_0\alpha_1$
α_2
$\alpha_0\alpha_1 \cdots \cdots \alpha_{|\overline{A}|-1}$
Σ
$\Gamma_A(\phi)$
$\Gamma_A(a_0)$
$\Gamma_A(a_2)$
$\Gamma_A(a_0a_2)$
2
$REGAP(\overline{A}) \rightarrow Bl(\overline{A})$
3
$REGAP(\overline{A}) \times REGAP(A \cap B) \rightarrow S(\overline{A}; A \cap B)$
$REGAP(A) \times REGAP(\overline{A})$

Figure 4.1: **DS-Conditional-One model.** Quantities related to arbitrary $Bl(B|A)$ (or $Bl(B\|A)$) computation when $A = \{a_0, a_1, \ldots, a_{|A|-1}\}$, $\overline{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$, and $B = \{a_0, a_2\} \subseteq A$.

Henceforth, we will denote the conditioning proposition A, its complement $\overline{A}$, and the conditioned proposition B as $\{a_0, a_1, \ldots, a_{|A|-1}\}$, $\{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$, and $\{b_0, b_1, \ldots, b_{|B|-1}\}$, respectively. Here, $\Theta = \{\theta_0, \theta_1, \ldots, \theta_{n-1}\}$ denotes the FoD and $a_i, \alpha_j, b_k \in \Theta$. When dealing with FH and Dempster's conditioning, it is implicitly assumed that $A \in \widehat{\mathfrak{F}}$ and $Bl(\overline{A}) \neq 1$, respectively.

Furthermore, we will represent singletons of the conditioning event $A = \{a_0, a_1, \ldots, a_{|A|-1}\}$ as *column singletons* and singletons of the complement of conditioning event $\overline{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$ as *row singletons* in a DS-Matrix. See Fig. 4.1.

The proposed DS-Conditional-One computational model allows direct identification of $REGAP(A)$, $REGAP(\overline{A})$, $REGAP(A \cap B)$, $(REGAP(\overline{A}) \times REGAP(A \cap B))$, $(REGAP(\overline{A}) \times REGAP(A))$, $(REGAP(\overline{A}) \times B)$, and $\Gamma_A(C)$, $\forall C \subseteq B$. Among these, the following three quantities are required to compute both FH and Dempster's conditional belief of an arbitrary proposition (see Propositions 2 and 3):

(a) $REGAP(A \cap B)$: Use this to compute $Bl(A \cap B)$ (see Algorithm 7).

(b) $REGAP(\overline{A})$ which is $\Gamma_A(\emptyset)$: Use this to compute $Bl(\overline{A})$ (see Algorithm 8).

(c) $(REGAP(\overline{A}) \times REGAP(A \cap B))$, the Cartesian product of $REGAP(\overline{A})$ and $REGAP(A \cap B)$: Use this to compute $\mathcal{S}(\overline{A}; A \cap B)$ (see Algorithm 9).

Fig. 4.1 depicts these quantities for $A = \{a_0, a_1, \ldots, a_{|A|-1}\}$, $\overline{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$, and $B = \{a_0, a_2\} \subseteq A$.

The following quantity is required to compute Dempster's conditional mass of an arbitrary proposition (see Proposition 4):

$(A \cap B) + (REGAP(\overline{A}) \times (A \cap B))$, the Cartesian product of $REGAP(\overline{A})$ and $(A \cap B)$: Use this to compute $\mathcal{T}(\overline{A}; A \cap B)$ (see Algorithm 10).

In the algorithms to follow, we use a lookup table named *power* to enhance the computational efficiency. It contains 2 to the power of singleton indexes in increasing order and it is implemented using a dynamic array that replaces run-time computation of 2 to the power values with a simpler array indexing operation. The J-th entry of the *power* table, $power[J]$, refers to 2^J. A dynamic array, $index[]$, keeps the indexes of subset propositions of $A \cap B$.

4.2.1 Time Complexity of Algorithm 7

Algorithm 7 Compute $Bl(A \cap B)$ ($\mathcal{O}(2^{|A \cap B|})$)

```
1: procedure BLB(Singletons $A$, Singletons $B$, DS-Matrix $BBA[][]$)
2:     $belief b \leftarrow 0$
3:     $count \leftarrow 0$
4:     for each $a_J$ in $A \cap B$ do
5:         $index[count] \leftarrow power[J]$
6:         $temp \leftarrow count$
7:         $count \leftarrow count + 1$
8:         for $t \leftarrow 0, temp - 1$ do
9:             $index[count] \leftarrow index[t] + power[J]$
10:            $count \leftarrow count + 1$
11:        end for
12:    end for
13:    for $t \leftarrow 0, power[|A \cap B|] - 2$ do
14:        $belief b \leftarrow belief b + BBA[0][index[t]]$
15:    end for
16:    Return $belief b$
17: end procedure
```

This computes $Bl(A \cap B)$ in $\mathcal{O}(2^{|A \cap B|})$ complexity.

Line #1: The algorithm inputs are the conditioning event A, conditioned event B, and the DS-Matrix $BBA[][]$.

Lines #4-12: The outer loop is executed ($|A \cap B|$) times.

Lines #8-11: The inner loop is executed ($temp$) times. It can be shown that for $\ell = 0, 1, 2, \ldots, |A \cap B| - 1$, $temp = (2^{\ell} - 1)$.

Lines #5 and #9 are constant time operations. Thus, the computational complexity of *lines #4-12* is given by

$$\begin{aligned} \sum_{\ell=0}^{|A\cap B|-1} (1 + temp) &= \sum_{\ell=0}^{|A\cap B|-1} 2^{\ell} \\ &= 2^{|A\cap B|} - 1 = \mathcal{O}(2^{|A\cap B|}). \end{aligned} \tag{4.3}$$

Lines #13-15: The required number of iterations is $(2^{|A\cap B|}-1)$ and the complexity of this segment is $\mathcal{O}(2^{|A\cap B|})$.

Line #16: The algorithm output is $Bl(A \cap B)$.

4.2.2 Time Complexity of Algorithm 8

Algorithm 8 Compute $Bl(\overline{A})$ ($\mathcal{O}(2^{|\overline{A}|})$)

```
1: procedure BLACOMP(Singletons $\overline{A}$, DS-Matrix $BBA[][]$)
2:     $beliefacomp \leftarrow 0$
3:     for $i \leftarrow 1, power[|\overline{A}|] - 1$ do
4:         $beliefacomp \leftarrow beliefacomp + BBA[i][0]$
5:     end for
6:     Return $beliefacomp$
7: end procedure
```

This computes $Bl(\overline{A})$ in $\mathcal{O}(2^{|\overline{A}|})$ complexity.

Line #1: The algorithm inputs are the complement of conditioning event $\overline{A}$ and the DS-Matrix $BBA[][]$.

Lines #3-5: The required number of iterations is $(2^{|\overline{A}|}-1)$ and the computational complexity of this segment is $\mathcal{O}(2^{|\overline{A}|})$.

Line #6: The algorithm output is the belief potential $Bl(\overline{A})$.

4.2.3 Time Complexity of Algorithm 9

Algorithm 9 Compute $\mathcal{S}(\overline{A}; A \cap B)$ ($\mathcal{O}(2^{|\overline{A}|+|A\cap B|})$)

```
1: procedure STRAD(Singletons $\overline{A}$, Singletons $A$, Singletons $B$, DS-Matrix $BBA[][]$)
2:     $strad \leftarrow 0$
3:     $count \leftarrow 0$
4:     for each $a_J$ in $A \cap B$ do
5:         $index[count] \leftarrow power[J]$
6:         $temp \leftarrow count$
7:         $count \leftarrow count + 1$
8:         for $t \leftarrow 0, temp - 1$ do
9:             $index[count] \leftarrow index[t] + power[J]$
10:            $count \leftarrow count + 1$
11:        end for
12:    end for
13:    for $t \leftarrow 0, power[|A \cap B|] - 2$ do
14:        for $i \leftarrow 1, power[|\overline{A}|] - 1$ do
15:            $strad \leftarrow strad + BBA[i][index[t]]$
16:        end for
17:    end for
18:    Return $strad$
19: end procedure
```

This computes $\mathcal{S}(\overline{A}; A \cap B)$ in $\mathcal{O}(2^{|\overline{A}|+|A\cap B|})$ complexity.

Line #1: The algorithm inputs are the complement of conditioning event $\overline{A}$, the conditioning and conditioned propositions A and B, respectively, and the DS-Matrix $BBA[][]$.

Lines #4-12: Subset propositions of $A \cap B$ are generated via $REGAP(A \cap B)$. Computational complexity of this segment is $\mathcal{O}(2^{|A\cap B|})$, which can be obtained from equation 4.3.

Lines #13-17: The outer loop is executed $(2^{|A\cap B|} - 1)$ times. *Lines #14-16:* The inner loop is executed $(2^{|\overline{A}|} - 1)$ times. Complexity of an access operation is

$\mathcal{O}(1)$. Thus, the computational complexity of *lines #13-17* is $(2^{|\overline{A}|} - 1)\,(2^{|A \cap B|} - 1) = \mathcal{O}(2^{|\overline{A}|+|A\cap B|})$.

Line #18: The algorithm output is $\mathcal{S}(\overline{A}; A \cap B)$.

4.2.4 Time Complexity of Algorithm 10.

Algorithm 10 Compute $\mathcal{T}(\overline{A}; A \cap B)$ $(\mathcal{O}(max(2^{|\overline{A}|}, |A \cap B|)))$

```
1: procedure StradStrictB(Singletons $\overline{A}$, Singletons $A$, Singletons $B$, DS-Matrix $BBA[][]$)
2:     $stradstrictb \leftarrow 0$
3:     $t \leftarrow 0$
4:     for each $a_J$ in $A \cap B$ do
5:         $t \leftarrow t + power[J]$
6:     end for
7:     for $i \leftarrow 0, power[|\overline{A}|] - 1$ do
8:         $stradstrictb \leftarrow stradstrictb + BBA[i][t]$
9:     end for
10:    Return $stradstrictb$
11: end procedure
```

This computes $\mathcal{T}(\overline{A}; A\cap B)$ in $\mathcal{O}(max(2^{|\overline{A}|}, |A\cap B|))$ complexity. When the proposition index $(A \cap B)$ is available, this becomes $\mathcal{O}(2^{|\overline{A}|})$.

Line #1: The algorithm inputs are the complement of conditioning event $\overline{A}$, the conditioning and conditioned propositions A and B, respectively, and the DS-Matrix $BBA[][]$.

Lines #4-6: Implicit index of the proposition is calculated by adding 2 to the power of indexes ($power[J]$) relevant to all singleton propositions in $A \cap B$. The required number of iterations is $(|A \cap B|)$ and the computational complexity of this segment is $\mathcal{O}(|A\cap B|)$. When the proposition index $(A\cap B)$ is available, this becomes a constant time $(\mathcal{O}(1))$ operation.

Lines #7-9: The required number of iterations is $(2^{|\overline{A}|})$ and the computational complexity of this segment is $\mathcal{O}(2^{|\overline{A}|})$.

Line #10: The algorithm output is $\mathcal{T}(\overline{A}; A \cap B)$.

4.2.5 Space Complexity of Algorithms 7, 8, 9, and 10.

The matrix in Fig. 4.1 is of size $2^{|A|} \times 2^{|\overline{A}|}$. Hence, the space complexity associated with each algorithm above is $\mathcal{O}(2^{|\Theta|})$.

Note that in DS-Conditional-One model, $REGAP(A)$ captures all propositions that may contribute to the conditional core $\mathfrak{F}_A$. $REGAP(\overline{A})$ and $(REGAP(\overline{A}) \times REGAP(A))$ which is the Cartesian product of $REGAP(\overline{A})$ and $REGAP(A)$, capture all propositions whose masses are annulled (as identified by Lemma 1 [85]). See Fig 4.1.

4.3 Efficient Computation of DST Conditionals

In this section, we discuss efficient computation of *arbitrary* DST conditionals using the — *DS-Conditional-One* — computational model (See Fig. 4.1). This model can be employed to compute both the FH and Dempster's conditional beliefs of an *arbitrary* proposition, as well as to compute Dempster's conditional masses of an *arbitrary* proposition.

4.3.1 Computation of the FH Conditional Belief of an Arbitrary Proposition

To compute the FH conditional belief of an arbitrary proposition B, one can now use the expression in Proposition 2, where $Bl(A \cap B)$, $Bl(\overline{A})$ and $\mathcal{S}(\overline{A}; A \cap B)$ are obtained via Algorithms 7, 8, and 9, respectively. Thus the computational complexity of this computation remains as $\mathcal{O}(2^{|\overline{A}|+|A \cap B|})$.

As an example, to compute $Bl(B|A)$, where $B = \{a_0, a_2\}$, we may proceed as follows:

(a) $REGAP(A \cap B)$ captures the propositions that contribute to $Bl(A \cap B)$. Use Algorithm 7 to compute this.

(b) $REGAP(\overline{A})$ captures the propositions that contribute to $Bl(\overline{A})$. Use Algorithm 8 to compute this. Note that $Bl(\overline{A})$ is represented by $\Gamma_A(\emptyset)$ in Fig. 4.1.

(c) The Cartesian product $(REGAP(\overline{A}) \times REGAP(A \cap B))$ captures the propositions that contribute to $\mathcal{S}(\overline{A}; A \cap B)$. Use Algorithm 9 to compute this.

$\mathcal{S}(\overline{A}; A \cap B) = \Gamma_A(\{a_0\}) + \Gamma_A(\{a_2\}) + \Gamma_A(\{a_0, a_2\})$.

Then, $Bl(B|A)$ for $B = \{a_0, a_2\}$ is computed as

$$Bl(B|A) = \frac{Bl(A \cap B)}{1 - \Gamma_A(\{\emptyset\}) - \Gamma_A(\{a_0\}) - \Gamma_A(\{a_2\}) - \Gamma_A(\{a_0, a_2\})}. \tag{4.4}$$

4.3.2 Computation of the Dempster's Conditional Belief of an Arbitrary Proposition

To compute the Dempster's conditional belief of an arbitrary proposition B, one can use the expression in Proposition 3, where $Bl(A \cap B)$, $Bl(\overline{A})$ and $\mathcal{S}(\overline{A}; A \cap B)$ are obtained via Algorithms 7, 8, and 9, respectively. Thus the computational complexity is $\mathcal{O}(2^{|\overline{A}|+|A\cap B|})$.

Consider the same example as before, viz., $B = \{a_0, a_2\}$. Then, we may compute $Bl(B\|A)$ as

$$\begin{aligned} &Bl(B\|A) = \\ &\frac{Bl(A\cap B) + \Gamma_A(\{a_0\}) + \Gamma_A(\{a_2\}) + \Gamma_A(\{a_0, a_2\})}{1 - \Gamma_A(\{\emptyset\})}. \end{aligned} \tag{4.5}$$

4.3.3 Computation of the Dempster's Conditional Mass of an Arbitrary Proposition

To compute the Dempster's conditional mass of an arbitrary proposition B, one can use the expression in Proposition 4, where $Bl(\overline{A})$ and $\mathcal{T}(\overline{A}; A\cap B)$ are obtained via Algorithms 8, and 10, respectively. Thus the computational complexity is $\mathcal{O}(max(2^{|\overline{A}|}, |A\cap B|)$.

Consider the same example as before, viz., $B = \{a_0, a_2\}$. Then, to compute $m(B\|A)$ we may proceed as follows:

(a) $REGAP(\overline{A})$ captures the propositions that contribute to $Bl(\overline{A})$. Use Algorithm 8 to compute this.

(b) The Cartesian product $(REGAP(\overline{A}) \times (A\cap B)) + (A\cap B)$ captures the propositions that contribute to $\mathcal{T}(\overline{A}; A\cap B)$, for $\emptyset \neq B \subseteq A$. Use Algorithm 10 to compute this. When $\emptyset = B \subseteq A$, $m(B\|A) = 0$.

$\mathcal{T}(\overline{A}; A\cap B) = \Gamma_A(\{a_0, a_2\}) + m(\{a_0, a_2\})$.

Then, $m(B\|A)$ for $B = \{a_0, a_2\}$ is computed as

$$m(B\|A) = \frac{m(A\cap B) + \Gamma_A(\{a_0, a_2\})}{1 - \Gamma_A(\{\emptyset\})}. \tag{4.6}$$

Computation of the Dempster's Conditional Mass Using Specialization Matrix. It is noteworthy that [43] and [46] have proposed a matrix calculus based algorithm for *direct* computation of Dempster's conditional *masses.* It employs a $2^{|\Theta|} \times 2^{|\Theta|}$-sized stochastic matrix $\mathfrak{S}_A$ (with each entry '0' or '1') referred to as the conditioning specialization matrix and a $2^{|\Theta|} \times 1$-sized vector $m(\cdot)$ containing the BoE's focal elements. Then $m(\cdot \| A) = \mathfrak{S}_A \cdot m(\cdot)$ yields Dempster's conditioning masses *without normalization.* The computational and space complexity of the specialization matrix multiplication is $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$, a prohibitive burden even for modest FoD sizes.

4.4 Experiments

Recall that Algorithms 7, 8, and 9 yield all the parameters (viz., $Bl(A \cap B)$, $Bl(\overline{A})$, and $\mathcal{S}(\overline{A}; A \cap B)$) required for both FH and Dempster's conditional belief computations. Once these quantities are computed, computational times for both conditional belief computations are similar because they require constant time (see Propositions 2 and 3).

For a given FoD size, we selected a random set of focal elements, with randomly selected mass values, and conducted 10,000 conditional computations for randomly chosen propositions A and $B \subseteq A$. Table 4.1 lists the average computational times taken by the DS-Conditional-One model and the specialization matrix based method in [43] and [46].

With the DS-Conditional-One model (which applies to *both* FH and Dempster's conditionals), we use a 'brute force' approach to compute all the conditional beliefs (i.e., compute the conditional belief of every proposition); we then use the FMT to get the conditional masses for all the propositions [14, 40]. The specialization matrix based method (which applies to the Dempster's conditional *only*) yields the conditional masses of *all* propositions, but the time taken already far exceeds what the DS-Conditional-One model takes (even including the FMT). So we did not compute the conditional beliefs with the specialization matrix based method (which would have required the FMT).

All conditional computations for an *arbitrary* proposition were done on an iMac running Mac OS X 10.12.3 (with 2.9GHz Intel Core i5 processor and 8GB of 1600MHz DDR3 RAM). Conditional computations for *all* propositions were done on the same iMac for smaller FoDs and on a supercomputer (`http://ccs.miami.edu/pegasus`) for larger FoDs (underlined in Table 4.1). The complete C++ library (CCL) is available at [65].

Method →		**DS-Conditional-One Model**			**Specialization Matrix**
Conditional →		**FH or Dempster's**			**Dempster's**
FoD		$Bl(B\|A)$ or $Bl(B\|\|A)$	$Bl(B\|A)$ or $Bl(B\|\|A)$	$m(B\|A)$ or $m(B\|\|A)$	$m(B\|\|A)$
$\|\Theta\|$	**Max.** $\|\mathfrak{F}\|$	**(Arbitrary)**	**(All)**	**(All)**	**(All)**
2	3	0.0005	0.0011	0.0016	0.0011
4	15	0.0005	0.0038	0.0050	0.0063
6	63	0.0006	0.0128	0.0170	0.0696
8	255	0.0009	0.0517	0.0679	1.0154
10	1,023	0.0017	0.2428	0.3090	$\underline{93.1590}$
12	4,095	0.0040	1.3528	1.6186	$\underline{1{,}485.6300}$
14	16,383	0.0120	$\underline{18.4885}$	$\underline{22.4995}$	$\underline{25{,}051.8200}$
16	65,535	0.0405	$\underline{146.1480}$	$\underline{151.9600}$	$\underline{***}$
18	262,143	0.1516	$\underline{1{,}087.2800}$	$\underline{1{,}113.5300}$	$\underline{***}$
20	1,048,575	0.6011	$\underline{8{,}485.4500}$	$\underline{8{,}862.9800}$	$\underline{***}$

Table 4.1: DS-Conditional-One model versus specialization matrix based method. Average computational times (ms) (*** denotes computations not completed within a feasible time).

The significant speed advantage offered by the proposed computational model over the specialization matrix based approach is evident from Table 4.1. For larger FoDs, the computational burden associated with the specialization matrix based approach becomes prohibitive because of its space complexity of $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$. For example, an FoD of size 20 would need 128 $(= 2^{20} \times 2^{20}/8)$ GB of memory to represent the specialization matrix, if each matrix entry occupies only 1 bit. With increasing FoD size, the computational time requirement of the DS-Conditional-One model is significantly less compared to what the specialization matrix based approach requires.

4.5 Concluding Remarks

This chapter provides a general framework for computation of DST conditionals. The DS-Conditional-One model that we propose can also serve as a tool for visualization and further analysis of the conditional computation process. Our experiment results demonstrate that the average computational time taken to compute the conditional belief of an arbitrary proposition by the proposed approach is less than 2 (μs) for a frame of size 10 and 0.7 (ms) for a frame of size 20 ($\sim$1 million focal elements).

The efficiency of these algorithms is mainly because of the significantly reduced number of operations that are executed. Computational complexity associated with conditional belief computation of an arbitrary proposition is $\mathcal{O}(2^{|\overline{A}|+|A\cap B|})$. This is a significant improvement over the $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$ complexity associated with the specialization matrix based approach. The DS-Conditional-One model also provides a significant advantage in terms of memory usage: it requires a $\mathcal{O}(2^{|\Theta|})$ space complexity versus $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$ for the specialization matrix based approach.

Another advantage of the proposed approach is that it can be utilized for either the FH conditional or Dempster's conditional belief computations. An outcome of this research is *CCL (Conditional Computation Library)* which is available at [65]. We believe that the algorithms we have developed constitute a significant step forward in harnessing the strengths of DST methods in practical applications.

CHAPTER 5

DS-Conditional-All: Efficient and Exact Computation of All Conditionals

The main contribution of this chapter is a novel scalable, generalized computational model — *DS-Conditional-All* — for computing all DST *conditionals.* This model offers greater flexibility and computational capability for implementation of DST conditional strategies. It is a significantly better approach to conditional computations in contrast to the currently available methods. DS-Conditional-All can also be utilized as a visualization tool for conditional computations and in analyzing characteristics of conditioning and updating operations.

Computation of *all* FH and Dempster's conditional beliefs are discussed using this model. It can also be used to compute Dempster's conditional masses of *all* propositions. We provide the complexity analysis, experimental validation of the utility, efficiency, implementation of the associated data structures and algorithms.

This chapter is organized as follows: Section 5.1 introduces the conditional computational framework; Section 5.2 contains algorithms for efficient computation of all DST conditionals; Section 5.3 contains comparisons and experimental results; Section 5.4 provides the concluding remarks.

5.1 DS-Conditional-All Computational Model

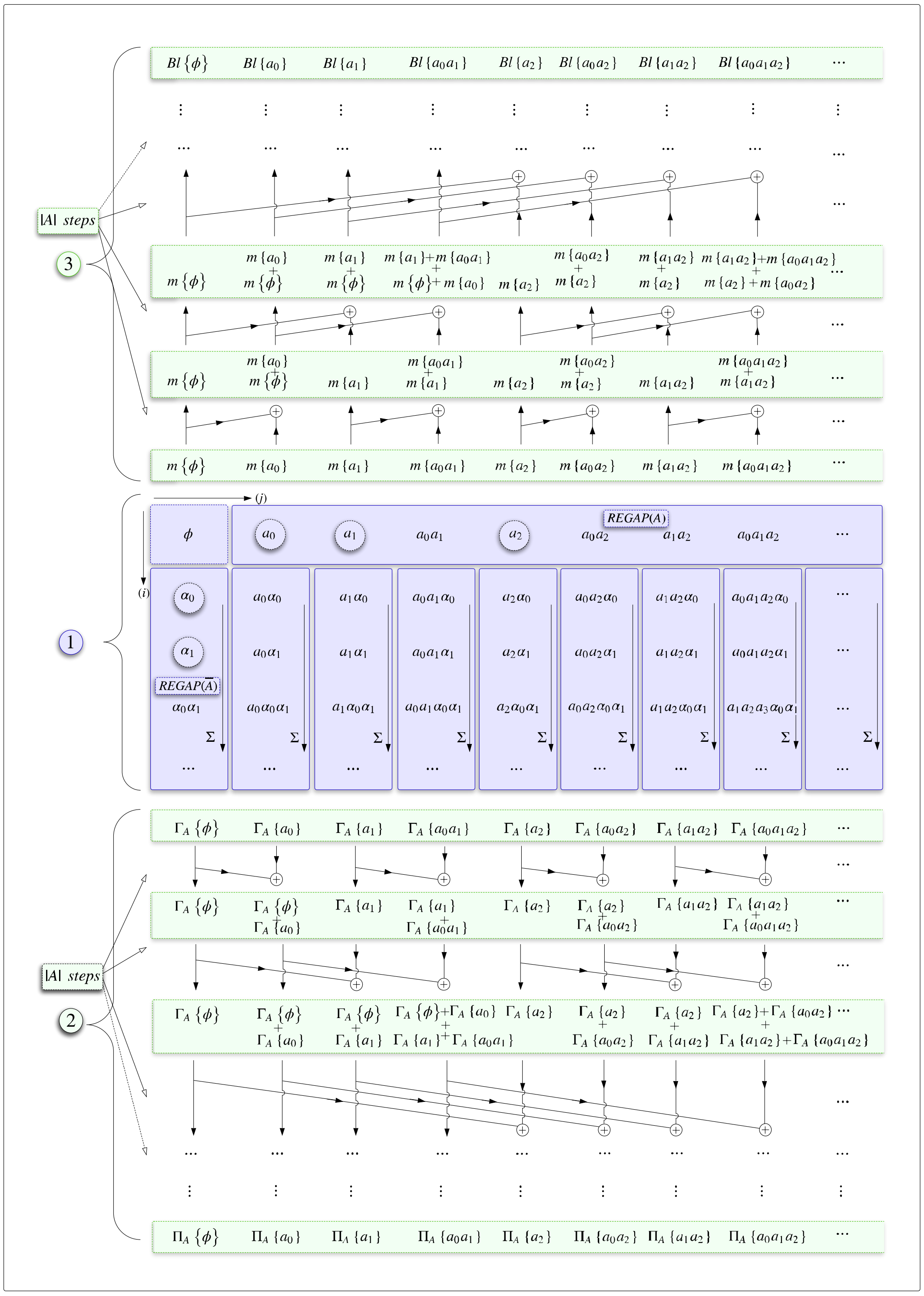
The propositions recursively generated via the REGAP property can be represented as a vector, DS-Vector, a matrix, DS-Matrix, or a tree, DS-Tree, and utilized to capture a BoE. We will utilize the REGAP property, the DS-Matrix and DS-Vector structures in this work too. We also utilize the theoretical derivations obtained in Section 4.1.

In this chapter, we develop the *DS-Conditional-All* computational model that enables one to compute the FH and Dempster's conditional beliefs of *all* propositions. This model can also be used to compute Dempster's conditional masses of *all* propositions. DS-Conditional-All model facilitates the representation, access, and efficient computation of the quantities that are needed to compute these conditionals (see Propositions 1, 2, 3 and 4).

For this purpose, we utilize the results in [33–35, 50] which provide a fast Möbius transform, which is analogous to the fast Fourier transform (FFT), to convert a BBA to its corresponding belief potential (as in Definition 2) or to convert a belief potential to its corresponding BBA (as in (2.2)). When one notices the relationship between $\Gamma(\cdot)$ to $\Pi(\cdot)$ (see Proposition 1) and the relationship between a BBA and its corresponding belief (see Definition 2), it is clear that this same fast Möbius transform can be used to convert $\Gamma(\cdot)$ to $\Pi(\cdot)$, or vice versa.

We will represent singletons of the conditioning event $A = \{a_0, a_1, \ldots, a_{|A|-1}\}$ as *column singletons* and singletons of the complement of conditioning event $\overline{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$ as *row singletons* in a DS-Matrix. See Fig. 5.1.

The proposed DS-Conditional-All computational model allows direct identification of $REGAP(A)$, $REGAP(\overline{A})$, $(REGAP(\overline{A}) \times REGAP(A))$, $(REGAP(\overline{A}) \times B)$,



$Bl\{\phi\}$ $Bl\{a_0\}$ $Bl\{a_1\}$ $Bl\{a_0a_1\}$ $Bl\{a_2\}$ $Bl\{a_0a_2\}$ $Bl\{a_1a_2\}$ $Bl\{a_0a_1a_2\}$
|A| steps
3
$m\{\phi\}$ $m\{a_0\}$ $m\{a_1\}$ $m\{a_0a_1\}$ $m\{a_2\}$ $m\{a_0a_2\}$ $m\{a_1a_2\}$ $m\{a_0a_1a_2\}$
(j)
(i)
1
ϕ a_0 a_1 a_0a_1 a_2 a_0a_2 a_1a_2 $a_0a_1a_2$
REGAP(A)
α_0 α_1 $REGAP(\overline{A})$ $\alpha_0\alpha_1$
Σ
2
$\Gamma_A\{\phi\}$ $\Gamma_A\{a_0\}$ $\Gamma_A\{a_1\}$ $\Gamma_A\{a_0a_1\}$ $\Gamma_A\{a_2\}$ $\Gamma_A\{a_0a_2\}$ $\Gamma_A\{a_1a_2\}$ $\Gamma_A\{a_0a_1a_2\}$
$\Pi_A\{\phi\}$ $\Pi_A\{a_0\}$ $\Pi_A\{a_1\}$ $\Pi_A\{a_0a_1\}$ $\Pi_A\{a_2\}$ $\Pi_A\{a_0a_2\}$ $\Pi_A\{a_1a_2\}$ $\Pi_A\{a_0a_1a_2\}$

Figure 5.1: **DS-Conditional-All model.** Quantities related to $Bl(B|A)$ (or $Bl(B\|A)$) computation for *all* $B \subseteq A$ when $A = \{a_0, a_1, \ldots, a_{|A|-1}\}$, and $\overline{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$.

$\Gamma_A(B)$, $\forall B \subseteq A$ and $\Pi_A(B)$, $\forall B \subseteq A$. Among these, the following four quantities are required to compute both FH and Dempster's conditional belief of all propositions, as well as to compute Dempster's conditional mass of all propositions (see Propositions 1, 2, 3 and 4):

(a) $REGAP(\overline{A})$: Use this to compute $Bl(\overline{A})$ which is $\Gamma_A(\emptyset)$ or $\Pi_A(\emptyset)$ (see Algorithm 8, or $\Gamma_A(\emptyset)$ can be obtained from the output of Algorithm 11).

(b) $(REGAP(\overline{A}) \times B)$, $\forall B \subseteq A$: Add the BBA of each column except the BBA of top element to compute $\Gamma_A(B)$, $\forall B \subseteq A$. Section 1 of Fig. 5.1 shows this computation (see Algorithm 11).

(c) Use $REGAP(A)$ to identify the propositions relevant for $\Pi_A(B)$, $\forall B \subseteq A$, computation and apply the fast Möbius transform to get the $\Pi_A(\cdot)$ values from $\Gamma_A(B)$, $\forall B \subseteq A$. Section 2 of Fig. 5.1 illustrates this transformation (see Algorithm 12).

(d) Use $REGAP(A)$ and apply the fast Möbius transform to get the belief values $Bl(B)$, $B \subseteq A$, from the BBA $m(B)$, $\forall B \subseteq A$. Section 3 of Fig. 5.1 illustrates this transformation (steps of this computation are similar to Algorithm 12).

Fig. 5.1 depicts these quantities related to $Bl(B|A)$ (or $Bl(B\|A)$) computation for *all* $B \subseteq A$ when $A = \{a_0, a_1, \ldots, a_{|A|-1}\}$, and $\overline{A} = \{\alpha_0, \alpha_1, \ldots, \alpha_{|\overline{A}|-1}\}$.

5.1.1 Time Complexity of Algorithm 11

This computes all Γ_A in $\mathcal{O}(2^{|\Theta|})$ complexity.

Line #1: The algorithm inputs are the complement of conditioning event $\overline{A}$, conditioning event A, and the DS-Matrix $BBA[][]$.

Algorithm 11 Compute all Γ_A ($\mathcal{O}(2^{|\Theta|})$)

```
1:  procedure ALLΓ_A(Singletons $\overline{A}$, Singletons $A$, DS-Matrix $BBA[][]$)
2:      for $j \leftarrow 0$, $power[|A|] - 1$ do
3:          $\Gamma_A[j] \leftarrow 0$
4:      end for
5:      for $j \leftarrow 0$, $power[|A|] - 1$ do
6:          for $i \leftarrow 1$, $power[|\overline{A}|] - 1$ do
7:              $\Gamma_A[j] \leftarrow \Gamma_A[j] + BBA[i][j]$
8:          end for
9:      end for
10:     Return $\Gamma_A[]$
11: end procedure
```

Lines #2-4: The required number of iterations is ($2^{|A|}$) and the complexity of this segment is $\mathcal{O}(2^{|A|})$.

Lines #5-9: The outer loop is executed ($2^{|A|}$) times. *Lines #6-8:* The inner loop is executed ($2^{|\overline{A}|} - 1$) times. Complexity of an access operation is $\mathcal{O}(1)$. Thus, the computational complexity of *lines #5-9* is $(2^{|\overline{A}|} - 1)\,(2^{|A|}) = \mathcal{O}(2^{|\overline{A}|+|A|}) = \mathcal{O}(2^{|\Theta|})$.

Line #10: The algorithm output is $\Gamma_A[]$.

5.1.2 Space Complexity of Algorithm 11

The matrix in Fig. 5.1 is of size $2^{|A|} \times 2^{|\overline{A}|}$. Hence, the space complexity associated with the above algorithm is $\mathcal{O}(2^{|\Theta|})$.

5.1.3 Time Complexity of Algorithm 12

This computes all Π_A in $\mathcal{O}(2^{|A|} \times |A|)$ complexity.

Line #1: The algorithm inputs are the conditioning event A, and the DS-Vector $\Gamma_A[]$.

Algorithm 12 Compute all Π_A ($\mathcal{O}(2^{|\Theta|} \times |\Theta|)$)

```
1: procedure AllΠ_A(Singletons $A$, DS-Vector $\Gamma_A[]$)
2:     for $j \leftarrow 0$, $power[|A|] - 1$ do
3:         $\Pi_A[j] \leftarrow \Gamma_A[j]$
4:     end for
5:     for $J \leftarrow 0$, $|A| - 1$ do
6:         for $t \leftarrow 0$, $power[|A| - J] - 2$ step 2 do
7:             for $s \leftarrow 0$, $power|J| - 1$ do
8:                 $\Pi_A[(t+1) * power[J] + s] \leftarrow \Pi_A[(t+1) * power[J] + s] + \Pi_A[t * power[J] + s]$
9:             end for
10:        end for
11:    end for
12:    Return $\Pi_A[]$
13: end procedure
```

Lines #2-4: The required number of iterations is $(2^{|A|})$ and the complexity of this segment is $\mathcal{O}(2^{|A|})$.

Lines #5-11: The outer loop is executed $(|A|)$ times. *Lines #6-10:* The middle loop is executed $(2^{|A|-J-1})$ times. *Lines #7-9:* The inner loop is executed (2^J) times. Complexity of an access operation is $\mathcal{O}(1)$. Thus, the computational complexity of *lines #5-11* is $(|A|)(2^{|A|-J-1})\,(2^J) = \mathcal{O}(2^{|A|} \times |A|)$.

Line #12: The algorithm output is $\Pi_A[]$.

5.1.4 Space Complexity of Algorithm 12

The $\Gamma_A[]$ and $\Pi_A[]$ vectors in Fig. 5.1 is of size $2^{|A|}$. Hence, the space complexity associated with the above algorithm is $\mathcal{O}(2^{|A|})$.

Note that in DS-Conditional-All model, $REGAP(A)$ captures all propositions that may contribute to the conditional core $\mathfrak{F}_A$. $REGAP(\overline{A})$ and $(REGAP(\overline{A}) \times REGAP(A))$ which is the Cartesian product of $REGAP(\overline{A})$ and $REGAP(A)$, capture all propositions whose masses are annulled (as identified by Lemma 1 [85]). See Fig 5.1.

5.2 Efficient Computation of All DST Conditionals

In this section, we discuss efficient computation of *all* DST conditionals using the — *DS-Conditional-All* — computational model (See Fig. 5.1). This model can be employed to compute both the FH and Dempster's conditional beliefs of *all* propositions, as well as to compute Dempster's conditional masses of *all* propositions. FH conditional masses of *all* propositions can be obtained via applying fast Möbius transform to computed FH conditional beliefs of *all* propositions.

5.2.1 Computation of the FH Conditional Beliefs of All Propositions

DS-Conditional-All model provides $\Pi_A(A \cap B)$ and $Bl(A \cap B)$ for all $B \subseteq A$ via Algorithms 11, and 12. Now use the expressions in Proposition 1 and 2 to obtain $Bl(B|A)$ as

$$Bl(B|A) = \frac{Bl(A \cap B)}{1 - \Pi_A(A \cap B)}. \tag{5.1}$$

We may compute all conditional belief values by iterating on the propositions in $REGAP(A)$ and applying the above equation.

5.2.2 Computation of the Dempster's Conditional Beliefs of All Propositions

As before, we can obtain $\Pi_A(A \cap B)$ and $Bl(A \cap B)$ for all $B \subseteq A$ via Algorithms 11, and 12. Then use the expressions in Proposition 1 and 3 to obtain $Bl(B \| A)$ as

$$Bl(B \| A) = \frac{Bl(A \cap B) + \Pi_A(A \cap B) - \Gamma_A(\{\emptyset\})}{1 - \Gamma_A(\{\emptyset\})}. \tag{5.2}$$

We can compute all conditional belief values by iterating on the propositions in $REGAP(A)$ and applying the above equation.

5.2.3 Computation of the FH Conditional Masses of All Propositions

FH conditional masses of *all* propositions can be obtained via applying fast Möbius transform to computed FH conditional beliefs of *all* propositions. All FH conditional beliefs can be computed as the discussion in Sec. 5.2.1.

5.2.4 Computation of the Dempster's Conditional Masses of All Propositions

We can obtain $\Gamma_A(A \cap B)$ for all $B \subseteq A$ via Algorithm 11. Then use the expressions in Proposition 1 and 4 to obtain $m(B \| A)$ as

$$m(B \| A) = \frac{m(A \cap B) + \Gamma_A(A \cap B)}{1 - \Gamma_A(\{\emptyset\})}. \tag{5.3}$$

All conditional masses are computed by iterating on the propositions in $REGAP(A)$ and applying the above equation. The computational complexity of iterating over

$REGAP(A)$ is $\mathcal{O}(2^{|A|})$. $m(A \cap B)$ can be accessed in $\mathcal{O}(1)$. Also after applying Algorithm 11, we can access $\Gamma_A(\{\emptyset\})$ in $\mathcal{O}(1)$. Thus the computational complexity of computing all conditional masses is $\mathcal{O}(2^{|\Theta|})$.

Computation of the Dempster's Conditional Mass Using the Specialization Matrix. It is noteworthy that [43] and [46] have proposed a matrix calculus based algorithm for *exact* computation of Dempster's conditional *masses.* It employs a $2^{|\Theta|} \times 2^{|\Theta|}$-sized stochastic matrix $\mathfrak{S}_A$ (with each entry '0' or '1') referred to as the conditioning specialization matrix and a $2^{|\Theta|} \times 1$-sized vector $m(\cdot)$ containing the BoE's focal elements. Then $m(\cdot \| A) = \mathfrak{S}_A \cdot m(\cdot)$ yields Dempster's conditioning masses *without normalization.* The computational and space complexity of the specialization matrix multiplication is $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$, a prohibitive burden even for modest FoD sizes.

5.3 Experiments

We have developed a novel computation library, which we refer to as *DS-CONAC* *(<u>DS-C</u>onditional-<u>O</u>ne a<u>n</u>d DS-Conditional-<u>A</u>ll in <u>C</u>++)* [67]. This includes the implementation of required data structures and algorithms of proposed computational models along with simulation tools. All conditional computations for smaller FoDs were simulated on a Macintosh desktop computer (iMac) running Mac OS X 10.13.4 (with 2.9GHz Intel Core i5 processor and 8GB of 1600MHz DDR3 RAM). Conditional computations for larger FoDs were done on a supercomputer (`http://ccs.miami.edu/pegasus`) (<u>underlined</u> in Table 5.2).

Note that Algorithms 7, 8, and 9 yield all the parameters (viz., $Bl(A \cap B)$, $Bl(\overline{A})$, and $\mathcal{S}(\overline{A}; A \cap B)$) required for both FH and Dempster's conditional belief computations of an *arbitrary* proposition. Similarly, Algorithms 11, and 12 yield all the parameters (viz., $Bl(A \cap B)$, $\Gamma_A(\{\emptyset\})$, and $\Pi_A(A \cap B)$) required for both FH and Dempster's conditional belief computations of *all* propositions. Once these quantities are computed, computational times for both conditional belief computations (FH and Dempster's) are similar because the final operations require only a constant time (see Propositions 1, 2 and 3).

Results were obtained by executing the algorithms for 10000 randomly chosen conditioning (A) and conditioned ($B \subseteq A$) propositions from the FoD and noting the average CPU time. A random set of focal elements were generated in the core for each FoD size.

With the DS-Conditional-One model (which applies to *both* FH and Dempster's conditionals), we first compute arbitrary conditional beliefs to compute average computational times it takes; we next use a 'brute force' approach to compute all the conditional beliefs (i.e., compute the conditional belief of every proposition); we then use the FMT to get the conditional masses for all the propositions [14, 40]. We also compute arbitrary Dempster's conditional masses to measure the average computational times. Table 5.1 lists the average computational times taken by the DS-Conditional-One model.

Table 5.2 lists the average computational times taken by the DS-Conditional-All model and the specialization matrix based method in [43] and [46].

With the DS-Conditional-All model (which applies to *both* FH and Dempster's conditionals), we compute all the conditional beliefs; we then use the FMT to get the conditional masses for all the propositions. We also compute all Dempster's conditional masses (directly, without FMT) to determine the average computational

Method →		**DS-Conditional-One Model**			
Conditional →		**FH or Dempster's**			**Dempster's**
FoD		$Bl(B\vert A)$ or $Bl(B\Vert A)$	$Bl(B\vert A)$ or $Bl(B\Vert A)$	$m(B\vert A)$ or $m(B\Vert A)$	$m(B\Vert A)$
$\vert\Theta\vert$	**Max.** $\vert\mathfrak{F}\vert$	**(Arbitrary)**	**(All)**	**(All)**	**(Arbitrary)**
2	3	0.0008	0.0016	0.0024	0.0008
4	15	0.0008	0.0057	0.0068	0.0008
6	63	0.0009	0.0189	0.0208	0.0009
8	255	0.0011	0.0707	0.0758	0.0010
10	1,023	0.0016	0.3038	0.3208	0.0012
12	4,095	0.0033	1.5535	1.6206	0.0016
14	16,383	0.0095	15.0000	17.1429	0.0030
16	65,535	0.0323	131.8750	136.8750	0.0074
18	262,143	0.1223	1,072.2200	1,077.7800	0.0218
20	1,048,575	0.4724	8,670.0000	8,698.0000	0.0771
22	4,194,303	3.1889	71,115.9000	73,942.3000	0.2853
24	16,777,215	18.7807	653,268.0000	660,883.0000	0.6467
26	67,108,863	83.0787	1.6334 cpu hours	1.6915 cpu hours	1.1744
28	268,435,455	338.2960	***	***	31.2735
30	1,073,741,823	1,509.5000	***	***	111.2910

Table 5.1: DS-Conditional-One model. Average computational times with DS-CONAC library (ms) (*** denotes computations not completed within a feasible time or space requirement).

Method →		**DS-Conditional-All Model**			**Specialization Mat.**
Conditional →		**FH or Dempster's**		**Dempster's**	**Dempster's**
FoD		$Bl(B\|A)$ or $Bl(B\\|A)$	$m(B\|A)$ or $m(B\\|A)$	$m(B\\|A)$	$m(B\\|A)$
$\|\Theta\|$	**Max.** $\|\mathfrak{F}\|$	**(All)**	**(All)**	**(All)**	**(All)**
2	3	0.0011	0.0014	0.0015	0.0015
4	15	0.0014	0.0018	0.0020	0.0070
6	63	0.0026	0.0034	0.0031	0.0767
8	255	0.0067	0.0091	0.0054	1.1264
10	1,023	0.0211	0.0303	0.0135	98.4795
12	4,095	0.0770	0.1133	0.0427	1,581.8300
14	16,383	0.2950	0.4378	0.1532	24,847.0000
16	65,535	1.1592	1.7243	0.5814	396,860.0000
18	262,143	6.5901	9.2096	2.3430	1.7637 cpu hours
20	1,048,575	26.7221	39.0397	9.3537	***
22	4,194,303	112.4180	166.0070	43.5348	***
24	16,777,215	500.3420	689.8700	233.6080	***
26	67,108,863	2,239.2400	2,908.7000	1,118.9500	***
28	268,435,455	9,273.8100	12,406.4000	4,976.9700	***
30	1,073,741,823	42,087.2000	52,055.8000	25,354.9000	***

Table 5.2: DS-Conditional-All model versus specialization matrix based method. Average computational times (ms) (*** denotes computations not completed within a feasible time or space requirement).

times it takes. The specialization matrix based method (which applies to the Dempster's conditional *only*) yields the conditional masses of *all* propositions, but the time taken already far exceeds what the DS-Conditional-One model as well as the DS-Conditional-All model takes (even including the FMT). So we did not compute the conditional beliefs with the specialization matrix based method (which would have required the FMT). The DS-Conditional-All model provide better results for conditional computations of all propositions than the DS-Conditional-One model.

Method →		**DS-Conditional-One and All**			**Specialization Mat.**
Conditional →			**FH or**		
		Dempster's	**Dempster's**	**Dempster's**	**Dempster's**
			$Bl(B\|A)$ or		
FoD		$m(B\|\|A)$	$Bl(B\|\|A)$	$m(B\|\|A)$	$m(B\|\|A)$
$\|\Theta\|$	**Max.** $\|\mathfrak{F}\|$	**(Arbitrary)**	**(Arbitrary)**	**(All)**	**(All)**
2	3	0.0003	0.0004	0.0005	0.0008
4	15	0.0004	0.0004	0.0007	0.0128
6	63	0.0004	0.0006	0.0017	0.2048
8	255	0.0006	0.0009	0.0058	3.2768
10	1023	0.0010	0.0017	0.0218	52.4288
12	4095	0.0021	0.0042	0.0737	838.8608
14	16383	0.0054	0.0120	0.2755	13421.7728
16	65535	0.0145	0.0413	1.0831	214748.3648
18	262143	0.0440	0.1558	4.4525	0.95 cpu hours
20	1048575	0.1509	0.6129	18.2815	15.27 cpu hours*
22	4194303	3.1688	6.6125	369.7840	244.33 cpu hours*
24	16777215	10.8877	28.2246	1546.8100	3909.37 cpu hours*
26	67108863	40.1846	125.2820	6440.7600	7.14 cpu years*
28	268435455	132.3020	496.1380	25180.4000	114.24 cpu years*
30	1073741823	463.2340	2329.1100	101009.0000	1827.94 cpu years*

Table 5.3: DS-Conditional-One model and DS-Conditional-All model versus specialization matrix based method. Theoretical computational times (*ms*) (* denotes that computation becomes prohibitive exceeding the memory limit and could not be completed within a feasible time).

The significant speed advantage offered by the proposed computational models over the specialization matrix based approach is evident from Table 5.1 and Table 5.2. For larger FoDs, the computational burden associated with the specialization matrix based approach becomes prohibitive because of its time complexity as well as the space complexity ($\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$). For example, an FoD of size 22 would need 2 ($= 2^{22} \times 2^{22}/8$) TB of memory to represent the specialization matrix, if each matrix entry occupies only 1 bit. This is a prohibitive space requirement for practical applications. For large FoDs, the computational time requirement of specialization matrix based approach rapidly becomes infeasible. This is clearly evident from Table 5.3.

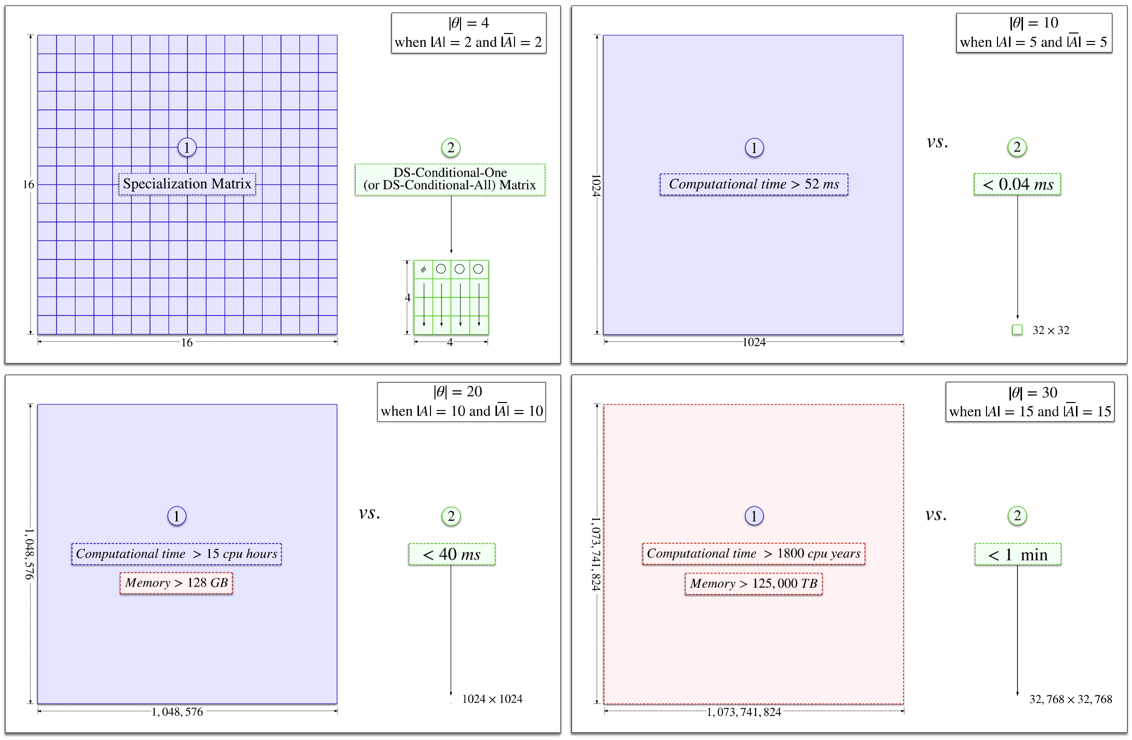
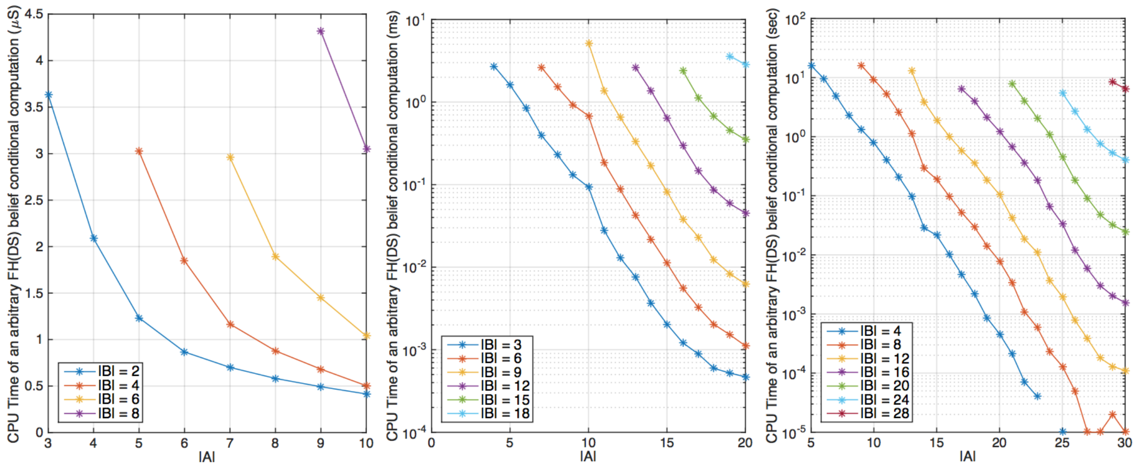


Figure 5.2: Time and space complexity comparison of DS-Conditional-One (or DS-Conditional-All) model with the specialization matrix approach (Theoretical computational times are calculated assuming 10,000,000 computational iterations per second).

With increasing FoD size, the computational time requirements of the DS-Conditional-One and DS-Conditional-All models are significantly less compared to what the specialization matrix based approach requires. Fig. 5.2 illustrates the computational and space complexity comparison of DS-Conditional-One (or DS-Conditional-All) model with the specialization matrix approach.



(a) $|\Theta| = 10$ (b) $|\Theta| = 20$ (c) $|\Theta| = 30$

Figure 5.3: Variation of CPU time for arbitrary FH (Dempster's) belief conditional computation versus $|A|$ for different $|B|$ values (when $|\Theta| = 10$, $|\Theta| = 20$, and $|\Theta| = 30$).

Figs 5.3(a), 5.3(b), and 5.3(c) show the variation of CPU time for FH conditional computation versus $|A|$ for different $|B|$ values. Experiments of the FoD size of 10 and 20 were simulated on the desktop computer and FoD size of 30 were completed on the supercomputer. The results were obtained by executing the algorithms for 1000 randomly chosen B propositions and noting the average CPU time. The experiment was repeated for different FoD sizes $|\Theta|$ and the cardinality $|A|$ of the conditioning proposition. A random set of focal elements were generated as the core for each BoE generated.

With increasing size of $|A|$, the average computational time of a randomly chosen belief function decreases. Computational time increases with increasing size of $|B|$. Computational time resolution is 1 μs in the desktop computer and it is 10 ms in the supercomputer. Due to this reduced sensitivity of the supercomputer to smaller computational times, when $|A| > 22$, $|B| = 4$ and $|B| = 8$, an artifact is observed in the plot (Fig. 5.3(c)).

5.4 Concluding Remarks

This chapter provides a scalable, generalized framework for efficient and exact computation of DST conditionals. The DS-Conditional-All model that we propose can also serve as a tool for visualization and advanced analysis of the conditional computation process. We believe that the computational framework that we have developed establishes a significant step forward in harnessing the strengths of DST methods in practical applications.

By carefully reducing the number of operations being executed, the proposed approach takes significantly less computational and space complexity when compared with other approaches for conditional computation.

(a) A smaller matrix is used (only the BoE itself).

(b) Matrix multiplications are avoided (only element additions are involved which are computationally less expensive than multiplications).

(c) Algorithms are developed avoiding repetitive computations.

(d) Access operation of a focal element takes only constant time.

For both Fagin-Halpern and Dempster's conditionals, this framework provides algorithms to compute arbitrary conditional belief (See Chapter 4) in $\mathcal{O}(2^{|\overline{A}|} \times 2^{|A \cap B|})$

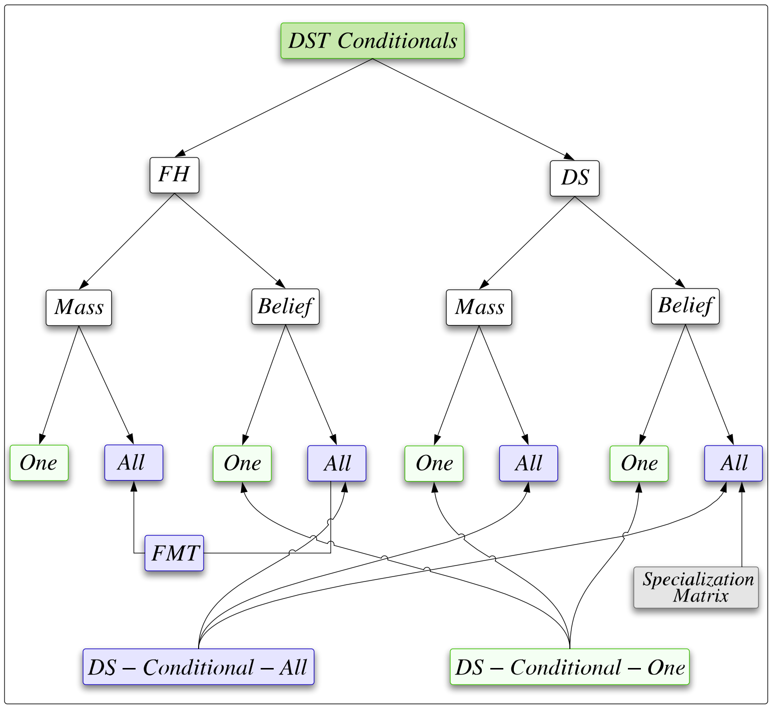


Figure 5.4: Illustration of the best use of DST conditional computation models.

($\leq \mathcal{O}(2^{|\Theta|})$) and all conditional beliefs in $\mathcal{O}(max(2^{|A|} \times |A|, 2^{|\Theta|}))$ ($\leq \mathcal{O}(2^{|\Theta|} \times |\Theta|)$). It also provides algorithms to compute arbitrary Dempster's conditional mass in $\mathcal{O}(max(2^{|\overline{A}|}, |A \cap B|))$ ($\leq \mathcal{O}(2^{|\Theta|})$), and all Dempster's conditional masses in $\mathcal{O}(2^{|\Theta|})$. Therefore the novel contributions in this work provide a seminal advancement in comparison with the well known specialization matrix approach [46] for Demspter's conditional and Conditional Core Theorem approach [59] for Fagin-Halpern conditional, which lead to the computational complexity of $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$. As a clear example, when we consider an FoD of size 30 ($\sim$1 billion focal elements) the proposed framework can be used to compute all conditional masses o beliefs within 1 minute while the specialization matrix will take more than 1800 cpu years. The space complexity of these algorithms is $\mathcal{O}(2^{|\Theta|})$, a promising improvement in contrast to the prohibitive $\mathcal{O}(2^{|\Theta|} \times 2^{|\Theta|})$ space complexity of the specialization matrix approach.

Computational time of an arbitrary conditional belief depends only on the size $|B|$ of the conditioned event and the size $|\overline{A}|$ of the complement of conditioning event. When we increase the cardinality $|A|$ of the conditioning event while keeping the same FoD size $|\Theta|$, the computational time decreases.

Another advantage of the proposed approach is that it can be utilized for either the FH conditional or Dempster's conditional belief computations. An outcome of this research is the *DS-CONAC* library (in C++) which is available at [67]. We expect that this library will be useful for practical application of DST methods. This novel computational framework will be useful for a significant performance improvement in real-time evidence fusion and uncertainty reasoning applications as well as advanced applications using dynamic FoDs, dynamic BoEs and multiple BoEs.

CHAPTER 6

Operations on Dynamic Frames

This chapter provides algorithms to work with dynamic FoDs. In an application that uses a large frame and works for a longer duration, the existing discernible information may vary. A greater possibility is there to dynamically change the presence of existing singletons in the FoD. Removing one singleton from a frame will remove half of the proposition from the BoE. Thus, in comparison with the static frames in practice, keeping the ability of adding, removing and changing singletons to maintain the smallest possible frame is highly important to improve the performance [86].

This chapter is organized as follows: Section 6.1 provides algorithms to add a new singleton to a frame; Section 6.2 includes algorithms to remove a singleton from a frame; Section 6.3 contains the concluding remarks and future directions.

6.1 Adding a Singleton to a Frame

According to the Fig. 3.1, when a new singleton is added, all the new focal elements relevant to it are generated by adding the singleton proposition to all the existing focal elements. This can be implemented using a DS-Vector, DS-Matrix or a DS-Tree.

6.1.1 DS-Vector

Illustration in the Fig. 3.2 provides a visualization of a dynamic BoE. According to the figure, consider adding $\{\theta_2\}$ when $\{\theta_0\}$ and $\{\theta_1\}$ are present in the FoD. It generates $\{\theta_2\}$, $\{\theta_0\theta_2\}$, $\{\theta_1\theta_2\}$ and $\{\theta_0\theta_1\theta_2\}$ focal elements. Algorithm 13 provides an implementation of adding focal elements of a new singleton.

Algorithm 13 Adding focal elements of a singleton in a DS-Vector

```
1: procedure AddFocalEle(Singletons $\Theta$, Singleton $\theta$, Normalize $Nlz$)
2:     add $\theta$ to $\Theta$
3:     resize DS-Vector to $power[|\Theta|]$
4:     for $i \leftarrow power[|\Theta| - 1], power[|\Theta|] - 1$ do
5:         read $potential[i]$
6:         $Nlz \leftarrow Nlz + potential[i]$
7:     end for
8: end procedure
```

The important steps in this algorithm are as follows:

Line #1: Input parameters are FoD; Θ, new singleton θ and normalizing constant Nlz. Normalizing constant is the summation of all the belief potential values.

Line #4-7: Adding the potentials of new focal elements relevant to the new singleton and updating the normalizing constant.

6.1.2 DS-Matrix

Fig. 3.3 provides a representation of a dynamic BoE. According to the figure, consider adding $\{\theta_2\}$ when $\{\theta_0\}$ and $\{\theta_1\}$ are present in the FoD. It generates $\{\theta_2\}$, $\{\theta_0\theta_2\}$, $\{\theta_1\theta_2\}$ and $\{\theta_0\theta_1\theta_2\}$ focal elements.

Algorithm :14 provides an implementation of adding focal elements of a new singleton.

Algorithm 14 Adding focal elements of a singleton in a DS-Matrix

```
1: procedure AddFocalEle(EvenSingletons $\Theta_e$, OddSingletons $\Theta_o$, Singleton $\theta$,
   Normalize $Nlz$)
2:     if $|\Theta_o| \leq |\Theta_e|$ then
3:         add $\theta$ to $\Theta_o$
4:         resize $potential$ to $power[|\Theta_o|]$
5:         for $i \leftarrow power[|\Theta_o| - 1], power[|\Theta_o|] - 1$ do
6:             for $j \leftarrow 0, power[|\Theta_e|] - 1$ do
7:                 read $potential[i][j]$
8:                 $Nlz \leftarrow Nlz + potential[i][j]$
9:             end for
10:        end for
11:    else
12:        add $\theta$ to $\Theta_e$
13:        for $i \leftarrow 0, power[|\Theta_o|] - 1$ do
14:            resize $potential[i]$ to $power[|\Theta_e|]$
15:            for $j \leftarrow power[|\Theta_e| - 1], power[|\Theta_e|] - 1$ do
16:                read $potential[i][j]$
17:                $Nlz \leftarrow Nlz + potential[i][j]$
18:            end for
19:        end for
20:    end if
21: end procedure
```

Line #1: Input parameters are even singletons; Θ_e, odd singletons; Θ_o, new singleton θ and normalizing constant Nlz. Normalizing constant is the summation of all the belief potential values.

Line #2-10: Adding the potentials of new focal elements relevant to new odd singleton and updating the normalizing constant.

Line #11-20: Adding the potentials of new focal elements relevant to new even singleton and updating the normalizing constant.

6.1.3 DS-Tree

Illustration in the Fig. 3.4 provides a balanced binary tree representation of a dynamic BoE. According to the figure, consider adding $\{\theta_2\}$ when $\{\theta_0\}$ and $\{\theta_1\}$ are present in the FoD. It generates $\{\theta_2\}$, $\{\theta_0\theta_2\}$, $\{\theta_1\theta_2\}$ and $\{\theta_0\theta_1\theta_2\}$ focal elements. Algorithm 15 provides an implementation of adding focal elements of a new singleton.

Salient steps of the algorithm are as follows:

Line #1: In this algorithm, input parameters are singletons; Θ, new singleton θ, DSTree T and normalizing constant Nlz. Normalizing constant is the summation of all the belief potential values.

Line #4-7: Creating the new root node and updating the tree.

Line #9-11: Adding the first element of the right sub-tree.

Line #12: Adding the remaining elements of the right sub-tree.

Line #15-26: Procedure for adding the elements of a sub-tree.

Algorithm 15 Adding focal elements of a singleton in a DS-Tree

```
1: procedure AddFocalEle(Singletons $\Theta$, Singleton $\theta$, DSTree $T$, Normalize $Nlz$)
2:     add $\theta$ to $\Theta$
3:     level $\leftarrow |\Theta| - 1$
4:     create $node$
5:     $Nlz \leftarrow Nlz + node.mass$
6:     $node.left \leftarrow T.root$
7:     $T.root \leftarrow node$
8:     if $level > 0$ then
9:         create $node$
10:        $Nlz \leftarrow Nlz + node.mass$
11:        $T.root.right \leftarrow node$
12:        AddSubTree($root.right$, $level - 1$)
13:    end if
14: end procedure
15: procedure AddSubTree(Node $leaf$, Level $level$)
16:    if $level > 0$ then
17:        create $node$
18:        $Nlz \leftarrow Nlz + node.mass$
19:        $leaf.left \leftarrow node$
20:        AddSubTree($leaf.left, level - 1$)
21:        create $node$
22:        $Nlz \leftarrow Nlz + node.mass$
23:        $leaf.right \leftarrow node$
24:        AddSubTree($leaf.right, level - 1$)
25:    end if
26: end procedure
```

6.2 Removing a Singleton from a Frame

Removing one singleton on an FoD will remove half of the elements of the BoE. So keeping the minimum possible elements in the BoE significantly improves the performance. The following discussion is on removing the focal elements relevant to a singleton on a DS-Vector, DS-Matrix or a DS-Tree.

6.2.1 DS-Vector

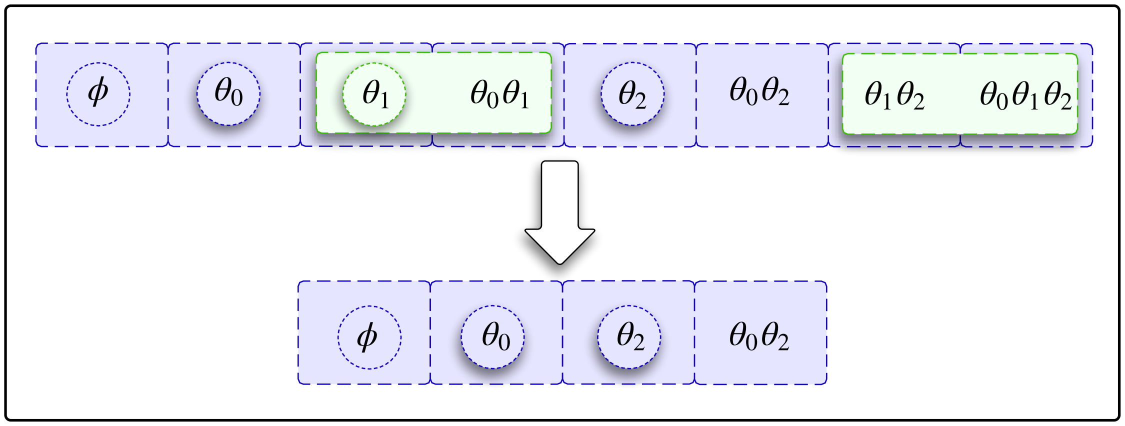


Figure 6.1: Removing focal elements of a singleton in a DS-Vector

Illustration in the Fig. 6.1 provides a visualization of removing focal elements relevant to the singleton $\{\theta_1\}$. According to the figure, it removes $\{\theta_1\}$ and the next element $\{\theta_0\theta_1\}$, and also last two elements $\{\theta_1\theta_2\}$ and $\{\theta_0\theta_1\theta_2\}$. Algorithm :16 provides an implementation of removing focal elements relevant to a singleton.

The important steps in this algorithm are as follows:

Line #1: Input parameters are FoD; Θ, removing singleton θ_n and normalizing constant Nlz. Normalizing constant is the summation of all the belief potential values.

Algorithm 16 Removing focal elements of a singleton in a DS-Vector

```
1: procedure RemoveFocalEle(Singletons $\Theta$, Singleton $\theta_n$, Normalize $Nlz$)
2:     $start \leftarrow 1$
3:     while $power[n] * 2 * start \leq power[|\Theta|]$ do
4:         for $i \leftarrow power[n] * (2 * start - 1), power[n] * 2 * start - 1$ do
5:             $Nlz \leftarrow Nlz - focalelement[i]$
6:         end for
7:         if $start > 1$ then
8:             $copy(focalelement.begin() + power[n] * 2 * (start - 1), focalelement.begin() + power[n] * 2 * (start - 1) + power[n], focalelement.begin() + power[n] * (start - 1))$
9:         end if
10:        $start \leftarrow start + 1$
11:    end while
12:    remove $\theta_n$ from $\Theta$
13:    resize $focalelement$ to $power[|\Theta|]$
14: end procedure
```

Line #3: Iterating through removing focal element segments.

Line #4-6: Removing potentials relevant to the removing singleton and updating the normalizing constant.

Line #7-9: Replace the memory segment of the removed potentials by the next available focal element segment.

Line #12-13: remove θ_n from FoD; Θ and resizing the DS-Vector.

6.2.2 DS-Matrix

Fig. 6.2 provides a representation of removing focal elements relevant to the singleton θ_0. It removes the 1st and 3rd columns of focal elements and leaves focal elements relevant to θ_1, θ_2 and θ_3. Algorithm 17 provides an implementation of removing focal elements relevant to a singleton.

The important steps in this algorithm are as follows:

Algorithm 17 Removing focal elements of a singleton in a DS-Matrix

```
1: procedure RemoveFocalEle(EvenSingletons $\Theta_e$, OddSingletons $\Theta_o$, Singleton $\theta_n$, Normalize $Nlz$)
2:     if $\theta_n$ in $\Theta_o$ then
3:         while $power[n] * 2 * start \leq power[|\Theta_o|]$ do
4:             for $i \leftarrow 0, power[|\Theta_e|] - 1$ do
5:                 for $j \leftarrow power[n] * (2 * start - 1), power[n] * 2 * start - 1$ do
6:                     $Nlz \leftarrow Nlz - focalelement[i][j]$
7:                 end for
8:                 if $start > 1$ then
9:                     $copy(focalelement[i].begin() + power[n] * 2 * (start - 1), focalelement[i].begin() + power[n] * 2 * (start - 1) + power[n], focalelement[i].begin() + power[n] * (start - 1))$
10:                end if
11:            end for
12:            $start \leftarrow start + 1$
13:        end while
14:        remove $\theta_n$ from $\Theta_o$
15:        for $i \leftarrow 0, power[|\Theta_e|] - 1$ do
16:            resize $focalelement[i]$ to $power[|\Theta_o|]$
17:        end for
18:    else if $\theta_n$ in $\Theta_e$ then
19:        while $power[n] * 2 * start \leq power[|\Theta_e|]$ do
20:            for $j \leftarrow 0, power[|\Theta_e|] - 1$ do
21:                for $i \leftarrow power[n] * (2 * start - 1), power[n] * 2 * start - 1$ do
22:                    $Nlz \leftarrow Nlz - focalelement[i][j]$
23:                end for
24:                if $start > 1$ then
25:                    $count \leftarrow 0$
26:                    for $i \leftarrow power[n] * 2 * (start - 1), power[n] * 2 * (start - 1) + power[n] - 1$ do
27:                        $focalelement[count + power[n] * (start - 1)][j] \leftarrow focalelement[i][j]$
28:                        $count \leftarrow count + 1$
29:                    end for
30:                end if
31:            end for
32:            $start \leftarrow start + 1$
33:        end while
34:        remove $\theta_n$ from $\Theta_e$
35:        resize $focalelement$ to $power[|\Theta_e|]$
36:    end if
37: end procedure
```

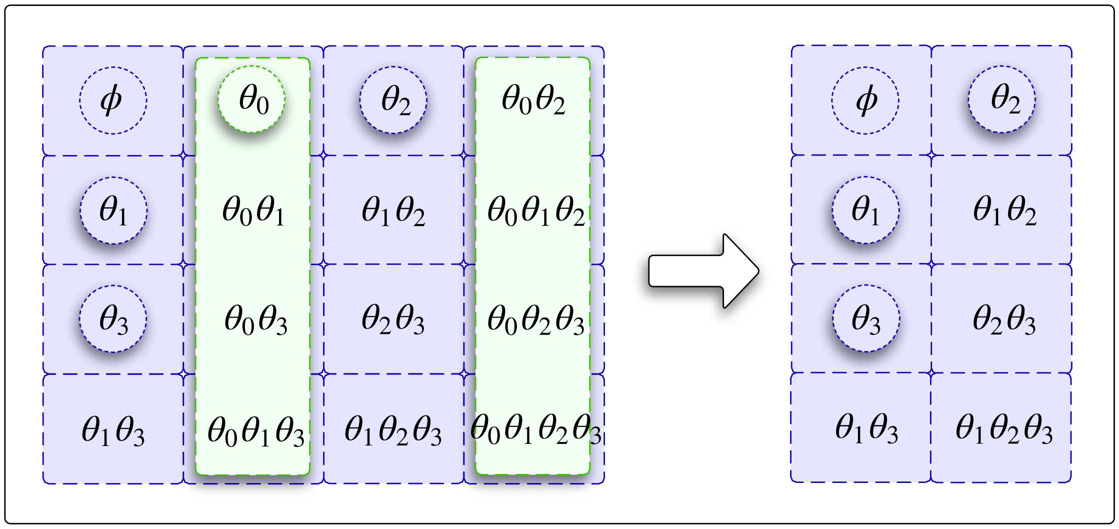


Figure 6.2: Removing focal elements of a singleton in a DS-Matrix

Line #1: Input parameters are even singletons; Θ_e, odd singletons; Θ_o, removing singleton θ_n and normalizing constant Nlz. Normalizing constant is the summation of all the belief potential values.

Line #2-17: If θ_n is an odd singleton, executing row operations.

Line #18-36: If θ_n is an even singleton, executing column operations.

Line #3: Iterating through removing focal element segments.

Line #14-16: remove θ_n from Θ_o and removing unused rows.

Line #34-35: remove θ_n from Θ_e and removing unused columns.

6.2.3 DS-Tree

Fig. 6.3 provides a representation of removing focal elements relevant to the singleton θ_2. It removes sub branches represent by green dotted lines and leaves focal elements relevant to θ_0, θ_1 and θ_3. Algorithm :18 provides an implementation of removing focal elements relevant to a singleton.

Algorithm 18 Removing focal elements of a singleton in DS-Tree

```
1: procedure DeleteSubTree(Node $leaf$)
2:     $Nz \leftarrow 0$
3:     if leaf.left then
4:         $Nz \leftarrow Nz+$ DeleteSubTree($leaf.left$)
5:     end if
6:     if leaf.right then
7:         $Nz \leftarrow Nz+$ DeleteSubTree($leaf.left$)
8:     end if
9:     if leaf then
10:        $Nz \leftarrow Nz + leaf.mass$
11:        delete $leaf$
12:    end if
13:    Return $Nz$
14: end procedure
15: procedure RemoveSubTrees(DSTree $T$, Node $leaf$, Step $step$, Node $parent$,
    bool $atroot$, bool $leftchild$)
16:    if $step = 0$ then
17:        if $atroot$ then
18:            $T.root \leftarrow leaf.left$
19:        else
20:            if $leftchild$ then
21:                $parent.left \leftarrow leaf.left$
22:            else
23:                $parent.right \leftarrow leaf.left$
24:            end if
25:        end if
26:        $T.Nlz \leftarrow T.Nlz-$ DeleteSubTree($leaf.right$)
27:        $T.Nlz \leftarrow T.Nlz - leaf.mass$
28:        delete $leaf$
29:    else
30:        RemoveSubTrees($T$, $leaf.left$, $step - 1$, $leaf$, $false$, $true$)
31:        RemoveSubTrees($T$, $leaf.right$, $step - 1$, $leaf$, $false$, $false$)
32:    end if
33: end procedure
34: procedure RemoveFocalEle(Singletons $\Theta$, Singleton $\theta_n$, DSTree $T$)
35:    RemoveSubTrees($T$, $T.root$, $|\Theta| - n$, $null$, $true$, $true$)
36:    remove $\theta_n$ from $\Theta$
37: end procedure
```

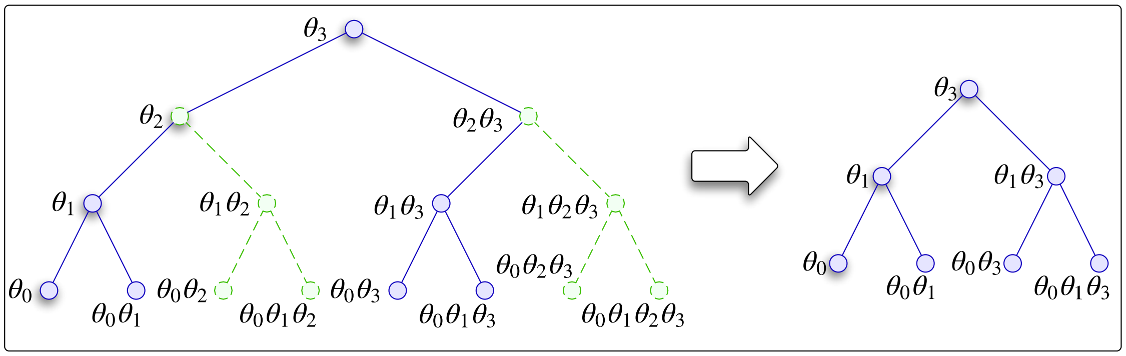


Figure 6.3: Removing focal elements of a singleton in a DS-Tree

Salient steps of the algorithm are as follows:

Line #1: Procedure to traverse a removing sub tree.

Ling #15: Procedure to make recursive call to removing sub trees.

Line #34: In this algorithm, input parameters are singletons; Θ, removing singleton θ_n, DSTree T.

6.3 Concluding Remarks

This chapter provides a general framework along with data structures and algorithms to work with dynamic FoDs. The data structures DS-Vector, DS-Matrix and DS-Tree can also be utilized as tools to visualize dynamic operations. Removing one singleton from a frame removes half of the propositions that need to be considered. Thus, from the computational perspective, incorporating the ability to add and remove singletons from a frame is highly important for enhanced resource utilization. We are conducting a wide range of experiments with the topics discussed in Chapter 7. The visualization tools that we are developing in Section 7.4 will be useful to further optimize the dynamic operations.

CHAPTER 7

Future Work

This chapter provides a discussion of our future work.

This chapter is organized as follows: Section 7.1 efficient computation of DST fusion strategies; Section 7.2 computations on low density BoEs; Section 7.3 baseline network selection in InSAR; Section 7.4 effective visualizations; Section 7.5 computational libraries.

7.1 DST Fusion Strategies

7.1.1 Dempster's Rule of Combination

The combination operation plays an important role in DST applications for evidence fusion [87]. Among these various evidence combination notions that have been proposed over the years [11, 18], perhaps the most widely used DST combination is the Dempster's rule of Combination [12].

7.1.1.1 Arbitrary Computations

We can develop an efficient algorithm to compute arbitrary combination computations. Fig. 7.1 visualizes the Dempster's rule of combination propositions. Fig. 7.2 represents proposition of a conjunctive combination operation.

7.1.1.2 Visualizing Conflicts, Addressing Limits

Fig. 7.3 visualizes a conflict [88] of a Dempster's Rule of Combination Operation. We can create a new combination rule to address these limitations.

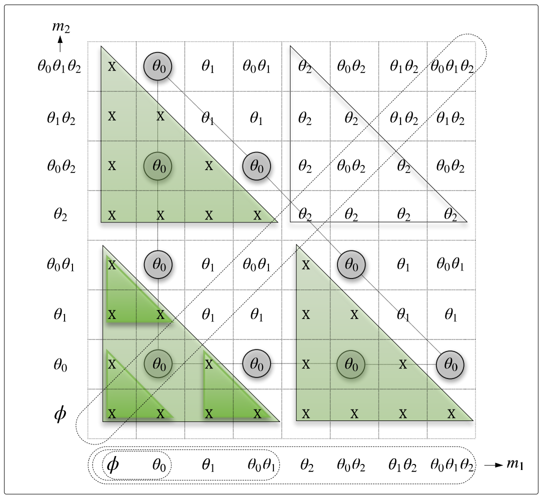


Figure 7.1: Visualizing the Dempster's rule of combination for 2 BoEs.

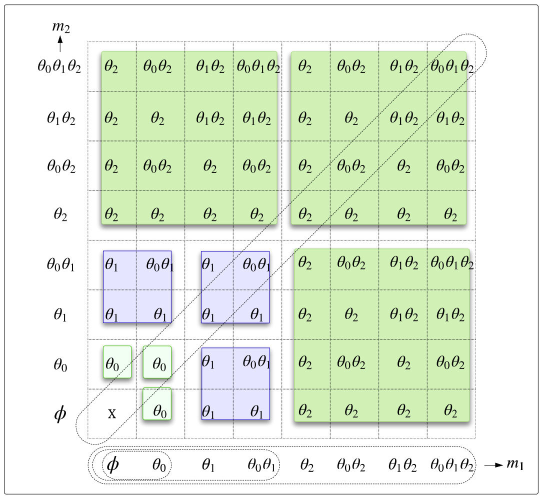


Figure 7.2: Visualizing a conjunctive combination for 2 BoEs.

7.1.1.3 Generalized Algorithm for Multiple BoEs

Fig. 7.4 provides a 3D visualization of Dempster's rule of combination operation. We can develop generalized arbitrary computational algorithms to work with any number of BoEs by analyzing its properties.

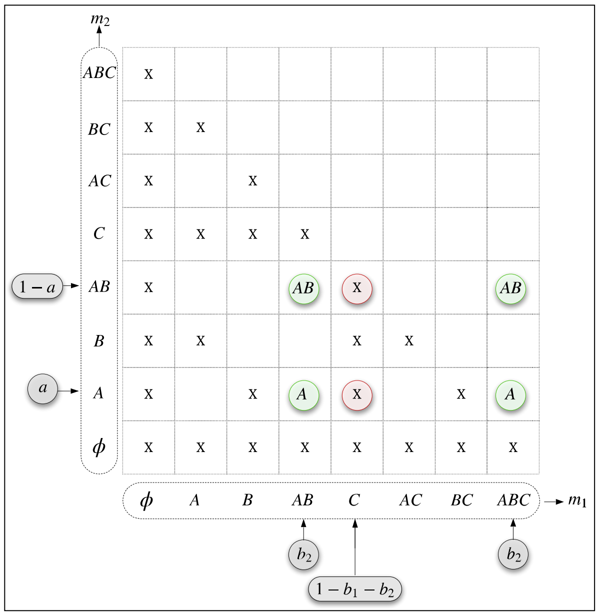


Figure 7.3: Visualizing a conflict of a Dempster's rule of combination Operation.

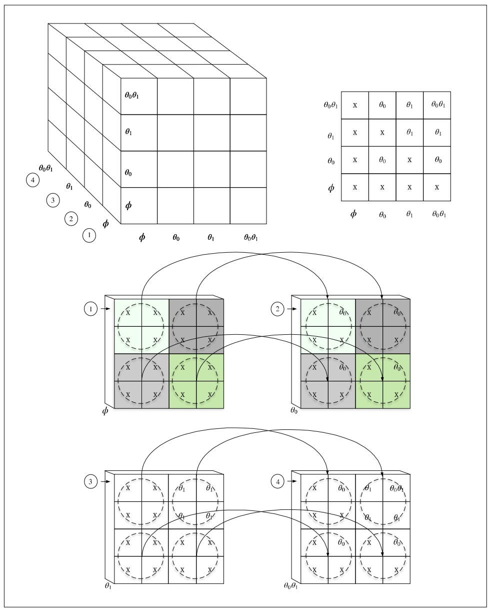


Figure 7.4: 3D Visualization of a Dempster's rule of combination for 3 BoEs.

7.1.2 Conditional Update Equation (CUE), Conditional Fusion Equation (CFE) and Pignistic Transformation

Our current work involves developing efficient algorithms for the following DST fusion strategies:

- Conditional Update Equation (CUE) [47].
- Conditional Fusion Equation (CFE) [68].
- Pignistic transformation [16, 69, 70].

We are also conducting a wide range of experiments with dynamic operations to improve and conclude the work discussed in Chapter 6.

7.2 Efficient Computations on Low Density BoEs

The current work that is being carried out includes developing efficient algorithms and data structures to work with general low density BoEs.

We are developing efficient algorithms for the following categories:

- Dirichlet belief functions [71, 72].
- Consonant belief functions [14].
- Single focal element (SFE) BoEs.
- Other low density BoEs.

7.3 Baseline Network Selection in Interferometric Synthetic Aperture Radar (InSAR)

Synthetic aperture radar (SAR) interferometry (InSAR) [73] is an important technique to estimate the land surface deformation caused by natural and anthropogenic processes including earthquakes, volcanic eruptions, landslides, and hydrological subsidence [74–79]. SAR images are generated by processing (satellite) radar data. Each image produces amplitude and phase for a given area. InSAR exploits the difference of pairs of SAR images in order to extract signals of subsidence or uplifting in an area of interest.

7.3.1 InSAR Processing

Interferometry uses two SAR images of the same area taken from the same position and finds the difference in phase between them, producing an image known as an interferogram [73]. The satellites must be as close as possible to the same spatial position when the SAR images are collected. This means that images from two satellite platforms with different orbits cannot be compared, and for a given satellite, data from the same orbital track must be used. The perpendicular distance between them is known as the baseline. We can represent SAR images on a network by having time on the x-axis and perpendicular baseline on the y-axis [89]. The time axis accounts for the time that has elapsed since the first SAR was acquired while the y axis accounts for the difference in perpendicular baseline relative to the first SAR collection in the sequence [90]. This network is referred to as a baseline network. Selected interferograms can be represented as edges in a baseline network.

7.3.1.1 Challenge

Selecting the best baseline network is still an open challenge. Current network selection criterion removes the interferograms based on a predefined minimum baseline perpendicular distance and predefined time duration. However quality of the final output changes even after that. Then it is required to get a fresh start to the processing again by changing the baseline network. Selecting the best baseline network before the start of InSAR processing is greatly beneficial to optimize the InSAR processing work.

7.3.1.2 Approach

First, we completed a study on baseline network selection using network centrality concepts [80]. Then, we developed a deep learning model to select good networks based on expert opinions. The remaining work is, developing baseline network selection criterion to work with imperfect data by incorporating a wide range of uncertainties.

7.3.2 Improvements Using Network Centrality Concepts

We have analyzed and compared the optimization of InSAR baseline network selection process by applying different centrality methods [80]. We have conducted the study with degree centrality, eigenvector centrality, Katz centrality, page-rank centrality, authority+hub centrality, closeness centrality, betweenness centrality and flow-betweenness centrality. Temporal coherence [91] is the measurement we have used to measure the quality, however thoughtful discussion is still needed to select the appropriate measurement. The coherence value ranges from 0 (the interferometric phase is just noise) to 1 (complete absence of phase noise). The main focus was to design and optimize the network by keeping the highest possible temporal coherence. We conducted the experiment with several network formations. Flow-betweenness centrality gave significantly improved results.

7.3.2.1 Data Set

Heroica Nogales, more commonly known as Nogales, is a city and the county seat of the Municipality of Nogales. It is located on the northern border of the Mexican state of Sonora. This is a well-known area for a lot of tunnel constructions and activities. The SAR acquisition covers this geographical area. A data set of 50 SAR images from December 16, 2009, to January 25, 2012 was used to generate relevant outcomes of this study. The Initial network contained 225 interferograms.

7.3.2.2 Improvements from Flow-betweenness Centrality

Flow-betweenness [80] is measuring the betweenness of vertices in a network in which a maximal amount of information is continuously pumped between all sources and targets. Flow-betweenness takes account of more than just the geodesic paths between vertices, since flow can go along non-geodesic paths as well as geodesic ones.

Suppose $\tilde{n}_{kl}^{(i)}$ denotes the amount of flow thorough vertex $\#i$ when a maximum flow is transmitted from vertex $\#l$ to $\#k$. Then flow betweenness centrality of vertex $\#i$ is

$$X_i = \sum_{k,l} \frac{\tilde{n}_{kl}^{(i)}}{g_{kl}}, \tag{7.1}$$

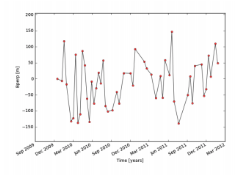
where g_{kl} denotes the total # of geodesic paths between vertex $\#k$ and vertex $\#l$. Here, we use the convention $\frac{\tilde{n}_{kl}^{(i)}}{g_{kl}} = 0$, whenever both $\tilde{n}_{kl}^{(i)}$ and g_{lk} are zero.

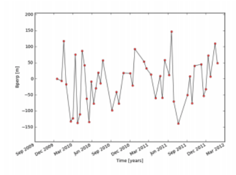
Flow-betweenness centrality measures were calculated in each and every nodes of the data set. The least important nodes according to flow-betweenness centrality measures were obtained by arranging the relevant values of all the nodes in ascending order. Table: 7.1 contains the relevant nodes with respective centrality measures.

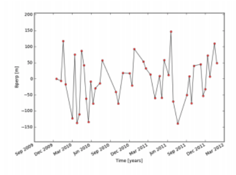
Fig. 7.5-(a), Fig. 7.5-(e), and Fig. 7.5-(i) are the initial network of baseline history, network of interferograms, and the map of temporal coherence of the data set, respectively. Fig. 7.5-(b), Fig. 7.5-(f), and Fig. 7.5-(j) are obtained by removing the least important 2 nodes according to the flow-betweenness centrality. Likewise, Fig. 7.5-

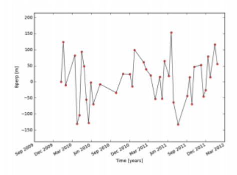
Bottom rank	Node	Flow-betweenness centrality value
1	100804	68
2	100815	91
3	100702	126
4	100220	127
5	100906	127
6	100621	130
7	91216	132
8	100303	134
9	100724	139
10	101009	139

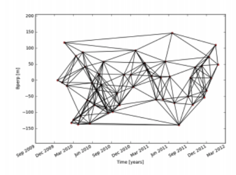
Table 7.1: The least important nodes according to flow-betweenness centrality.

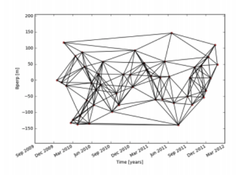

(a) n = 0

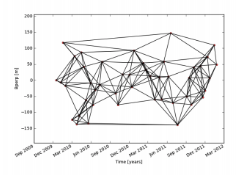

(b) n = 2

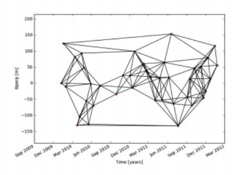

(c) n = 5

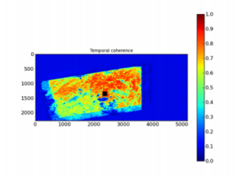

(d) n = 10

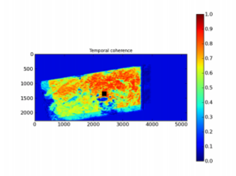

(e) n = 0

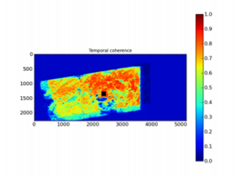

(f) n = 2


(g) n = 5


(h) n = 10


(i) n = 0


(j) n = 2


(k) n = 5

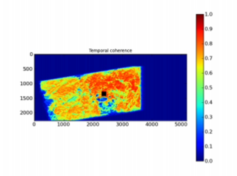

(l) n = 10

Figure 7.5: Comparison of flow-betweenness centrality corrections. (a)-(d) Network of baseline history. (e)-(h) Network of interferograms. (i)-(l) Map of temporal coherence.

(c), Fig. 7.5-(g), Fig. 7.5-(k) are plotted by removing the least important 5 nodes and Fig. 7.5-(d), Fig. 7.5-(h), Fig. 7.5-(l) are generated by removing the least important 10 nodes.

Comparing the temporal coherence maps in Fig. 7.5, it is clear that removing the least important nodes using flow-betweenness not only optimizes, but also increases the temporal coherence and quality of the results. Analyzing the temporal coherence maps of Fig. 7.6 it is evident that removing 5 nodes using betweenness centrality measure provides the initial temporal coherence. However, Temporal coherence was reduced when removing 2, and 10 nodes.

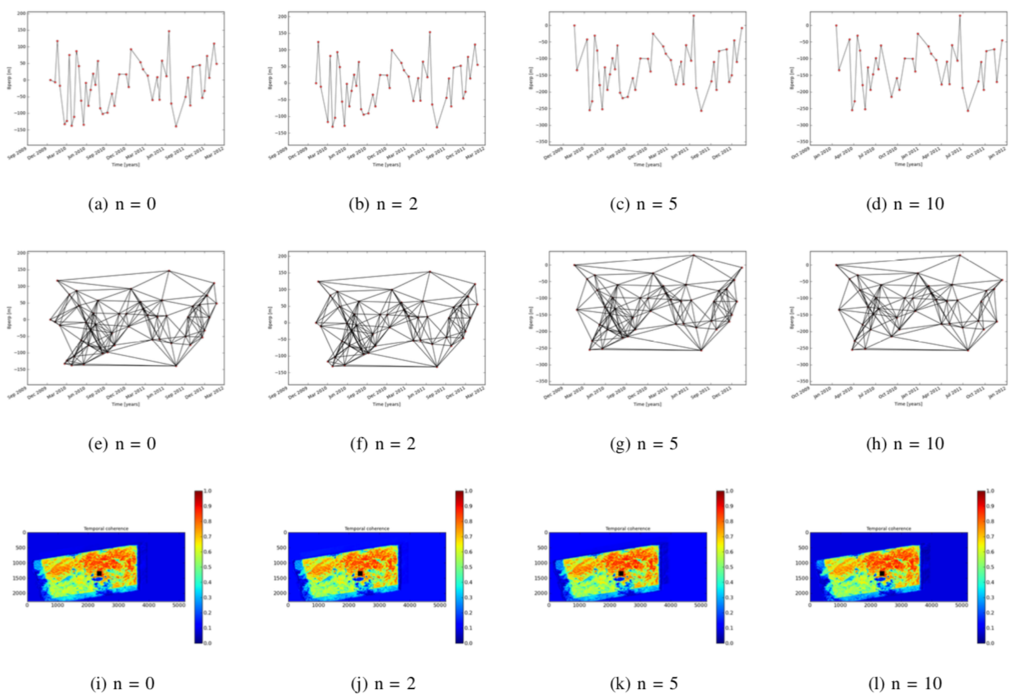


Figure 7.6: Comparison of betweenness centrality corrections. (a)-(d) Network of baseline history. (e)-(h) Network of interferograms. (i)-(l) Map of temporal coherence.

In this experiment; degree, eigenvector, Katz, page-rank, authority+hub, and closeness centralities did not contribute to any critical result.

The main objective of this study was to find better network selection criteria to improve the process efficiency. According to the results, the main objective is almost achieved. However, selection of the most appropriate network quality measurement or collection of measurements is a topic that is still under discussion.

7.3.2.3 A Deep Learning Model to Measure Network Quality Based on Expert Opinions

Deep learning is a process of training a neural network to achieve a specific task. We have developed an artificial intelligence (AI) prototype using deep learning [81] to measure the quality of networks. It gave an impressive 99.3% accuracy. We have used TensorFlow [92], pandas [93], SciPy [94] and Scikit-learn [95] libraries, in the development. The prototype was developed using Python and the synthetic data generator was developed in C++.

Our objective was to create an AI model to measure the quality of networks. Lets consider the below example:

We have selected 10 nodes (lets consider them as SAR images), then divided the nodes into two groups: 5 green, and 5 blue as shown in Fig. 7.7. We have created a deep neural network with three hidden layers: 1000 nodes in the first hidden layer; 2000 nodes in the second hidden layer; and 1000 nodes in the third hidden layer. The learning rate used was 0.001 and 1000 training epochs were conducted during the training phase. We created 1200 synthetic networks for the selected 10 nodes (see Fig. 7.7). We can assign +1 to the edges between the same color nodes and -1 otherwise (lets consider them as the quality of interferograms). Then we can add the edge values to get a final value for the network. Fig. 7.7 contains 6 networks with

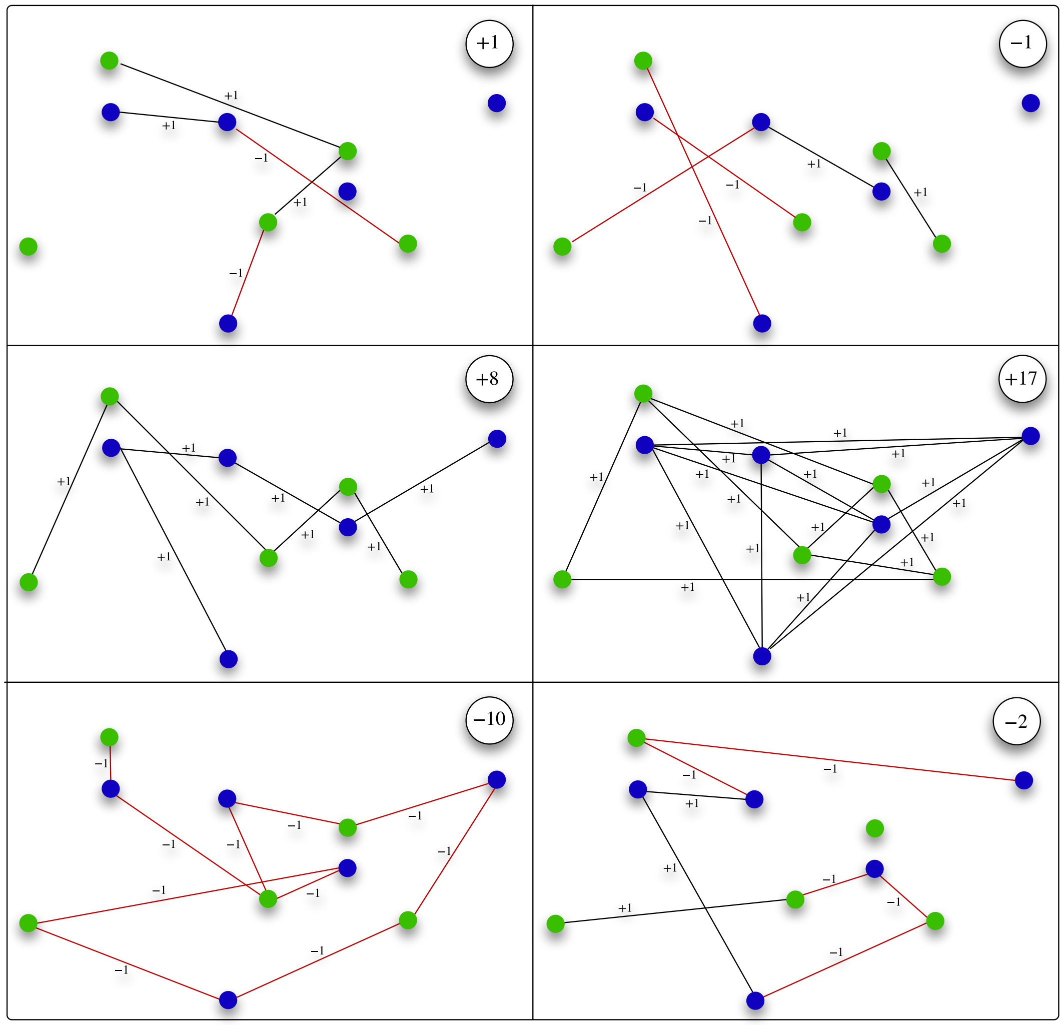


Figure 7.7: Six Networks out of 1200 used to train and test the AI prototype of network quality measurement. Each network contains 5 blue and 5 green nodes. The value of the edges between the same color nodes is +1 and otherwise -1.

values assigned. Node colors as well as the individual edge values should be hidden to the prototype. Adjacency matrix of generated networks is the input and the quality measurement is the output. We have used 1000 networks as training data and 200 networks as testing data. The trained AI prototype gave 99.3% accuracy.

Expert opinion about the network can be used to train and test the prototype. In InSAR processing, experts can get an idea about the quality of interferograms by carefully observing the processed data. With a large collection of baseline networks with accurate quality values assigned, it is possible to create a deep neural network model to measure the quality of new baseline networks. To accommodate real data it is required to model with uncertainties.

7.4 Effective Visualizations

A review of current DST contributions [11,18,25] reveal that more work is needed in the fields of combinatorics, discrete and computational geometry. Evaluating large flows of data in an appealing way just by using a fundamental numerical format is almost impossible. It is challenging for scientists to quantitatively examine the non-deterministic polynomial-time hardness (NP-hard) problems associated with large FoDs. Investigating the qualitative nature of DST computations become even more challenging. Visualization can assist the scientist in several ways. Changes in parameters can be readily grasped, patterns and classifications become more visible and anomalies in the data can be easily detected. Therefore, the ability to visualize complex DST computations and simulations is absolutely essential to reinforce cognition, decision making, and reasoning [60,61].

The DS-LASIC, DS-TRISEV proposed in this chapter are valuable tools for enhanced visual representations. Current work involves creating an effective visualization framework to assist scientists working on reasoning under uncertainty. This would be of great value for interactive teaching and learning sessions.

7.4.1 Representation of Dynamic BoEs using Venn Diagrams

Venn diagrams were conceived by John Venn [82], but the roots of these type of diagrams go back further in history. [96] provides an excellent survey of work in Venn diagrams including the open problems.

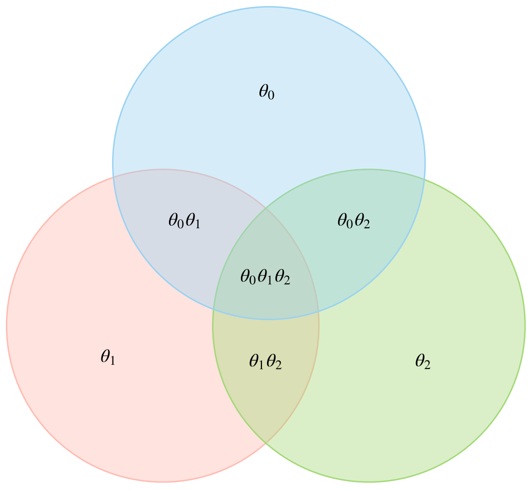


Figure 7.8: Representation of a dynamic BoE as a 3-set symmetric Venn diagram when $\Theta = \{\theta_0, \theta_1, \theta_2\}$, and $|\Theta| = 3$.

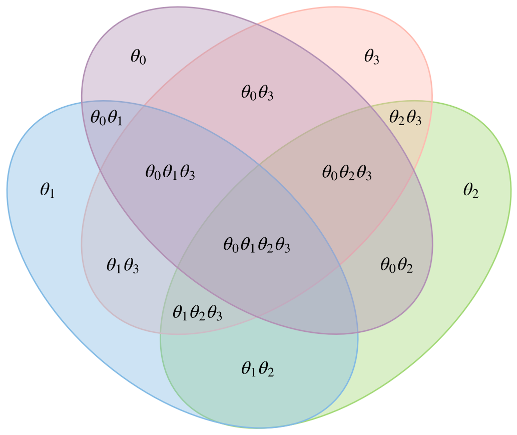


Figure 7.9: Representation of a dynamic BoE as a 4-set Venn diagram using ellipses when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3\}$, and $|\Theta| = 4$.

The classic three circle Venn diagram [82] is monotone, simple, and has polar symmetry. A dynamic BoE of an FoD of size 3 can be represented as in Fig. 7.8. In geometry, a polygon P in a plane is called *monotone* with respect to a straight line L, if every line orthogonal to L intersects P at most twice. A *simple* Venn diagram is one in which no more than two curves intersect at a common point. They are easiest to draw and understand. The Fig. 7.9 represents a dynamic BoE of an FoD of size 4 using a Venn diagram of 4 ellipses, originally found by Venn [82].

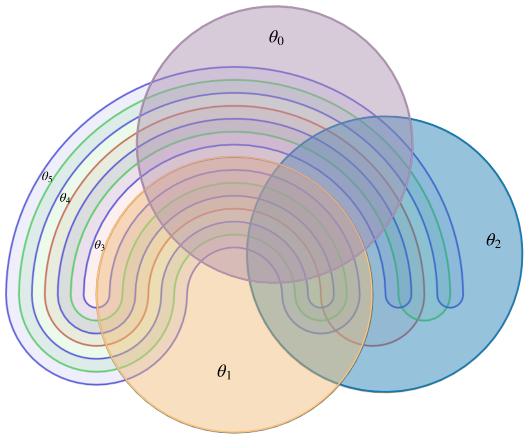


Figure 7.10: Representation of a dynamic BoE as a 6-set Venn diagram when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$, and $|\Theta| = 6$.

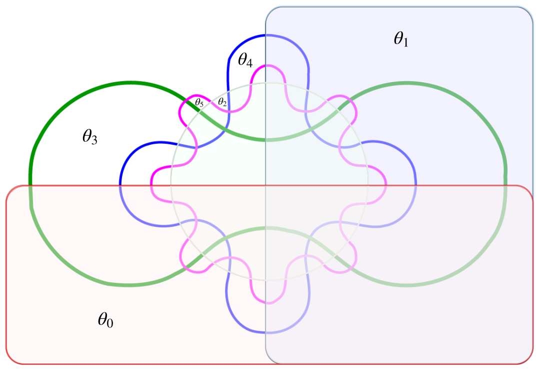


Figure 7.11: Representation of a dynamic BoE as a 6-set Edwards-Venn diagram when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$, and $|\Theta| = 6$.

Venn's construction, outlined in [82] explained the existence of Venn diagram for all the sizes. In the Fig. 7.10 red, green and blue shapes added after initial three circle. It should be clear where the next curve would be drawn. Edwards developed another general inductive construction that has several characteristics including some symmetry [97]. It starts with two perpendicular lines, next a circle is added. Successive curves will be weaving back and forth along the circle. In addition to the above discussion several constructions were published [98, 99]. Figs. 7.10 and 7.11 can be used to represent a dynamic BoE of an FoD of size 6. Using Venn's and Edwards constructions it is possible to represent dynamic BoEs for other FoD sizes too. However since they lack polar symmetry, visualization becomes complex for representation of DST computational operations.

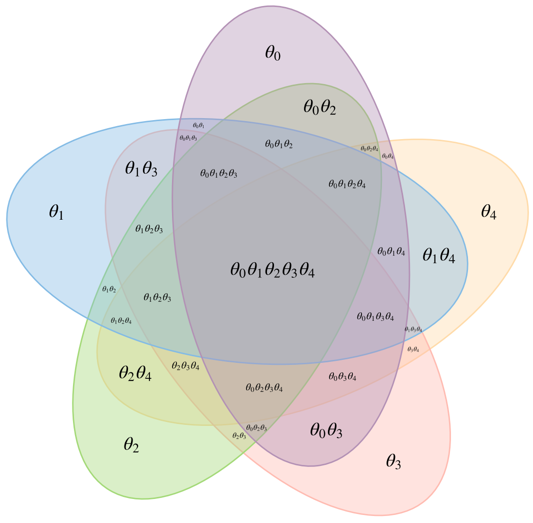


Figure 7.12: Representation of a dynamic BoE as a 5-set Venn diagram using congruent ellipses in a 5-fold rotationally symmetrical arrangement, when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4\}$, and $|\Theta| = 5$.

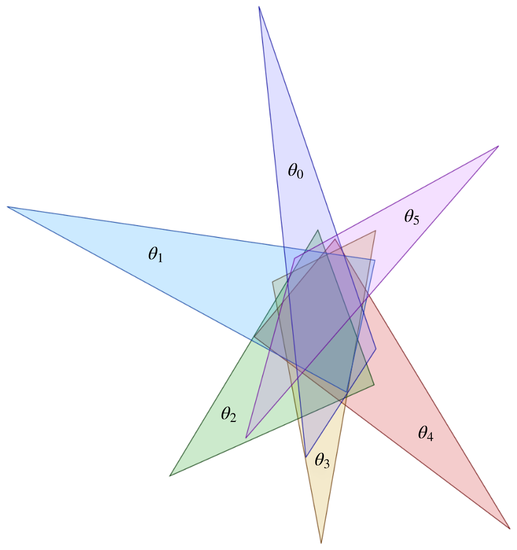


Figure 7.13: Representation of a dynamic BoE as a 6-set Venn diagram using triangles when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$, and $|\Theta| = 6$.

There is a simple symmetric 5-set Venn diagram in which each curve is a triangle [100]. There was an open question about a 6-set Venn diagram made of triangles [101]. This question was solved by Carroll [102], which can be used to represent a dynamic BoE of an FoD of size 6. See Fig. 7.13.

From a computational perspective, the ability to represent dynamic BoEs with a symmetrical or recursive approach is highly valuable to analyze computational operations on larger FoDs. Fig. 7.12 represents a dynamic BoE as a 5-set Venn diagram using congruent ellipses in a 5-fold rotationally symmetrical arrangement. This simple ellipse diagram is the only simple symmetric Venn diagram for 5-set representation, which was constructed by Grünbaum [103]. Referring to the 7-set representations, Grünbaum's 7-set non-monotone simple symmetric Venn diagram is a noticeable work [100]. It can be used to represent a dynamic BoE of an FoD of size 7 as in Fig. 7.14.

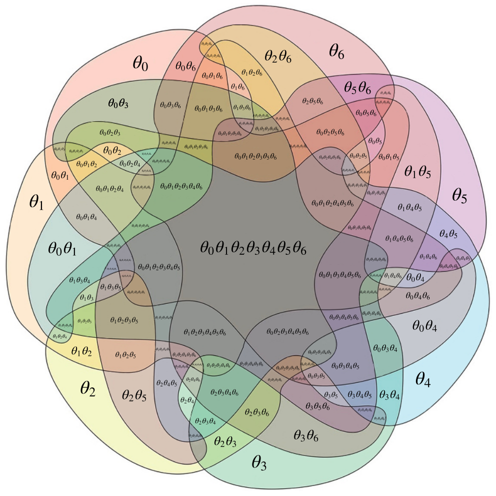


Figure 7.14: Representation of a dynamic BoE as a 7-set non-monotone simple symmetric Venn diagram when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$, and $|\Theta| = 7$.

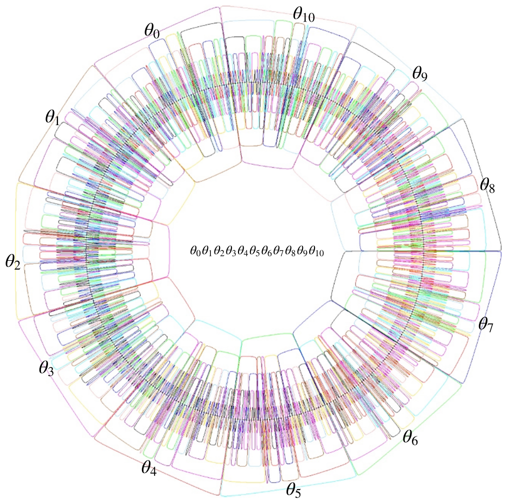


Figure 7.15: Representation of a dynamic BoE as a 11-set symmetric Venn diagram when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8, \theta_9, \theta_{10}\}$, and $|\Theta| = 11$.

The first symmetric Venn diagram for a 11-set representation was constructed by Hamburger [104]. It can be used to represent a dynamic BoE of an FoD of size of 11. See Fig. 7.15. The diagram is highly non-simple and there is still no simple symmetric 11-Venn diagram discovered. There are a lot of open problems to solve [96]. However it will be a highly challenging task to represent large FoDs using Venn diagrams. There are no simple symmetric Venn diagrams for set sizes above 7. To provoke deeper insights on DST computations and to make computational optimizations, a flexible visualization framework is greatly valuable.

7.4.2 Effective DST Visualizations

A countable set is a set with the same cardinality as some subset of the set of natural numbers. A countable set is either a finite set or a countably infinite set. The elements of a countable set are enumerable, which is the key to achieve computability.

As discussed in Section 3.1 and 3.2 DS-Vector (Fig. 3.2), DS-Matrix (Fig. 3.3), DS-Tree (Fig. 3.4) can be effectively utilized to analyze DST Computations.

Fig. 3.5 illustrates a belief computation. It would be clear from the figure that the computational time of belief computations depends only on the cardinality of the belief proposition, and not on the cardinality of the complete BoE. The overlapped region of the Fig. 3.6 illustrates propositions relevant to a plausibility computation.

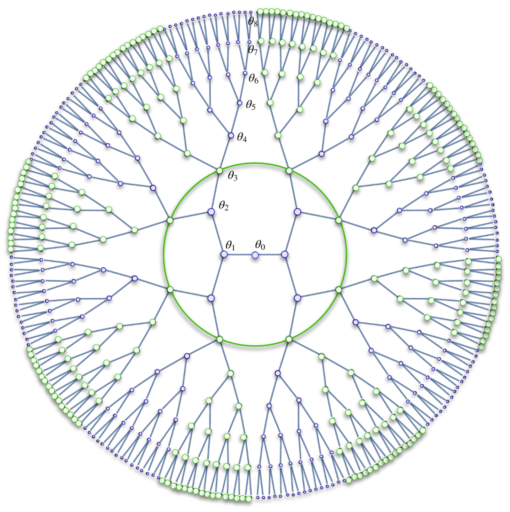


Figure 7.16: **DS-LASIC Diagram:** Representation of a dynamic BoE (a DS-Tree) as a layered symmetric clustering diagram when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8\}$, and $|\Theta| = 9$, here proposition $\{\theta_0\}$ is the root node.

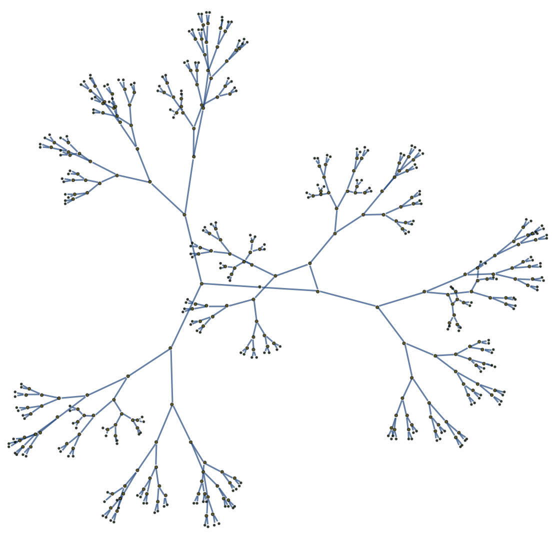


Figure 7.17: **DS-TRISEV Model:** 3D representation of a dynamic BoE (a DS-Tree) using spring electrical model when $\Theta = \{\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8\}$, and $|\Theta| = 9$.

7.4.2.1 DS-LASIC (DS-Layered Symmetric Clustering) Diagram

A dynamic BoE can be visualized as a layered symmetric clustering diagram as in Fig. 7.16. We refer to this diagram as a DS-LASIC (DS-Layered Symmetric Clustering) diagram. This was developed by incorporating DS-Tree properties. Root node is $\{\theta_0\}$ and the rest of the nodes exactly follow all the DS-Tree properties. We can use a DS-LASIC diagram to effectively visualize DST dynamic operations and computations. Fig. 7.16 is a DS-LASIC diagram of an FoD of size 9.

7.4.2.2 DS-TRISEV (DS-Three Dimensional Spring Electrical Visualization) Model

Fig. 7.17 provides a Three-Dimensional (3D) visualization of a dynamic BoE using spring electrical model [105–107]. We refer to this model as a DS-TRISEV (DS-Three Dimensional Spring Electrical Visualization) model. The spring-electrical algorithm has two forces. The repulsive force, which exists between any two vertices, is inversely proportional to the distance between them. The attractive force, exists only between neighboring vertices and is proportional to the square of the distance. Fig. 7.17 is a DS-TRISEV model of an FoD of size 9. The root node is $\{\theta_0\}$ and the rest of the nodes exactly follow the DS-Tree properties.

7.5 Computational Libraries

We have created three open source libraries, DS-BCL [62], DS-CCL [65] and DS-CONAC [67]. These libraries are being improved and computational libraries for other findings are being developed.

Bibliography

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[3] Y. Leviathan and Y. Matias, "Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone," May 2018. [Online]. Available: https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html

[4] Y. Liu, K. Gadepalli, M. Norouzi, G. E. Dahl, T. Kohlberger, A. Boyko, S. Venugopalan, A. Timofeev, P. Q. Nelson, G. S. Corrado, J. D. Hipp, L. Peng, and M. C. Stumpe, "Detecting Cancer Metastases on Gigapixel

Pathology Images," *CoRR*, vol. abs/1703.0, Mar. 2017. [Online]. Available: http://arxiv.org/abs/1703.02442

[5] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, R. Kim, R. Raman, P. C. Nelson, J. L. Mega, and D. R. Webster, "Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs," *JAMA*, vol. 316, no. 22, pp. 2402–2410, Dec. 2016.

[6] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, July 2015.

[7] J. Y. Halpern, *Reasoning about uncertainty*, 2nd ed. Cambridge, MA: The MIT Press, 2003.

[8] NHTSA, "Tesla Crash Preliminary Evaluation Report : The Office of Defects Investigation (ODI) PE 16-007," U.S. Department of Transportation, National Highway Traffic Safety Administration, Washington, DC, Tech. Rep., Jan. 2017. [Online]. Available: https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.pdf

[9] R. R. Yager, J. Kacprzyk, and M. Fedrizzi, Eds., *Advances in the Dempster-Shafer Theory of Evidence.* New York: Wiley, 1994.

[10] P. Smets, "Practical uses of belief functions," in *Proc. 15th Conf. Uncertainty in Artificial Intelligence (UAI)*, Stockholm, Sweden, July 1999, pp. 612–621.

[11] R. R. Yager and L. Liu, Eds., *Classic Works of the Dempster-Shafer Theory of Belief Functions.* Berlin Heidelberg: Springer-Verlag, 2008.

[12] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," *Ann. Math. Stat.*, vol. 38, no. 2, pp. 325–339, Apr. 1967.

[13] ——, "A generalization of Bayesian inference," *J. R. Stat. Soc. Ser. B*, vol. 30, no. 2, pp. 205–247, 1968.

[14] G. Shafer, *A Mathematical Theory of Evidence.* Princeton, NJ: Princeton Univ. Press, 1976.

[15] J. A. Barnett, "Computational methods for A mathematical theory of evidence," in *Proc. 7th Int. Joint Conf. Artificial Intelligence (IJCAI)*, Vancouver, Canada, Aug. 1981, pp. 868–875.

[16] P. Smets and R. Kennes, "The transferable belief model," *Artif. Intell.*, vol. 66, no. 2, pp. 191–234, Apr. 1994.

[17] J. Kohlas and P.-A. Monney, *A Mathematical Theory of Hints*, 1st ed. Berlin Heidelberg: Springer-Verlag, 1995, vol. 425.

[18] T. Denœux, "40 years of Dempster-Shafer theory," *Int. J. Approx. Reason.*, vol. 79, no. C, pp. 1–6, Dec. 2016.

[19] J. N. Heendeni, K. Premaratne, M. N. Murthi, J. Uscinski, and M. Scheutz, "A generalization of Bayesian inference in the Dempster-Shafer belief theoretic framework," in *Proc. 19th Int. Conf. Information Fusion (FUSION)*, Heidelberg, Germany, July 2016, pp. 798–804.

[20] M. S. Tonelli-Neto, J. G. M. Decanini, A. D. P. Lotufo, and C. R. Minussi, “Fuzzy based methodologies comparison for high-impedance fault diagnosis in radial distribution feeders,” *IET Gener. Transm. Distrib.*, vol. 11, no. 6, pp. 1557–1565, Apr. 2017.

[21] T. Kari, W. Gao, D. Zhao, Z. Zhang, W. Mo, Y. Wang, and L. Luan, “An integrated method of ANFIS and Dempster-Shafer theory for fault diagnosis of power transformer,” *IEEE Trans. Dielectr. Electr. Insul.*, vol. 25, no. 1, pp. 360–371, Feb. 2018.

[22] G. Shafer, “Perspectives on the theory and practice of belief functions,” *Int. J. Approx. Reason.*, vol. 4, no. 5-6, pp. 323–362, Sept.–Nov. 1990.

[23] P. Orponen, “Dempster's Rule of Combination is #P-complete,” *Artif. Intell.*, vol. 44, no. 1-2, pp. 245–253, July 1990.

[24] V. Y. Kreinovich, A. Bernat, W. Borrett, Y. Mariscal, and E. Villa, “Monte-Carlo methods make Dempster-Shafer formalism feasible,” NASA Johnson Space Center, Houston, TX, Tech. Rep., Jan. 1991. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=19930015951

[25] G. Shafer, “A Mathematical Theory of Evidence turns 40,” *Int. J. Approx. Reason.*, vol. 79, pp. 7–25, Dec. 2016.

[26] F. Voorbraak, “A computationally efficient approximation of Dempster-Shafer theory,” *Int. J. Man-Mach. Stud.*, vol. 30, no. 5, pp. 525–536, May 1989.

[27] D. Dubois and H. Prade, “Consonant approximations of belief functions,” *Int. J. Approx. Reason.*, vol. 4, no. 5-6, pp. 419–449, Sept.–Nov. 1990.

[28] B. Tessem, “Approximations for efficient computation in the theory of evidence,” *Artif. Intell.*, vol. 61, no. 2, pp. 315–329, June 1993.

[29] M. Bauer, “Approximation algorithms and decision making in the Dempster-Shafer theory of evidence — An empirical study,” *Int. J. Approx. Reason.*, vol. 17, no. 2-3, pp. 217–237, Aug.–Oct. 1997.

[30] D. Harmanec, “Faithful approximations of belief functions,” in *Proc. 15th Conf. Uncertainty in Artificial Intelligence (UAI)*, Stockholm, Sweden, July 1999, pp. 271–278.

[31] T. Denœux, “Inner and outer approximation of belief structures using a hierarchical clustering approach,” *Int. J. Uncertainty, Fuzziness Knowlege-Based Syst.*, vol. 9, no. 4, pp. 437–460, Aug. 2001.

[32] R. Haenni and N. Lehmann, “Resource bounded and anytime approximation of belief function computations,” *Int. J. Approx. Reason.*, vol. 31, no. 1-2, pp. 103–154, Oct. 2002.

[33] H. M. Thoma, “Factorization of Belief Functions,” Ph.D. dissertation, Dept. Stat., Harvard Univ., Cambridge, MA, 1989.

[34] R. Kennes and P. Smets, “Fast algorithms for Dempster-Shafer theory,” in *Proc. 3rd Int. Conf. Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, Paris, France, July 1990, pp. 14–23.

[35] H. M. Thoma, “Belief function computations,” in *Conditional Logic in Expert Systems*, I. R. Goodman, M. M. Gupta, H. T. Nguyen, and G. S. Rogers, Eds. Amsterdam, The Netherlands: Elsevier, 1991, ch. 9, pp. 269–308.

[36] SIPTA, "Software tools for imprecise probabilities," 2018. [Online]. Available: http://www.sipta.org/index.php?id=sfw

[37] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* San Francisco, CA: Morgan Kaufmann, 1988.

[38] G. Shafer, "Jeffrey's Rule of Conditioning," *Philos. Sci.*, vol. 48, no. 3, pp. 337–362, Sept. 1981.

[39] H. Ichihashi and H. Tanaka, "Jeffrey-like rules of conditioning for the Dempster-Shafer theory of evidence," *Int. J. Approx. Reason.*, vol. 3, no. 2, pp. 143–156, Mar. 1989.

[40] R. Fagin and J. Y. Halpern, "A new approach to updating beliefs," in *Proc. 6th Conf. Uncertainty in Artificial Intelligence (UAI)*, Cambridge, MA, July 1990, pp. 347–374.

[41] P. Smets, "About updating," in *Proc. 7th Conf. Uncertainty in Artificial Intelligence (UAI)*, Los Angeles, CA, July 1991, pp. 378–385.

[42] C. Yu and F. Arasta, "On conditional belief functions," *Int. J. Approx. Reason.*, vol. 10, no. 2, pp. 155–172, Feb. 1994.

[43] F. Klawonn and P. Smets, "The dynamic of belief in the transferable belief model and specialization-generalization matrices," in *Proc. 8th Conf. Uncertainty in Artificial Intelligence (UAI)*, Stanford, CA, July 1992, pp. 130–137.

[44] H. T. Nguyen and P. Smets, "On dynamics of cautious belief and conditional objects," *Int. J. Approx. Reason.*, vol. 8, no. 2, pp. 89–104, Feb. 1993.

[45] H. Xu and P. Smets, “Reasoning in evidential networks with conditional belief functions,” *Int. J. Approx. Reason.*, vol. 14, no. 2-3, pp. 155–185, Feb.–Apr. 1996.

[46] P. Smets, “The application of the matrix calculus to belief functions,” *Int. J. Approx. Reason.*, vol. 31, no. 1-2, pp. 1–30, Oct. 2002.

[47] K. Premaratne, M. Murthi, J. Zhang, M. Scheutz, and P. Bauer, “A Dempster-Shafer theoretic conditional approach to evidence updating for fusion of hard and soft data,” in *Proc. 12th Int. Conf. Information Fusion (FUSION)*, Seattle, WA, July 2009, pp. 2122–2129.

[48] T. L. Wickramarathne, K. Premaratne, M. N. Murthi, M. Scheutz, S. Kübler, and M. Pravia, “Belief theoretic methods for soft and hard data fusion,” in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Prague, Czech Republic, May 2011, pp. 2388–2391.

[49] R. Kennes and P. Smets, “Computational aspects of the Möbius transform,” in *Proc. 6th Conf. Uncertainty in Artificial Intelligence (UAI)*, Cambridge, MA, July 1990, pp. 344–351.

[50] R. Kennes, “Computational aspects of the Möbius transformation of graphs,” *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 2, pp. 201–223, Mar./Apr. 1992.

[51] H. Xu and R. Kennes, “Steps toward efficient implementation of Dempster-Shafer theory,” in *Advances in the Dempster-Shafer Theory of Evidence*, R. R. Yager, J. Kacprzyk, and M. Fedrizzi, Eds. New York: Wiley, 1994, ch. 8, pp. 153–174.

[52] N. Wilson, “Algorithms for Dempster-Shafer theory,” in *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, D. M. Gabbay and P. Smets, Eds. Dordrecht, The Netherlands: Kluwer Academic, 2000, vol. 5, ch. 10, pp. 421–475.

[53] R. Haenni and N. Lehmann, “Implementing belief function computations,” *Int. J. Intell. Syst.*, vol. 18, no. 1, pp. 31–49, Jan. 2003.

[54] N. Lehmann, “Fast projection of focal sets,” in *Proc. 3rd Int. Conf. Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Melbourne, Australia, Dec. 2007, pp. 191–196.

[55] T. Augustin, F. P. A. Coolen, G. de Cooman, and M. C. M. Troffaes, Eds., *Introduction to Imprecise Probabilities.* Chichester, UK: Wiley, 2014.

[56] L. G. Polpitiya, K. Premaratne, M. N. Murthi, and D. Sarkar, “A framework for efficient computation of belief theoretic operations,” in *Proc. 19th Int. Conf. Information Fusion (FUSION)*, Heidelberg, Germany, July 2016, pp. 1570–1577.

[57] ——, “Efficient computation of belief theoretic conditionals,” in *Proc. 10th Int. Symp. Imprecise Probability: Theories and Applications (ISIPTA)*, Lugano, Switzerland, July 2017, pp. 265–276.

[58] T. L. Wickramarathne, K. Premaratne, and M. N. Murthi, “Focal elements generated by the Dempster-Shafer theoretic conditionals: a complete characterization,” in *Proc. 13th Int. Conf. Information Fusion (FUSION)*, Edinburgh, UK, July 2010, pp. 1–8.

[59] ——, "Toward efficient computation of the Dempster-Shafer belief theoretic conditionals," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 712–724, Apr. 2013.

[60] T. A. Defanti and M. D. Brown, "Visualization in Scientific Computing," *Adv. Comput.*, vol. 33, pp. 247–307, Jan. 1991.

[61] A. Cairo, *The Functional Art: An introduction to information graphics and visualization.* San Francisco, CA: Pearson Education, Peachpit, 2013.

[62] ProFuSELab, "Belief Computation Library," 2016. [Online]. Available: https://github.com/ProFuSELab/Belief-Computation-Library

[63] L. Liu, "A relational representation of belief functions," in *Proc. 3rd Int. Conf. Belief Functions (BELIEF)*, Oxford, UK, Sept. 2014, pp. 161–170.

[64] H. Wu, "Sensor Data Fusion for Context-Aware Computing Using Dempster-Shafer Theory," Ph.D. dissertation, Robotics Inst., Carnegie Mellon Univ., Pittsburgh, PA, 2003.

[65] ProFuSELab, "Conditional Computation Library," 2017. [Online]. Available: https://profuselab.github.io/Conditional-Computation-Library/

[66] L. G. Polpitiya, K. Premaratne, M. N. Murthi, and D. Sarkar, "Efficient and Exact Computation of Conditionals in the Dempster-Shafer Belief Theoretic Framework," 2018.

[67] ProFuSELab, "DS-CONAC: DS-Conditional-One and DS-Conditional-All in C++," 2018. [Online]. Available: https://profuselab.github.io/DS-CONAC/

[68] T. L. Wickramarathne, K. Premaratne, and M. N. Murthi, “Consensus-Based Credibility Estimation of Soft Evidence for Robust Data Fusion,” in *Proc. 2nd Int. Conf. Belief Functions (BELIEF)*, Compiègne, France, May 2012, pp. 301–309.

[69] P. Smets, “Decision Making in a Context where Uncertainty is Represented by Belief Functions,” in *Belief functions in business decisions*, R. P. Srivastava and T. J. Mock, Eds. Heidelberg, Germany: Physica-Verlag, 2002, vol. 88, ch. 2, pp. 17–61.

[70] ——, “Decision making in the TBM: the necessity of the pignistic transformation,” *Int. J. Approx. Reason.*, vol. 38, no. 2, pp. 133–147, Feb. 2005.

[71] A. Jøsang and Z. Elouedi, “Interpreting Belief Functions as Dirichlet Distributions,” in *Proc. European Conf. Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, Hammamet, Tunisia, Oct. 2007, pp. 393–404.

[72] A. Jøsang, J. Diaz, and M. Rifqi, “Cumulative and averaging fusion of beliefs,” *Inf. Fusion*, vol. 11, no. 2, pp. 192–200, Apr. 2010.

[73] R. F. Hanssen, *Radar Interferometry - Data Interpretation and Error Analysis*, ser. Remote Sensing and Digital Image Processing. Dordrecht, The Netherlands: Kluwer Academic, 2001, vol. 2.

[74] A. K. Gabriel, R. M. Goldstein, and H. A. Zebker, “Mapping small elevation changes over large areas: Differential radar interferometry,” *J. Geophys. Res.*, vol. 94, no. B7, pp. 9183–9191, Jul 1989.

[75] H. A. Zebker, P. A. Rosen, R. M. Goldstein, A. Gabriel, and C. L. Werner, "On the derivation of coseismic displacement fields using differential radar interferometry: The Landers earthquake," *J. Geophys. Res.*, vol. 99, no. B10, pp. 19 617–19 634, Oct. 1994.

[76] D. Massonnet and K. L. Feigl, "Radar interferometry and its application to changes in the Earth's surface," *Rev. Geophys.*, vol. 36, no. 4, pp. 441–500, Nov. 1998.

[77] F. Amelung, D. L. Galloway, J. W. Bell, H. A. Zebker, and R. J. Laczniak, "Sensing the ups and downs of Las Vegas: InSAR reveals structural control of land subsidence and aquifer-system deformation," *Geology*, vol. 27, no. 6, pp. 483–486, June 1999.

[78] S. Jónsson, H. Zebker, P. Cervelli, P. Segall, H. Garbeil, P. Mouginis-Mark, and S. Rowland, "A shallowdipping dike fed the 1995 flank eruption at Fernandina Volcano, Galápagos, observed by satellite radar interferometry," *Geophys. Res. Lett.*, vol. 26, no. 8, pp. 1077–1080, Apr. 1999.

[79] F. Amelung, S. Jónsson, H. Zebker, and P. Segall, "Widespread uplift and trapdoor' faulting on Galápagos volcanoes observed with radar interferometry," *Nature*, vol. 407, no. 6807, pp. 993–996, Oct. 2000.

[80] M. Newman, *Networks*, 1st ed. New York: Oxford Univ. Press, 2010.

[81] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[82] J. Venn, “On the diagrammatic and mechanical representation of propositions and reasonings,” *London, Edinburgh, Dublin Philosoph. Mag. J. Sci.*, vol. 10, no. 59, pp. 1–18, July 1880.

[83] R. Sedgewick, *Algorithms in C++*, 3rd ed. Boston, MA: Addison-Wesley, 2002.

[84] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: The MIT Press, 2009.

[85] E. C. Kulasekere, K. Premaratne, D. A. Dewasurendra, M.-L. Shyu, and P. H. Bauer, “Conditioning and updating evidence,” *Int. J. Approx. Reason.*, vol. 36, no. 1, pp. 75–108, Apr. 2004.

[86] L. G. Polpitiya, K. Premaratne, M. N. Murthi, and D. Sarkar, “Data Structures Toward Precise Real-Time Computations of Belief Theoretic Operations on Dynamic Frames of Discernment,” 2018.

[87] G. Shafer, “Dempster’s rule of combination,” *Int. J. Approx. Reason.*, vol. 79, pp. 26–40, Dec. 2016.

[88] S. Destercke and T. Burger, “Toward an axiomatic definition of conflict between belief functions,” *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 585–596, Apr. 2013.

[89] H. Fattahi and F. Amelung, “DEM Error Correction in InSAR Time Series,” *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 7, pp. 4249–4259, July 2013.

[90] S. Baker and F. Amelung, “Top-down inflation and deflation at the summit of Klauea Volcano, Hawaii observed with InSAR,” *J. Geophys. Res.*, vol. 117, no. B12406, Dec. 2012.

[91] R. Wagstaff, “Exploiting Phase Fluctuations to Improve Temporal Coherence,” *IEEE J. Ocean. Eng.*, vol. 29, no. 2, pp. 498–510, Apr. 2004.

[92] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, and G. Brain, “TensorFlow: A system for large-scale machine learning,” in *Proc. 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, Savannah, GA, Nov. 2016, pp. 265–278.

[93] W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proc. 9th Python in Science Conf. (SciPy)*, Austin, TX, Jun 2010, pp. 51–56.

[94] E. Jones, T. Oliphant, P. Peterson *et al.*, “{SciPy}: Open source scientific tools for {Python},” 2001. [Online]. Available: http://www.scipy.org/

[95] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, no. Oct., pp. 2825–2830, 2011.

[96] F. Ruskey, “A Survey of Venn Diagrams,” *Electron. J. Combinatorics*, vol. 4, p. 3, Feb. 1997. [Online]. Available: http://www.combinatorics.org/files/Surveys/ds5/ds5v1-1997/VennEJC.html

[97] A. Edwards, “Venn diagrams for many sets,” *New Scientist*, vol. 121, no. 1646, pp. 51–56, Jan. 1989.

[98] T. More and Jr., "On the construction of venn diagrams," *J. Symbolic Logic*, vol. 24, no. 4, pp. 303–304, Dec. 1959.

[99] D. E. Anderson and F. L. Cleaver, "Venn-type diagrams for arguments of n terms," *J. Symbolic Logic*, vol. 30, no. 2, pp. 113–118, Jun. 1965.

[100] B. Grünbaum, "Venn Diagrams II," *Geombinatorics*, vol. 2, no. 2, pp. 25–32, Sept. 1992.

[101] B. Grunbaum, "On Venn Diagrams and the Counting of Regions," *College of Mathematics J.*, vol. 15, no. 5, pp. 433–435, Nov. 1984.

[102] J. J. Carroll, "Drawing Venn Triangles," Hewlett-Packard Laboratories, Bristol, UK, Tech. Rep., Dec 2000. [Online]. Available: http://www.hpl.hp.com/techreports/2000/HPL-2000-73.html?mtxs=rss-hpl-tr

[103] B. Grünbaum, "Venn Diagrams and Independent Families of Sets," *Mathematics Mag.*, vol. 48, no. 1, pp. 12–23, Jan. 1975.

[104] P. Hamburger, "Doodles and doilies, non-simple symmetric Venn diagrams," *Discrete Mathematics*, vol. 257, no. 2-3, pp. 423–439, Nov. 2002.

[105] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, Nov. 1991.

[106] Y. Hu, "Efficient, High-Quality Force-Directed Graph Drawing," *Mathematica J.*, vol. 10, no. 1, pp. 37–71, 1999.

[107] C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing," in *Proc. 8th Int. Symp. Graph Drawing*, Colonial Williamsburg, VA, Sept. 2000, pp. 171–182.