2004

Software quality and reliability prediction using Dempster -Shafer theory

Lan Guo
West Virginia University

Software Quality and Reliability Prediction Using Dempster-Shafer Theory

Lan Guo

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer and Information Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2004

Keywords: Dempster-Shafer Theory, Dempster-Shafer Belief Network, Information Fusion, Software Quality Prediction, Software Reliability Assessment

ABSTRACT

Software Quality and Reliability Prediction Using Dempster-Shafer Theory

Lan Guo

As software systems are increasingly deployed in mission critical applications, accurate quality and reliability predictions are becoming a necessity. Most accurate prediction models require extensive testing effort, implying increased cost and slowing down the development life cycle. We developed two novel statistical models based on Dempster-Shafer theory, which provide accurate predictions from relatively small data sets of direct and indirect software reliability and quality predictors. The models are flexible enough to incorporate information generated throughout the development life-cycle to improve the prediction accuracy.

Our first contribution is an original algorithm for building Dempster-Shafer Belief Networks using prediction logic. This model has been applied to software quality prediction. We demonstrated that the prediction accuracy of Dempster-Shafer Belief Networks is higher than that achieved by logistic regression, discriminant analysis, random forests, as well as the algorithms in two machine learning software packages, See5 and WEKA. The difference in the performance of the Dempster-Shafer Belief Networks over the other methods is statistically significant.

Our second contribution is also based on a practical extension of Dempster-Shafer theory. The major limitation of the Dempsters rule and other known rules of evidence combination is the inability to handle information coming from correlated sources. Motivated by inherently high correlations between early life-cycle predictors of software reliability, we extended Murphys rule of combination to account for these correlations. When used as a part of the methodology that fuses various software reliability prediction systems, this rule provided more accurate predictions than previously reported methods. In addition, we proposed an algorithm, which defines the upper and lower bounds of the belief function of the combination results. To demonstrate its generality, we successfully applied it in the design of the Online Safety Monitor, which fuses multiple correlated time varying estimations of convergence of neural network learning in an intelligent flight control system.

Acknowledgments

I would like to take this opportunity to say thanks to my advisor Dr. Bojan Cukic. He has not only inspired me through the research, but also shown his kindness, patience, perseverance, creativity, and scientific research approach to me. I am thankful for his solid training. Without all his help, I could not have accomplished what I have now. My Ph.D. projects were supported by the NSF grant CCR-0093315 and NASA Cooperative Agreement NCC5-685.

I would like to express my appreciation to Dr. Katerina Goseva-Popstojanova, Dr. Harshinder Singh, and Dr. Carol Smidts, who have given me great scientific guidance and support. Without their help, I could not have gone through my Ph.D. study within the expected time. I would like to thank Dr. Supratik Mukhopadhyay for guiding me through his two great projects, which have broadened my knowledge. Although they are not included in this dissertation, I am thankful for his valuable time and instructions.

I would like to thank all faculties and staff in our department. They create such a wonderful atmosphere that I enjoy every day of studying and working here.

I would like to thank my lab colleagues, Paola Bracchi, Dejan Desovski, Vijai Gandikota, Yue Jiang, Yan Liu, Martin Mladenovski, Petar Popic, and Sampath Yerramalla for all their help, support and friendship. I appreciate the amiable working environment.

I would like to thank Yan Ma, Sujan Parthasaradhi, Ming Li, and Avik Sinha for their help and cooperation in my Ph. D. projects.

Last but not least, I would like to thank my family, mom, dad, and my husband, Yong, for their understanding, sacrifice, and support. Without their love and support, I could not have gone this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As software systems are being increasingly deployed in mission critical applications, it has become imperative to accurately predict software quality and reliability before their deployments. Most accurate prediction models, however, require extensive testing effort, implying increasing cost and slowing down of the development life-cycle. There are many indirect assessment models, which can provide reliability prediction without extensive testing. However, these models are mostly inaccurate, and many of their predictions are statistically correlated. There remains a problem to combine these indirect reliability models for a more accurate prediction, and meanwhile, take correlation into account.

Accurate software quality and reliability prediction relies on good statistical models. Current models are mainly focused on two areas: evidence combination and probability reasoning under uncertainty. There are two important statistical theories for evidence combination: Bayesian theory and Dempster-Shafer (D-S) theory. Based on these two theories, two probability reasoning methodologies were developed: Bayesian Belief Networks (BBNs) and Dempster-Shafer Belief Networks (DSBNs). Both methodologies can be used to make predictions about unobserved variables based on observed ones by inference propagation through belief networks.

Bayesian inference methods have been widely applied in various application domains, especially in software engineering. However, as pointed out by Shafer [136], the way that the Bayesian theory handles complete ignorance is problematic. In addition, the Bayseian theory requires a solid *subjective prior*, which is often unobtainable. Rooted from the Bayesian theory, the Bayesian Belief Networks also suffer from such limitations. Additionally, the structure of the BBNs is difficult to build with missing data.

On the other hand, the D-S theory uses an *uncertainty factor* to represent ignorance, and does not require any *prior* for inference. Furthermore, Shafer proved that the Bayes' rule of conditioning is a restricted special case of the Demspter's rule of combination. Later, the Dempster-Shafer Belief Networks (DSBNs) were developed [94], which can be inducted from a relatively small data set in a more flexible and dynamic way compared to the BBNs. However, this promising probability reasoning methodology has not been applied to software engineering.

Another open problem in this research area is that both the Bayesian theory and the D-S theory cope with independent evidence. This assumption of "evidence independence" is restrictive and admittedly unrealistic in many applications, for instance when sources of evidence are correlated with each other.

Our research goal is to develop a reasoning methodology which is objective, flexible and dynamic, such that it overcomes the limitations of the Bayesian Belief Networks which are subjective and difficult to build. In addition, we would like to develop a general methodology that can combine evidence in general, including both correlated and uncorrelated evidence.

We have developed two novel statistical models based on Dempster-Shafer theory, and applied them to software quality and reliability prediction. Specifically, we developed a novel induction algorithm to build the Dempster-Shafer (D-S) networks. In addition, we developed a general methodology for evidence combination, which is based on the Murphy's rule of combination and fuzzy logic. This methodology can combine both correlated and uncorrelated evidence.

This dissertation first introduces a probability reasoning methodology based on the Dempster-Shafer (D-S) belief networks. We applied the methodology to predicting fault prone modules. It consists of three major parts: First, building the Dempster-Shafer network from an existing data set by the induction algorithm; Second, choosing the predictors (attributes) by feature selection; Third, feeding the predictors describing the modules of the current project into the inducted Dempster-Shafer network and identifying fault prone modules. We applied this methodology in two case studies based on NASA data sets. The prediction accuracy of our methodology is higher than that achieved by logistic regression, discriminant analysis, random forests, as well as the algorithms in two machine learning software packages See5 and WEKA on the same data sets. The difference in the performance of the proposed methodology over other methods is statistically significant.

The second methodology introduced focuses on information fusion, an important area of evidence combination. It is an extension of the Dempster-Shafer framework, which can combine both correlated and uncorrelated evidence. This framework is based on the Murphy's rule of combination and fuzzy logic. We applied it to the fusion of various software reliability prediction systems for a more precise prediction of software reliability. The prediction result is more accurate than previously reported methods. The proposed methodology was also applied in a realtime intelligent flight control system. The Online Safety Monitor constructed based on this methodology can accurately detect off-nominal behavior in the flight pattern data. This general framework can be applied to combining evidence for prediction in many other research areas.

This dissertation is organized as follows. Chapter 2 presents related work. It focuses on comparisons of the Bayesian theory and the D-S theory, as well as the Bayesian Belief Networks and the Dempster-Shafer Belief Networks. In addition, it overviews research in software quality and reliability prediction. Chapter 3 introduces our induction algorithm and the modifications made in the belief revision algorithm for the Dempster-Shafer networks. Chapter 4 applies the first methodology to predicting fault-prone modules in software engineering. Chapter 5 outlines the second methodology for information fusion of correlated evidence. Chapter 6 describes two applications of the proposed information fusion methodology. Finally, Chapter 7 summarizes our contributions and future work.

Chapter 2

Related Work

Statistical theories are very important in building various prediction models, which have been widely applied in software engineering, medical diagnosis, and recently bioinformatics. These models can be categorized into two areas: evidence combination, and probability reasoning under uncertainty. Evidence combination refers to combining available evidence to evaluate certain proposition(s), for instance to judge whether or not a coin is biased based on the frequencies of its head and tail during tossing. A good example for reasoning under uncertainty is medical diagnosis, where doctors make inference of a patient's possible disease based on the observable symptoms.

There are two most important statistical theories for evidence combination: Bayesian theory and Dempster-Shafer (D-S) theory. Based on these two theories, two inference methodologies were developed for reasoning under uncertainty: Bayesian Belief Networks and Dempster-Shafer Belief Networks.

The Bayesian theory is a very popular theory and has been extensively applied in many applications. However, it has also been questioned for its "subjectiveness" and its way of handling complete ignorance. In this chapter, we will introduce an alternative theory, Dempster-Shafer theory, and discuss why many researchers consider it more general and robust than the Bayesian theory.

The Bayesian Belief Networks (BBNs) have been deemed as the most promising inference methodology by many researchers and thus employed in numerous prediction applications. However, the Bayesian networks have drawbacks rooted in the Bayesian theory, and are difficult to build with missing data. On the other hand, the Dempster-Shafer Belief Networks (DSBN) were recently developed [94] and are more flexible than the BBNs. However, they have not been applied in software engineering.

As mentioned before, statistical models and machine learning algorithms are important for software quality and reliability prediction. Software engineers need accurate quality assessment of the software under development. Early prediction of fault prone components is of particular interest for software developers to quickly find defects and deliver more reliable software products. Software quality prediction focuses on identifying fault prone modules (procedures), while software reliability prediction aims at quantifying the probability that a program will execute without failure since its start time. Models have been developed to incorporate product and process metrics for software quality and reliability prediction.

Software quality models aim at predicting critical software components prior to testing. They are generally built from metrics collected in past projects or releases, and are used to identify fault prone modules in the current project/release and subject them to more rigorous verification activities. Successful models are characterized by high prediction accuracy, thus allowing software developers to quickly identify defects early in the software life cycle. Automated detection of fault prone modules during software development process is an important prerequisite for developing reliable large systems.

Software reliability is a statistical measure of how well software operates with respect to its requirements. There are two related software engineering research issues about reliability requirements. The first issue is *achieving* the necessary reliability, i.e., choosing and employing appropriate software engineering techniques in system design and implementation. The second issue is the *assessment* of reliability as a method of assurance that precedes system deployment. We are interested in the second issue in this dissertation.

Currently, there are many indirect software reliability assessment approaches, aiming at predicting software reliability in early life cycle. Most of them are, however, not accurate enough for reliable predictions. In addition, many of them are statistically dependent. Therefore, there is an urgent need for such an evidence combination framework that can integrate them for a more precise prediction, and take correlation into account.

In this chapter, we first introduce the two statistical theories for evidence combination (§2.1), the Bayesian theory and the D-S theory, and illustrate why Bayes' theorem is contained in the D-S scheme. Secondly, we discuss the two inference methodology for reasoning under uncertainty (§2.2), the Bayesian belief networks and the Dempster-Shafer belief networks. Thirdly, we introduce related work in software quality prediction (§2.3), and then software reliability assessment (§2.4). Finally, we summarize the chapter (§2.5).

2.1 Evidence Combination

The fundamental operation of probability reasoning is the combination of evidence. Information fusion is a very important area of evidence combination. The goal is to fuse different sources of information for a more precise probabilistic assessment of real-world representations. A variety of approaches have been presented in the literature, including Bayesian methods [56] [123] [125] [129] [142], Dempster-Shafer Theory [51] [95] [113], Artificial Neural Networks [15] [49] [100], Fuzzy reasoning [49] [114] [120] [131], Classifier Fusion Strategies (average, minimum, maximum, median, majority vote, and oracle) [77] [84], rule-based expert systems [151], Independent Component Analysis [50], Principal Component Analysis [32], Correspondence Factorial Analysis [32], Minimum Entropy fusion approach [162], Kalman filtering [61], weighted Boolean models [106], and Ordered Weighted Averaging (OWA) operator [161].

Most fusion techniques share a common assumption that different sources of information are independent from each other, which is restrictive and unrealistic in many situations. For instance, it is recognized that "independently built" classifiers exhibit positive correlation, and hence conceptualizing and quantifying diversity between classifier outputs is a challenging task on its own and will add a whole new dimension to classifier fusion [84]. Similar observation is also reported in applying Bayesian methods [142]. Research has been conducted to exploit the dependencies in information fusion. A method based on the Bayesian inference and the maximum entropy principle was proposed to tackle tightly-coupled fusion case [123]. Its solution happens to be equivalent to Canonical Correlation Analysis (CCA), which is similar to the Principal Component Analysis (PCA) method. Both CCA and PCA involve a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables. The reduced number of uncorrelated variables can then be used by various statistical methods for prediction.

The Murphy's rule of combination was proposed for the Dempster-Shafer framework [113], when the evidence is from the *same* source. However, there are other situations where sources of information are *not* distinct, but is *not* from the same source, either. Correlated evidence is one such example. Currently, no framework exists to combine evidence and take correaltion into account.

This section introduces two theories for evidence combination: the Bayesian theory (§2.1.1) and the Dempster-Shafer theory (§2.1.2). In §2.1.3, we illustrate Shafer's proof that Bayes' theorem is a special case of the Dempster's rule of combination [136].

2.1.1 Bayesian Theory

The Bayesian theory is a very popular theory of partial belief. It adopts the three basic rules of the frequentist approach as rules for one's degree of belief based on a given source of evidence, and it adopts the rule of conditioning as a general rule for updating one's belief when that evidence is augmented by the knowledge of a particular proposition.

The three basic rules of the frequentist approach are listed below. $P(A)$ represents the probability of an uncertain event A, which is defined by the frequency of that event based on previous observations:

1. $P(A) \to [0, 1]$.
2. If A represents a certain event, then $P(A) = 1$.
3. If A and B are mutually exclusive events, then $P(A \cup B) = P(A) + P(B)$.

Based on the third rule, it is derived that $P(A) = 1 - P(\neg A)$. This is also called *Bayes' rule of additivity.*

In the Bayesian theory, a subjective *prior* is first assigned to a proposition. Upcoming evidence is then incorporated to update the prior for a posterior probability. When we are totally ignorant about the proposition, each possible outcome is assigned a uniform prior. Bayes' theorem can be represented as:

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)} \tag{2.1}$$

where A is the vector of parameters we are interested in and B is the vector of sample observations. It is also called *Bayes' rule of conditioning.* In Equation 2.1, $P(A \mid B)$ is the posterior density function for A summarizing all the information about B, $P(B \mid A)$ is the sample information algebraically equivalent to the likelihood for A, and $P(A)$ is the prior representing the expert opinion about A. Therefore, Equation 2.1 can be presented as [28]:

$$P(A \mid B) \propto P(A) l(A \mid B) \tag{2.2}$$

In words, Equation 2.2 means:

$$\textit{Posterior} \propto \textit{Prior} \times \textit{Sample}$$

While Bayes' theorem is unanimously agreed as correct when such *prior* is a valid one, there is no guarantee that a subjective belief always represents the reality. Thus, Bayes' theorem is argued for its "subjectiveness" by many researchers. Bayes' theorem has an additional weakness in its way of representing complete ignorance: in this case, each possible outcome is assigned a uniform prior. Firstly, it might be confusing for people to interpret uniform priors: do all possible outcomes *actually* have the same probability, or we just don't have any clue about them? Secondly, the opponents of Bayes' theorem consider it logically inconsistent as it assigns uniform priors to represent complete ignorance. Here is a counter example: suppose we would like to predict whether or not the system has any defects. We have two possible outcomes for the proposition: *the system has a defect* and *the system has no defects.* Without any information, we should assign uniform priors to these two possible outcomes according to Bayes' theorem. Therefore, each possible outcome has probability of 0.5. Now, suppose we have a more refined set of possible outcomes: *the system has a software defect but no hardware defects, the system has a hardware defect but no software defects, the system has both software and hardware defects*, and *the system has no defects.* We notice that the first three outcomes of the more refined set can be combined as *the system has a defect.* With conventional uniform priors, we have 0.25 probability for each possible outcome. Combining the first three outcomes, we have 0.75 probability for *the system has a defect* and 0.25 probability for *the system has no defects.* A self-contradiction is evident in this case.

2.1.2 Dempster-Shafer Theory

Dempster-Shafer (D-S) theory is a complete formalism of evidential reasoning for computing and propagating evidential support (either confirming or disconfirming). It is considered a more general and robust theory than Bayes' theorem by its proponents [136]. Shafer demonstrated that the Bayesian theory is contained in the D-S theory as a restrictive special case, because Bayes' rule of conditioning is a special case of Dempster's rule of combination [136]. The difference between these two theoretical frameworks lies in that the D-S theory, unlike Bayes' theorem, does not require any priors. More importantly, it uses an *uncertainty factor* to represent ignorance. In the D-S theory, the belief for the proposition starts from *zero.* At this point, the uncertainty factor (representing the ignorance) is 1. Based on upcoming evidence, the belief for the proposition is updated by the Dempster's rule of combination [136]. The more we believe in the proposition, the less the uncertainty factor. In this way, the assessment for the proposition is very clear.

In the D-S theory, parameters such as *events* in probability theory are called *propositions* $\{\theta_1, \theta_2, \cdots, \theta_n\}$, and they must be mutually exclusive. The set of finite parameters $\Theta = \{\theta_i\}$ is called the *frame of discernment.* If 2^Θ denotes the power set of Θ, then 2^Θ includes all possible propositions of interest. In order to express the belief assigned to a proposition, we call function: $m : 2^{\Theta} \rightarrow [0, 1]$ a *basic probability assignment*, which satisfies:

1. $m(\emptyset) = 0$.

2. $\sum_{A \subset \Theta} m(A) = 1$.

The quantity $m(A)$ is called A's *basic probability number.* It is interpreted as the measure of the belief that is committed exactly to A and to no smaller subset.

The D-S theory defines *belief function* over the frame of discernment Θ as Bel. For each subset A of Θ, the number $Bel(A)$ can be understood as one's *degree of belief* that A is true. A subset A of Θ is called a *focal element* of a belief function Bel over Θ if $m(A) > 0$. While the quantity $m(A)$ measures the belief that one commits exactly to A, not the total belief that one commits to A, one must add to $m(A)$ the quantities $m(B)$ for all proper subsets B of A to obtain the measure of the total belief committed to A:

$$Bel(A) = \sum_{B \subseteq A} m(B) \tag{2.3}$$

In the D-S theory, belief functions do not have additivity as Bayes' theorem (described in §2.1.1). It means that $Bel(A) + Bel(\overline{A})$ does not have to be 1. This is the fundamental difference from the Bayesian theory. The D-S theory defines *ignorance* (the *uncertainty factor*) as $1 - Bel(A) - Bel(\overline{A})$.

Dempster's rule of combination is defined as follows:

$$m(C) = \frac{\sum_{A_i \bigcap B_j = C; C \neq \emptyset} m_1(A_i) m_2(B_j)}{\sum_{A_i \bigcap B_j \neq \emptyset} m_1(A_i) m_2(B_j)} \tag{2.4}$$

The belief function given by m is called the *orthogonal sum* of m_1 and m_2. This rule means that to combine two bodies of evidence, we get intersection $A_i \bigcap B_j$ for each pair of focal elements A and B, and the combined probability for subset C is the sum of intersections that are subset of C divided by the sum of all nonempty intersections. In the Dempster's rule, if the information given by different sources conflicts, it must be ignored.

Another important measurement is *plausibility*, which is also referred to as *upper probability*: $Pl(A) = 1 - Bel(\overline{A})$.

The Dempster-Shafer (D-S) framework is considered more natural for many application domains [113] [65], since it explicitly represents ignorance, and can incorporate domain knowledge about the inference process via belief functions. As mentioned earlier, the D-S theory is also considered more general and robust than Bayes' theorem. Not only does the Dempster's rule of combination contain the Bayes' rule of conditioning (described in the next section), but also the D-S theory allows different combination rules to fit in this framework [152].

Originally, the D-S theory is a framework that allows combining distinct sources of evidence and predicting under uncertainty. Here, a distinct source of evidence means an independence source of evidence. There are many combination rules for the D-S framework. Two traditional approaches are the Dempster's rule [136] and the Yager's rule [158]. In the Dempster's rule (Equation 2.4), if the information given by different sources conflicts, it must be ignored. On the other hand, Yager proposed that if the information by different sources conflicts, there is no way to tell which information is true. In this case, the conflicted portion is treated as uncertainty and is assigned to the *ignorance factor* ($m(\Theta) \longleftarrow m(\emptyset)$):

$$m(C) = \frac{\sum_{A_i \bigcap B_j = C; C \neq \emptyset} m_1(A_i) m_2(B_j)}{\sum_{A_i \bigcap B_j \in 2^\Theta} m_1(A_i) m_2(B_j)}, C \neq \Theta \tag{2.5}$$

$$m(D) = \frac{\sum_{A_i \bigcap B_j = D} m_1(A_i) m_2(B_j)}{\sum_{A_i \bigcap B_j \in 2^\Theta} m_1(A_i) m_2(B_j)}, D = \Theta \ or \ D = \emptyset \tag{2.6}$$

Both the Dempster's rule and the Yager's rule assign basic probability to all subsets including the empty set. They use orthogonal sum (sum of product of two sources of evidence) to perform evidence combination. The only difference is the way they treat conflicting information, which results in empty set during combination. In the Dempster's rule, the conflicting information is ignored during combination; while in the Yager's rule, it contributes to the uncertainty factor. Therefore, the combination of contradicting information by the Dempster's rule results in "averaged" belief values assigned to the elements in the frame of discernment, which means that contradictory evidence cancels out; while the combination by the Yager's rule results in small belief values assigned to the elements in the frame of discernment, and a big uncertainty factor representing ignorance.

Later, Matsuyama *et al.* proposed an integration method based on the mean of basic probability [101]:

$$m(C) = \sum_{A_i \bigcap B_j = C; C \neq \emptyset} \frac{m(A_i) + m(B_j)}{2} \tag{2.7}$$

This rule is very different from the previous two combination rules (the Dempster's rule or the Yager's rule). Instead of using product of two sources of evidence during combination, Matsuyama's rule uses the mean of two sources of evidence. During the combination, conflicting information is assigned *zero* probability.

Similar to the Matsuyama's rule, Horiuchi proposed a weighted integration method to give different source a different weight during combination [65]:

$$m(C) = \sum_{A_i \bigcap B_j = C} w_i m(A_i) + w_j m(B_j), \ (w_i + w_j = 1) \tag{2.8}$$

This rule also uses the sum, instead of product, to combine evidence. The difference between the Horiuchi's rule and the Matsuyama's rule is that, the Horiuchi's rule adds a weight factor to each source of evidence, and conflicting information is not assigned *zero* probability; while the Matsuyama's rule treats different sources of evidence equally, and conflicting information is assigned *zero* probability.

All the methods mentioned above assume that the sources of evidence are independent from each other. However, some evidence could come from the *same* source. The Murphy's rule of combination was proposed to accommodate this situation.

Murphy proposed an alternative rule of combination for the D-S framework [113]. The Murphy's rule is a general form of the Dempster's rule. It can be applied to the situations where the sources of evidence are either dependent or independent.

The Murphy's rule rewrites the Dempster's rule as:

$$m(C) = \frac{\sum_{A_i \bigcap B_j = C; C \neq \emptyset} f(m_1(A_i) m_2(B_j))}{\sum_{A_i \bigcap B_j \neq \emptyset} f(m_1(A_i) m_2(B_j))}, \tag{2.9}$$

where

$$f(m_1(A_i) m_2(B_j)) = [m_1(A_i) m_2(B_j)]^n, \ (0.0 < n \leq 1.0) \tag{2.10}$$

Function f can be referred to as *belief revision function.* If $n = 1$, it is the Dempster's rule (Equation 2.4) that is applicable for independent evidence.

According to Murphy, $n > 0.5$ means that the rule of combination should be *optimistic.* In this case, the new evidence is given more credit and has more weight during the integration. On the other hand, $n < 0.5$ means that the combination rule should be *pessimistic*, which results in a less weight of the new evidence during the integration. There are also values of n that produce neither optimistic nor pessimistic revision of belief. These values are referred to as being *neutral* and center around $n = 0.5$.

When $n < 1$, the Murphy's rule of combination is non-commutative. Unlike the Dempster's rule of combination which is commutative, the order of the presentation of evidence in the Murphy's rule entails different combination results.

The Murphy's rule of combination is suitable for the situation where the information is from the *same* source. However, it still does not solve the problem where information comes from *correlated* sources.

The D-S theory was first applied in AI community and spawned a wide interest [160]. It was introduced into software engineering to interpret inconsistencies in software requirement specifications [145]. It was also applied to pattern classification [65], when it was recognized that the Bayesian theory is not general and robust enough to cope with incomplete information.

2.1.3 Bayes' Theorem within the D-S Framework

Dempster's rule of combination permits a simple description of how the assimilation of new evidence should change our belief: our initial belief can be combined with the new evidence by their *orthogonal sum*. As discussed in §2.1.1, Bayes' rule of conditioning combines the new evidence with a Bayesian prior. As Shafer pointed out [136]: as long as the Bayesian prior is a valid one, the D-S scheme does not conflict with the Bayesian solution to the problem. In particular, he demonstrated that Bayes' rule of conditioning is a special case of the Demspter's rule of combination, where the new evidence is treated as certainty (which is essential to the Bayesian theory). In this case, the Bayes' rule of conditioning can be expressed by the orthogonal sum of the Bayesian prior and the new evidence according to the Dempster's rule of combination. The proof is as follows [136].

Suppose our initial belief is Bel_1 over the frame of discernment Θ. The effect of the new evidence Bel_2 on Θ is to establish a particular subset $B \subset \Theta$ with certainty. Then Bel_2 will give a degree of belief one to the proposition corresponding to B and to every proposition implied by it:

$$Bel_2(A) = \begin{cases} 1 & \text{if } B \subset A \\ 0 & \text{if } B \not\subset A \end{cases} \tag{2.11}$$

(The subset B is the only focal element of Bel_2, and its basic probability number is 1). Bel_1 and Bel_2 are combinable if and only if $Bel_1(\overline{B}) < 1$. If Bel_1 and Bel_2 are combinable, let $Bel_1(\cdot|B)$ denote $Bel_1 \oplus Bel_2$ (combination of Bel_1 and Bel_2), and let P_1 and $P_1(\cdot|B)$ denote the upper probability functions for Bel_1 and $Bel_1 \oplus Bel_2$, respectively. Then

$$Bel_1(A|B) = \frac{Bel_1(A \cup \overline{B}) - Bel_1(\overline{B})}{1 - Bel_1(\overline{B})}$$

and

$$P_1(A|B) = \frac{P_1(A \cap B)}{P_1(B)} \tag{2.12}$$

for all $A \subset \Theta$.

Note the similarity of Equation 2.1.3 to Bayes' rule of conditioning (Equation 2.1). They are essentially the same. Shafer demonstrated in greater details of how the Bayesian theory can be contained in Dempster-Shafer theory in [136].

2.2 Reasoning under Uncertainty

Belief networks are computational structures composed of clusters of nodes representing propositions interrelated by links signifying the independence relationships among the nodes. Some belief networks, such as Bayesian Belief Networks (BBNs), represent real world probabilistic knowledge with joint probability distributions and conditionals, while others, such as Dempster-Shafer Belief Networks (DSBNs), focus on the evidence propagation by belief-function measures of the nodes [94].

2.2.1 Bayesian Belief Networks

Bayesian Belief Networks (also known as Belief Networks, Causal Probabilistic Networks, Causal Nets, Graphical Probability Networks, Probabilistic Cause-Effect Models, and Probabilistic Influence Diagrams) have emerged as a promising solution for assessment reasoning under uncertainty. BBNs provide an intuitive graphical user interface (GUI) based on a sound mathematical theory of Bayesian probability. They have been proven useful in practical applications such as medical diagnosis, oil price forecasting and diagnosis of copier machine faults [119]. The most celebrated recent use is the Answer wizard in Microsofts Office 95 products for customer-tailored automated learning.

A BBN is a directed acyclic graph that represents probabilistic relationships among uncertain variables. The graph is made of nodes and arcs where the nodes represent uncertain variables and the arcs the causal/relevance relationships between the variables. Each node is associated with a node probability table (NPT). The NPT captures the conditional probabilities of a node given the value of its parent nodes. For nodes without parents, the NPTs are simply the marginal probabilities or prior distributions. There are several ways to determine the probabilities for the NPTs. We can accommodate both subjective probabilities elicited from domain experts and probabilities based on objective data. Each uncertain variable represents an event or a proposition. For instance, Figure 2.1 is a causal graph for reliability prediction and Table 2.1 is the NPT for the node "Reliability" [45]. In Figure 2.1, "# of latent faults " and "Operational usage" are the causal factors of "Reliability"; "Coder's performance" is the causal factor of "# of latent faults " and "Code complexity"; while

"Experience of staff", "Problem complexity" and "Use of IEC 1508" are the causal factors of "Coder's performance". "Reliability" can be predicted by the probability distributions of "Operational usage" and (number of) "faults" shown in Table 2.1, as well as conditional probabilities (which are not given in the paper). Each of these three nodes has three discrete values in Table 2.1: *low, med*, or *high*.

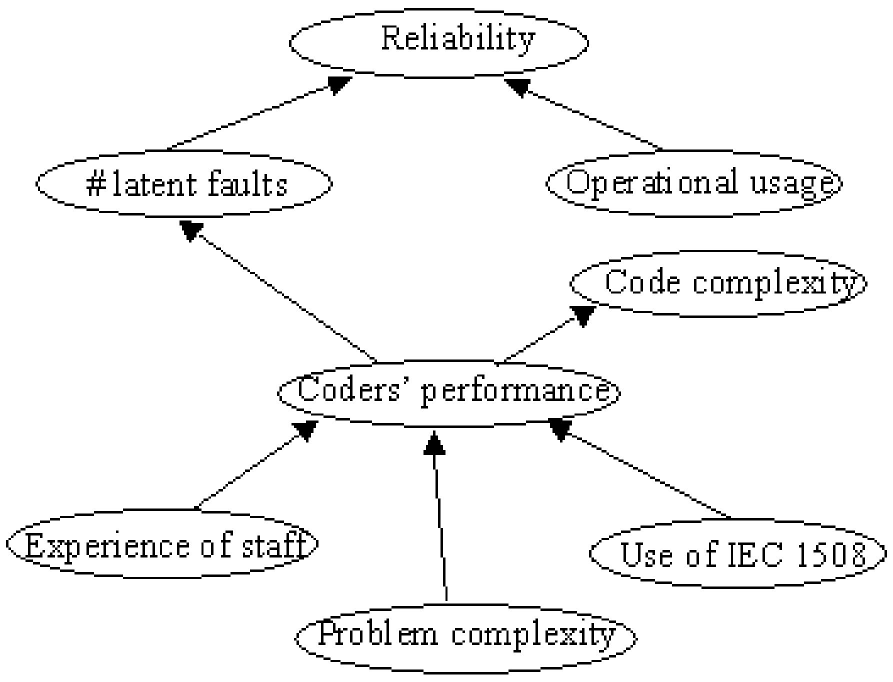


Figure 2.1: "Reliability prediction" BBN example [45]

The BBN directed acyclic graph with its NPTs specifies a joint marginal distribution of all events. When the actual state of a node is observed, the probabilities of all event states are updated by propagating the new evidence along the arcs in the graph. In this way, the probabilities change as the uncertainty and evidence change.

There have been many implementations of BBNs. The most notable one is the HUGIN tool [70] based on the award winning theoretical work of Lauritzen and Spiegelhalter [86].

The major benefit of Bayesian belief networks is that they explicitly predict the probability of unobserved events with some preexisting information of them. By using BBNs, an observer can justifiably hold that a certain statement of fact is true (subject probability); after observing new evidence, the observer can update the former belief ("prior probability") and obtain a "posterior probability". For example, in the reliability prediction BBN shown

Table 2.1: Node Probability Table (NPT) for the Node "Reliability" [45]

Operational usage		Low			Med			High		
faults		low	med	high	low	med	high	low	med	high
reliability	Low	0.10	0.20	0.33	0.20	0.33	0.50	0.20	0.33	0.70
	Med	0.20	0.30	0.33	0.30	0.33	0.30	0.30	0.33	0.20
	High	0.70	0.50	0.33	0.50	0.33	0.20	0.50	0.33	0.10

in Figure 2.1, we can assign our personal belief to the "operational usage" and the process nodes such as "experience of staff" and "problem complexity" as the prior distribution. After we obtain the evidence of the "operational usage" or the process information, we can update the probability distribution for each node in this BBN and obtain the prediction for reliability. However, the prior belief, which is required to fill in the relevant NPTs in a sensible way, is not always easy to obtain even from domain experts. When there is no prior belief available, the general approach is to assign a uniform prior.

Bayesian belief networks have been recognized as one of the most promising methodologies for prediction under uncertainty [2] [45] [37] [22] [108] [30] [31]. However, the important shortcoming in Bayesian theory we discussed above (the representation of complete ignorance) has not been taken seriously in BBN applications. Let's look at the reliability prediction example in Figure 2.1, which was once demonstrated in detail in [1]. In this example, the BBN was constructed to predict software reliability based on the product and process information. In the initial state, *experience of staff, problem complexity* and *operational usage* all have uniform priors. *Operational usage* has three discrete values, *low, med*, and *high*, with 33.33% each (see Figure 2.2), while the other two, *experience of staff* and *problem complexity* have two discrete values (*Yes/No* and *high/low*, respectively) with 50.00% each.

Then, they update the BBN with the evidence of best process: 100% experience of staff, 100% *low* problem complexity and use of IEC 1508. The updated reliability is then favoring *high* value compared to that of the initial state, which is of course a reasonable result. However, the updated belief of "operational usage" remains the same as the prior belief, as shown in Figure 2.2, since it is conditionally independent of the process information (this node is not connected with any other nodes in Figure 2.1 except for *reliability*).

In this case, the posterior belief of "operational usage" will be solely determined by the

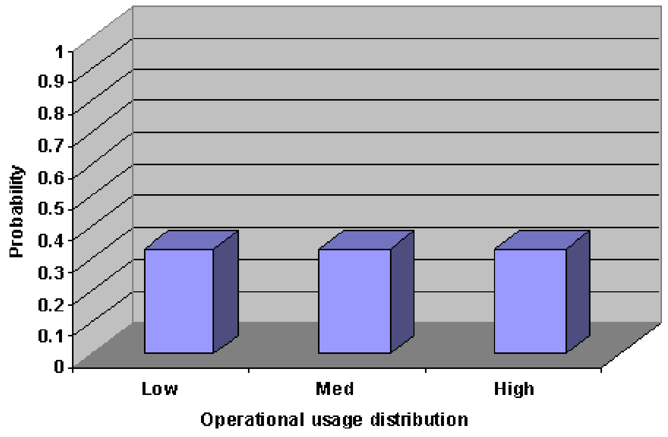


Figure 2.2: Operational usage distribution 1

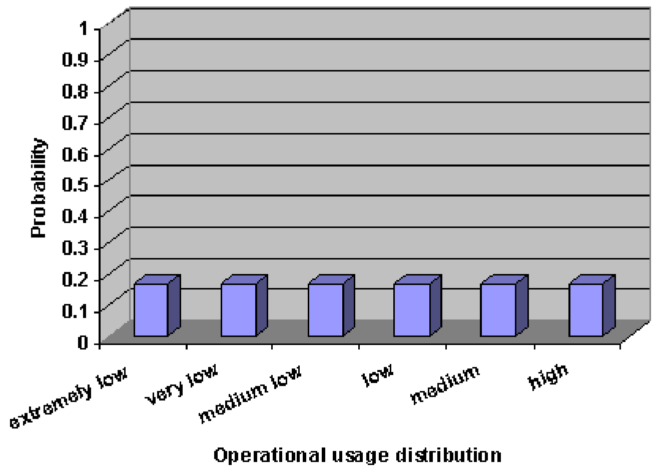


Figure 2.3: Operational usage distribution 2

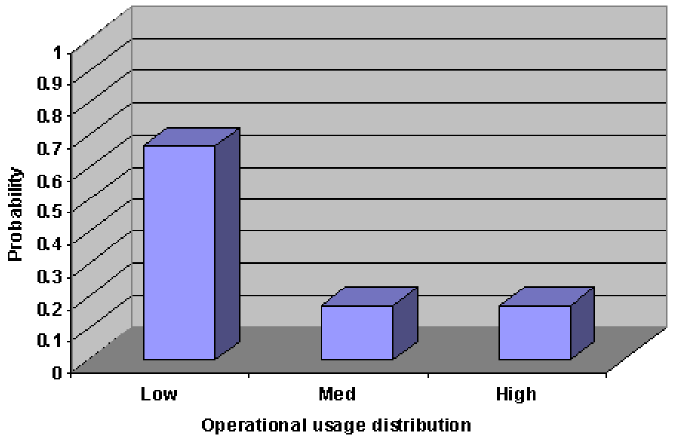


Figure 2.4: Operational usage distribution 3

prior assigned to this node. Therefore, the drawback with the uniform prior in Bayesian theory discussed in §2.1.1 is manifested here. Suppose we have six discrete values, instead of three, for operational usage: *extremely low, very low, medium low, low, medium*, and *high*. With uniform prior, it is now 16.67% probability assigned to each, and the distribution is shown in Figure 2.3. Notice that we can combine *extremely low, very low, medium low, low* as *low*, therefore, the combined distribution is 66.67% *low*, 16.67 *medium*, and 16.67% *high* shown in Figure 2.4. Figure 2.3 and Figure 2.4 are the same. However, they are inconsistent with the distribution in Figure 2.2. With this inconsistency between the distributions (as shown in Figure 2.2 and Figure 2.4) of operational usage, the predicted reliability will also be inconsistent.

We demonstrated in §2.1.1 that Bayes' theorem can be interpreted as:

$$Posterior \propto Prior \times Sample$$

indicating that given sample information, the posterior belief is directly impacted by the prior. If the prior is a problematic one, the resulted posterior belief is also problematic. As discussed above, with complete ignorance, the uniform prior assigned according to the Bayesian theory *is* problematic, because the distribution is subjective to human opinions. Consequently, the self-contradictory situation of uniform prior will also impact the updated posterior belief. While in the real-word applications where data is scarce and incomplete, it is common to profess ignorance of the prior about some parameter(s). In such situations, we could not rely on BBNs or any implementations directly operating on the Bayesian theory, and have to seek for a more general and robust framework, for instance Dempster-Shafer theory. One possible solution is the Dempster-Shafer belief networks, which will be described in the next section.

Besides the drawback rooted from the Bayesian theory, the Bayesian belief networks are difficult to build with missing data. Originally, the Bayesian networks were constructed based on human heuristics [60] and, thus, have been subject to human biases. Later, learning methods were designed to extract BBNs directly from application databases, replacing the insight gained by human domain experts [126] [134]. A common (mis)assumption made by most learning methods is that the database is complete, because exact Bayesian learning is intractable when data is missing. Unfortunately, real-world databases are rarely complete. Therefore, techniques to learn conditional probabilities from missing data were developed and explored: *Sequential updating* [157], the EM algorithm [35], and the Gibbs Sampling [143]. Solutions obtained from above methods are based on a common (mis)assumption that the unreported data is Missing at Random (MAR) so that the incomplete database is a representative sample of the complete one. Obviously, this assumption is unrealistic as the missing data may not always be uniformly distributed. In this case, *bound* and *collapse* (BC) method works better to derive the BBN structure from the incomplete database. Still, the results do not guarantee a correct BBN structure with missing data, as described in [127].

2.2.2 Dempster-Shafer Belief Networks

Dempster-Shafer belief networks form the basis for another inference methodology. The Dempster-Shafer networks can be induced automatically and dynamically from a data set. The structure of the D-S network does not represent causal relationship as in the Bayesian network. Instead, it represents implication relationship among the nodes. Unlike the Bayesian networks that need the complete knowledge of the real-world in order to build the correct causal model once and for all, the D-S networks can be constructed dynamically and efficiently based on available data. Therefore, the D-S network construction is more flexible than that of the Bayesian networks. The first induction algorithm for the Dempster-Shafer networks was proposed by Liu *et al.* [94] [96], based on binomial distribution. We developed an alternative induction algorithm [57], based on *prediction logic* [64], which is applicable for implication rules in general. This algorithm is presented in Chapter 3.

The induced D-S network is a directed graph. Each node represents an individual variable or hypothesis. Each arc in the graph signifies the existence of a direct implication (e.g., influence) rule between two adjacent nodes. The value of one variable is dependent on the values of all variables that influence it. When evidence from distinct sources is observed for certain nodes, it is combined by the Dempster-Shafer scheme [136]. Thus, beliefs for the corresponding nodes are updated and propagated to the neighboring nodes through the network. Dempster-Shafer networks are a promising methodology for prediction under uncertainty. However, they have not been applied in software engineering yet.

D-S Network Induction by Liu *et al.* [94]

Dempster-Shafer belief networks were first proposed by Liu *et al.* [94] [96]. The relationship between each pair of nodes in the D-S networks were induced based on binomial distribution. In Figure 2.5, each table is a contingency table for the corresponding implication relation. The shaded cells are the errors for the corresponding implication rule. For example, $A \wedge \neg B$ is the error cell for the implication rule $A \Rightarrow B$. $N_{A \wedge \neg B}$ represents the number of error occurrences. Ideally, if there is an implication rule $A \Rightarrow B$, we would never expect to find co-occurrences $A \wedge \neg B$. In reality, however, due to domain uncertainty or sampling errors, the error occurrences may not be *zero*. Therefore, they use a significance level α_c and a minimal conditional probability p_{min} to test whether the probability of the errors as in the contingency table (based on the lower tails of binomial distributions) is less than a threshold, α_c. The logically equivalent implication relations, for instance $A \Rightarrow B$ and $\neg B \Rightarrow \neg A$, are derived at the same time (called *modus ponens* and *modus tollens*, respectively). The detailed algorithm is shown in Figure 2.6. A numerical example of this algorithm is provided in [94].

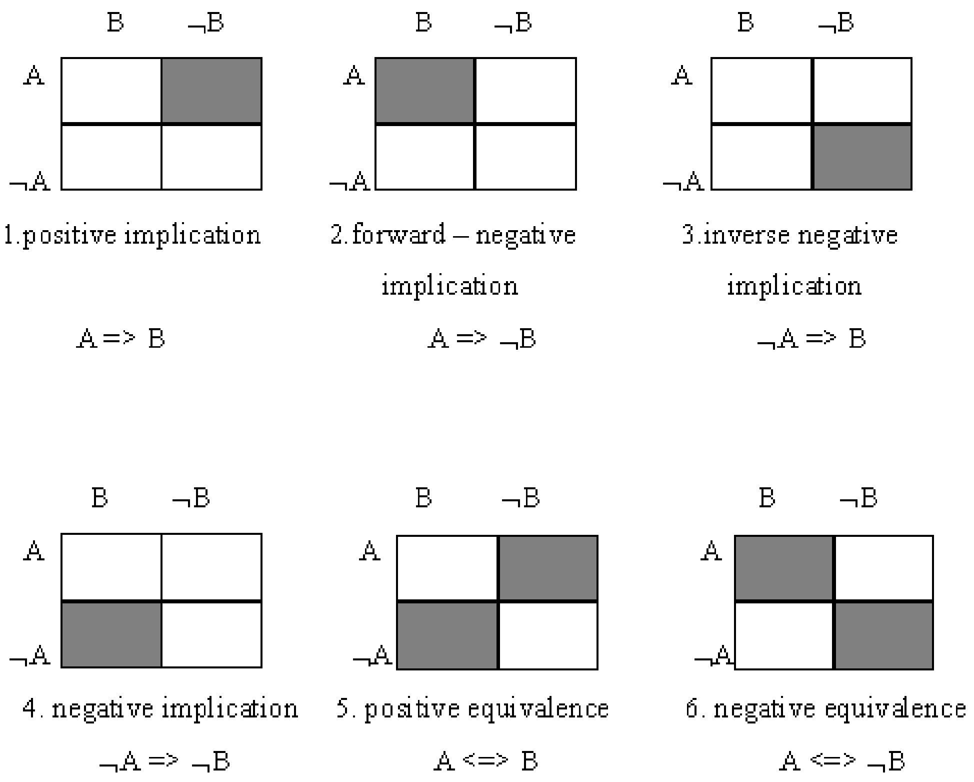


Figure 2.5: Six proposition types relating two dichotomous variables

Inference Propagation in D-S Networks

In D-S belief networks, the set of all possible outcomes of a node is called the frame of discernment, Θ, which must be exhaustive and disjoint. D-S theory allows a basic probability assignment to the subsets of a conclusion, which satisfies: $m : 2^\Theta \rightarrow [0, 1], m(\emptyset) = 0$, and $\sum_{\theta \subseteq \Theta} m(\theta) = 1$. A belief function, $Bel(B)$ is defined as the total belief committed to all subsets of B, i.e., $Bel(B) = \sum_{b \subseteq B} m(b)$.

Without loss of generality, suppose our frame of discernment Θ contains only two outcomes for proposition A, $\{a, \neg a\}$. Unlike the Bayesian theory, the uncertainty factor (igno-

```
The Implication Induction Algorithm by Liu et al. [94]
Begin
 Set a significance level $\alpha_c$ and a minimal conditional probability $p_{min}$
 for $node_i$, $i \in [0, n_{max} - 1]$ and $node_j$, $j \in [i + 1, n_{max}]$
   for all empirical case samples $N$
     Compute a contingency table

       $\mathbf{M}_{ij} = \begin{array}{|cc|} \hline N_{11} & N_{12} \\ N_{21} & N_{22} \\ \hline \end{array}$

     Where $N_{11}$, $N_{12}$, $N_{21}$, $N_{22}$ are the numbers of occurrences with respect
     to the following combinations:
       $N_{11} : node_i = TRUE \wedge node_j = TRUE$
       $N_{12} : node_i = TRUE \wedge node_j = FALSE$
       $N_{21} : node_i = FALSE \wedge node_j = TRUE$
       $N_{22} : node_i = FALSE \wedge node_j = FALSE$
   for each relation type $k$ test the following inequality

       $P(x \leq N_{error\_occurrences}) < \alpha_c$

   based on the lower tails of binomial distributions $Bin(N, P_{min})$ and $Bin(N', P_{min})$, where
   $N$ and $N'$ denote the occurrences of antecedent satisfactions in the two inferences using
   a type $k$ implication relation, i.e., in modus ponens and modus tollens, respectively.
   $\alpha_c$ is the alpha error of the conditional probability test.
  if the test succeeds, then return a type $k$ relation
End
```

Figure 2.6: The Induction Algorithm for Building D-S Networks by Liu et al. [94]

rance) is represented by $m(\Theta) = 1 - m(a) - m(\neg a)$. Furthermore, $Bel(a) + Bel(\neg a)$ does not have to be 1. This is a fundamental difference from Bayes' theorem. The direct implication is that in the D-S theory, when we assign belief $Bel(a)$ to proposition a, we do not automatically have belief $1 - Bel(a)$ assigned to its negation $\neg a$. The belief assigned to each proposition is only based on the evidence supporting it. When no evidence is available (complete ignorance), the belief assigned to each proposition is 0. In this case, the uncertainty factor $m(\Theta)$ equals 1.

The Dempster-Shafer scheme provides a means to combine evidence from distinct sources. By the Dempster's rule of combination [136], we can combine two independent sources of evidence by using Equation 2.4.

The Belief Revision Algorithm by Liu et al. [94]

{Initially, all the observed nodes (the predictors) are stored in a linked list, $link_{observ}$. *insert* and *get_next_node* are standard queuing functions. *Updated_belief* function computes the belief values based on Equation 2.4. $\Delta Bel()$ denotes the net change in beliefs before and after updating.}

```
Begin
for each observed node, $x_i$, in $link_{observ}$ do
  insert($x_i$, $queue$);
  while $queue$ is not empty, do
   $node \leftarrow$ get_next_node($queue$);
   if $node = TRUE$, then
    for each rule: $node \Rightarrow x_j$; $node \Rightarrow \neg x_j$;
        $x_j \Rightarrow \neg\ node$; $\neg x_j \Rightarrow \neg\ node$ do
      $Bel(x_j)$ = update_belief($node$, $x_j$);
      if $\Delta Bel(x_j) > threshold\ \theta$, then insert($x_j$, $queue$);
   if $node = FALSE$, then
    for each rule: $\neg node \Rightarrow x_j$; $\neg node \Rightarrow \neg x_j$;
        $x_j \Rightarrow node$; $\neg x_j \Rightarrow node$ do
      $Bel(x_j)$ = update_belief($node$, $x_j$);
      if $\Delta Bel(x_j) > threshold\ \theta$, then insert($x_j$, $queue$);
End
```

Figure 2.7: The Algorithm for Inference Propagation in D-S Networks [94]

Due to the node connectivity, the updated belief can be propagated throughout the network by the Belief Revision Algorithm [94] (see Figure 2.7). Belief revision starts from each observed node (the predictor), x_i, and propagates belief to its neighboring nodes based on the implication rules and the weight functions. During the propagation, it maintains a *queue* of next items from which the beliefs are to be propagated. The branching of a process stops whenever the path is terminated or the change in the belief value after updating is less than a threshold, θ (e.g. 0.1 percent). However, this algorithm does not make it clear when a node is considered as *true* or *false*.

Dempster-Shafer networks may not be singly connected. In order to prevent circular traversal of the graph, each node in the network is updated only once when an observation is made. Therefore, different order of the observations may result in different results, since different paths might be traversed [94] [96]. The complexity of the Belief Revision Algorithm is $\mathcal{O}(Nn^2)$, where N is the number of the implication rules, and n is the number of the nodes in the network. Since there is only one implication rule between each pair of nodes, N_{max} is

$\mathcal{O}(n^2)$. Hence, the complexity of the belief revision algorithm is $\mathcal{O}(n^4)$.

2.3 Modeling Software Quality from Product and Process Metrics

Software developers have a keen interest in software quality models, which automatically predict fault prone modules and subject them to more rigorous verification activities. Accurate predictions enable verification experts to concentrate their attention and resources at problem areas in the system under development.

2.3.1 Software Quality Models

Many modeling techniques have been developed and applied for software quality prediction. These include, logistic regression [10], discriminant analysis [78] [112], the discriminative power techniques [132], Optimized Set Reduction [17], artificial neural network [81], fuzzy classification [40], Bayesian Belief Networks (BBNs) [46] , genetic algorithms [9], and classification trees [53] [135] [148] [82]. The prediction accuracy of these models does not vary significantly. A tradeoff can be achieved by having a higher defect detection rate and compromising the overall prediction accuracy, or vise versa. Thus, a performance comparison of various models, if based on only one criterion (either the defect detection rate or the overall accuracy), may render the comparison only partially relevant. A model can be considered superior over its counterparts if it has both a higher defect detection rate, and a higher overall accuracy. About 65-75% of critical modules and non-fault prone modules were correctly predicted in [69] [78] [82]. The decision tree [135] correctly predicted 79.3% of high development effort or fault prone modules, while the trees generated from the best parameter combinations correctly identified 88.4% of those modules on the average. The discriminative power techniques correctly classified 75 of 81 fault free modules, and 21 of 31 faulty modules [132]. In one case study, within five common classification techniques: Pareto classification, classification trees, factor-based discriminant analysis, fuzzy classification, and neural network, fuzzy classification appears to yield best results with a defect detection rate of 86% [41]. Since most of these studies have been performed using different data sets, reflecting different software development environments and processes, the final judgement on "the best" fault-prone module prediction method is difficult to make. In addition, some papers do not report associated overall prediction accuracy, which makes objective comparisons even more difficult.

In addition, BBN models have been developed for predicting software safety [118], software dependability argumentation [119], defect prevention [117], decision-support and risk analysis [47] [46] [48], Fenton and Ohlsson used a BBN model [46] to explain their counter-intuitive empirical results: those modules which are the most fault-prone pre-release are among the least fault-prone post-release, while conversely, the modules which are most fault-prone post-release are among the least fault-prone pre-release [121]. Their observation shows that the commonly used fault density measure is misleading and flawed and challenges a commonly believed software engineering principle.

2.3.2 Metrics Used in Software Quality Prediction

Software size metrics predicted by component based method [72], object-oriented metrics [26], as well as many process and product metrics such as testability and communication metrics of a process [38], are candidate metrics to be included in software quality prediction.

In addition to some well known software metrics such as McCabe metrics [104] [105] and Halstead metrics [59], design-level metrics such as cohesion and coupling have also been used for software quality prediction [63] [122]. Bieman and Kang defined the design-level cohesion measures formally in [12], and used them to predict properties of implementations created from a given design. Their design-level cohesion (DLC) measure is similar to that used by Stevens *et al.* [144]. Later, Briand *et al.* introduced and compared various high-level design and measures for object-based software systems, which were derived based on an experimental goal, identifying fault-prone software parts, and several experimental hypotheses arising from the development of some high assurance systems [18]. Specifically, they defined a set of measures for cohesion and coupling and investigated the measures' relationship to fault-proneness.

Software ages have been a focus of software quality prediction. Graves *et al.* proposed their best model, the weighted time damp model, to predict fault potential by using a sum of contributions from all the changes to the model in its history [55]. The best generalized linear model they found uses numbers of changes to the module in the past together with a measure of the module's age. They found a model which can predict numbers of future faults from the numbers of past faults. Their most successful model measures the fault potential of a module as the sum of contributions from all of the times the module has been changed, with large, recent changes carrying the most weight.

2.4 Software Reliability Assessment

Software Reliability is formally defined as the conditional probability that a software failure which causes deviation from required output by more than specified tolerances, in a specified environment, does not occur during a specified exposure period, given that the software has not failed at time 0.

There are two general schemes for demonstrating that required software reliability have been achieved, formal verification and statistical testing. Different approaches to estimating software reliability by statistical testing will be reviewed in this section. As mentioned earlier, there are many indirect software reliability assessment approaches. However, there is no such an evidence combination framework that can integrate them for a more precise prediction of the software reliability.

2.4.1 Different Approaches to Software Reliability Estimation

There are following approaches to modeling software reliability:

- *Reliability growth models* [7] [91] [76] have been developed [98] and applied in software reliability prediction [74]. During the software development, the program implementation is repeatedly tested and repaired. A sequence of inter-failure times (usually measured in number of inputs) is recorded as the results. The goal is to construct a mathematical model to predict the reliability of the final program based on the observed inter-failure data.

- *Coverage-based models* establish the relationship between coverage metrics, such as branch coverage and block coverage, and defect coverage as well as software reliability.

- *Component-based models* targets reliability estimation of each component, and assess the system reliability based on the estimated component reliability. Component-based software reliability assessment models incorporate structural characteristics of software. They are especially valuable for reusable software such as Commercial-Off-The-Shelf (COTS)-based systems [54] and large software systems [138] [29].

The software reliability estimation models described above all assume that failure information available, i.e. inter-failure time (reliability growth models), failure probability of individual components (component-based models), as well as failure intensity and defect coverage (coverage-based models). However, there might be no failure observed during testing [8], for example, in some high assurance systems [21] [87] [11] [85] such as NASA projets and nuclear plant monitoring. In such cases, reliability growth models are incapable of overcoming the need for excessive amounts of testing [21]. Therefore, software reliability are modelled without failure information by following approaches:

- Estimating software reliability without failure history [13] [14] [25] [92] [102] [111].
- Predicting software reliability from process and product metrics.
- Estimating software reliability by *software testability*, which is the likelihood that defects can hide during software testing. Software testability could be estimated before testing takes place by static analysis of programs [149] [90] [80]. Therefore, it could help predict those programs that would reveal less defects during testing even if they contained defects. Voas and Miller believed that if software has high testability, a higher confidence could be assigned to the estimated reliability [149]. Bertolino and Strigini proposed an alternative exposition of testability measurement and pointed out that a higher testability does not always give higher confidence of estimated reliability.
- *Stopping rule* answers a question for all software testers: how much testing is enough to demonstrate that the required reliability has been achieved? Littlewood and Wright addressed the problem of specifying the numbers of test cases (or time periods) required for a test when the *previous* test has terminated as a result of failure, and proposed several novel Bayesian stopping rules [93]. Chávez provided a rigorous Bayesian framework to decide when to release a commercial software [24] based on the cost factors and the rate of bugs.

There are other software reliability models not belonging to the categories described above, such as *connectionist models* by using neural networks, training regimes, and data representation methods [75], *analogy models* and *regression models* [116] [137], as well as a model [20] to estimate the relationship between the predicted reliability and the true reliability. In this review, We will focus on related work in coverage-based models (§2.4.2), and predicting software reliability from process and product metrics (§2.4.3).

2.4.2 Coverage-Based Software Reliability Models

Code coverage metrics, such as branch coverage and block coverage, have been investigated for the relation to software reliability. Malaiya modelled the relation among software reliability, defect coverage, and code coverage in [99]. First, failure intensity λ is defined as:

$$\lambda = \frac{K}{T_L N} \tag{2.13}$$

where K is fault exposure ratio during the nth demand; T_L is the linear execution time; N is the number of defects remaining in the software. These parameters can be measured or estimated during testing. The determination of K will be further discussed later.

The number of remaining defects N can be obtained by:

$$N = \frac{N^0}{C^0} \tag{2.14}$$

where N^0 is the number of defects found by the test cases; C^0 is the defect coverage.

The relationship between the defect coverage C^0 and code coverage is:

$$C^0 = a_0 \ln[1 + a_1(e^{a_2 C_1} - 1)] \tag{2.15}$$

where C_1 is the code coverage achieved by the test cases; a_0, a_1, a_2 are coefficients. The coefficients can be estimated from data, or from previous projects [140].

Reliability $P_s(n)$ is calculated by:

$$P_s(n) = e^{-\lambda T(n)} \tag{2.16}$$

$T(n) = \tau * n$ (the duration of n demands)

$\tau = \frac{1}{\rho}$ (the average execution time per demand)

If we substitute $T(n)$ and λ (from Equation 2.13) in Equation 2.16, we obtain:

$$P_s(n) = e^{-\frac{K}{T_L} N \tau n} \tag{2.17}$$

Suppose the fault exposure ratio K is constant for all defects. Then the additional, unknown defects determined by Equation 2.14 have the same fault exposure ratio K as the known defects. From the testing results, when $n = 1$, N is the discovered defects, and $P_s = 1 - P_f$, from Equation 2.17 we can obtain K. P_f is the probability of failure per demand corresponding to the known defects, which can be estimated from the state machine in an application from [140]. A detailed case study will be described in §6.2.2.

Vouk [150] proposed a coverage growth model for non-operational testing with the following assumptions:

- Test cases are similar to sampling without replacement.
- The fault detection rate with respect to coverage is proportional to the coverage, also proportional to the number of residue faults.

- For each test case set there is a minimal C_{min} and a maximum coverage C_{max} ($0 \leq C_{min} \leq C \leq C_{max} \leq 1$).
- Fault correction is instantaneous and perfect.

From assumption it follows that the fault detection rate with respect to coverage is:

$$\frac{d\epsilon_d}{dC} = k\epsilon_r(C - C_{min}) \tag{2.18}$$

where ϵ_d is the effective number (or density) of detected faults.

Under another simplifying assumption that fault correction is instantaneous and perfect, the effective number of corrected faults, ϵ_c, is equal to the effective number of detected faults ϵ_d. So the effective number (or density) of residual faults is:

$$\epsilon_r = \epsilon_T - \epsilon_c \tag{2.19}$$

Where ϵ_T is the total number of faults in the program at coverage $C = 0$.

Therefore, the derived fault removal growth model is a variant of Raileigh distribution, i.e. a special case of Weibull distribution:

$$\epsilon_c = \epsilon_T[1 - e^{-\beta(C - C_{min})^2}] \tag{2.20}$$

where β is the coefficient.

Later, Rivers and Vouk [130] introduced *Testing Efficiency* function g_i to the coverage growth model. If $g_i = a$, the cumulative failure model is:

$$E_i = N - (N - E_{min})(\frac{G - C(i)}{G - C_{min}})^a \tag{2.21}$$

where N is the total number of defects; G is the total constructs; i the the number of test cases.

Piwowarski et al [124] developed a coverage growth model with the following assumptions:

- The program has G code constructs.
- Per test case, p constructs are sensitized (covered) on average.
- Test cases are sampling with replacement.

The relation between the coverage $C(i)$ and the test case i is similar to that in the Jelinski-Moranda model and the Geol-Okumoto model:

$$C(i) = 1 - e^{-\frac{p}{G}i} \tag{2.22}$$

The differences between the Piwowarski *et al.* model and the Rivers-Vouk model are:

1. The Rivers-Vouk model assumes that the test cases are not repeated as in the Piwowarski *et al.* model , i.e. (note the difference from Equation 2.22)

$$C(i) = \frac{p}{G}i \tag{2.23}$$

2. The Rivers-Vouk model is for systematic testing, which aim is to cover as many code constructs as possible.

3. The Piwowarski model is for operational testing, where test cases are repeated according to the distributions in the operational usage.

Therefore, a model is needed to map the gap between these two types of testing models. The Grottke model is such a model to generalize these two models [52]. The Grottke model is a vector Markov model with the following assumptions:

- At beginning, u_0 of G constructs are faulty.
- Per test case, p constructs are executed on average.
- A constant fraction r $(0 \leq r \leq 1)$ of those constructs may be tested again.
- The fault causes a failure with constant activation probability s $(0 < s \leq 1)$.
- The fault is removed instantaneously and perfectly.

The expected failure occurrences the the Grottke model is:

$$u(i) = u_0 \frac{s}{1 - r + rs} \times [1 - (1 - \frac{p}{G}(1 - r)i)^{\frac{1-r+rs}{1-r}}] \tag{2.24}$$

For $r = 1$, it is the Piwowarski model for operational testing:

$$u(i) = u_0[1 - e^{-\frac{p}{G}si}] \tag{2.25}$$

For $r = 0$, it is the Rivers-Vouk model for systematic testing with testing efficiency $g_i = 1$:

Table 2.2: Relationship between CMM Levels and Delivered Defects Mulivariate Approaches [73]

SEI CMM Levels	Defect Potentials	Removal Efficiency (%)	Delivered Defects (per KLOC)
1	5	85	0.75
2	4	89	0.44
3	3	91	0.27
4	2	93	0.14
5	1	95	0.05

$$u(i) = u_0 \cdot s \cdot \frac{p}{G} i = u_0 \cdot s \cdot C(i)$$

While many researchers strived to model a causal relationship between test coverage and defect coverage, Briand and Pfahl revealed that their study outcomes do not support a causal dependency between test coverage (including block, c-use, decision, and p-use coverage) and defect coverage [19]. Their observation contradicts the conclusion of other work described above. From their results, it is possible that we can't model defect coverage or software reliability from code coverage.

2.4.3 Predicting Software Reliability from Process and Product Metrics

Many researchers have been striving to incorporate product and process information to quantitatively predict software reliability. Hence, reliability estimation will not solely depend on the failure information collected during testing.

Attempts to predict software product reliability based on the quality of software process are not rare. Jones hypothesized the relationship between CMM levels and defect potentials as well as delivered defects [73] (See table 2.2). Fenton and Neil demonstrated that the use of a standard is likely either to deliver reliable and safe systems at an accepted cost or help predict reliability and safety accurately [44]. They examined a specific standard for safety critical systems (namely IEC 1508) and showed how it can be improved by applying their strategy.

Musa advocated to use an operational profile for software-reliability engineering and showed how to develop one step in step in [115]. He defined the operational profile as a quantitative characterization of how a system will be used that shows how to increase productivity and reliability and speed development by allocating development resources to function on the basis of use. Helander *et al.* presented modeling frameworks for distributing development effort among software components to facilitate cost-effective progress toward a system reliability goal [62]. Their approach, based on reliability allocation, uses the operational profiles to quantify the usage environment and uses a utilization matrix to link usage with system structure.

Schneidewind investigated an important facet of process capability, stability (how well a software operates without deterioration), as defined and evaluated by trend, change, and shape metrics, across release and within a release [133]. Their approach to integrating product and process measurement was illustrated to serve the dual purpose of using metrics to assess and predict reliability and risk and to evaluate process stability.

It is indicated that scenario based analysis according to ISO 9241-11 can be exploited for validation purposes [39] and requirement engineering [146]. Another standard, ISO/IEC 15504, becomes an emerging international standard on software process assessment. It defines numbers of software engineering processes and a scale for measuring their capability, one of which is software requirements analysis (SRA). EI Emam and Birk provided strong evidence of predictive validity for the SRA process capability measure used in ISO/IEC 15504 [42].

Fenton and Neil gave a critical review of previous software reliability prediction models and proposed Bayesian Belief Networks (BBNs) as the future research avenue [45]. According to them, defect counts cannot be used to predict reliability because it does not measure the software quality according to its operational usage. Despite the reported high correlations between design complexity and defects, the relationship is not entirely causal. The same problem exists when size and complexity metrics are used as predictors of defect. Interestingly, BBN enables managing the causal relationship between software product and process information and defects in the software. BBN was also applied to software reliability engineering self-assessment [36].

Cukic and Chakravarthy presented practical problems and challenges encountered in an effort to assess and quantify software reliability of a NASA's system [33]. They outlined a probabilistic Bayesian framework that allows accounting of rigorous verification and validation activities performed prior to system's deployment into the reliability assessment.

2.5 Summary

In this section, we gave a review on statistical models on evidence combination (information fusion) and probability reasoning under uncertainty. We also introduced software quality prediction and software reliability assessment. Having analyzed current techniques and open problems in these research areas, we developed two novel statistical methodologies based on Dempster-Shafer theory, and applied them to software quality and reliability prediction. Specifically, we developed a new induction algorithm for the Dempster-Shafer belief networks, which will be presented in Chapter 3. In addition, we propose a novel probability reasoning methodology based on D-S networks and apply it to software quality prediction in Chapter 4. The second proposed methodology focuses on information fusion. It extends the Murphy's rule of combination described in §2.1.2 to accommodate correlated information. This methodology was applied to software reliability prediction, which combines coverage-based reliability method (§2.4.2) and reliability methods based on product and process metrics (§2.4.3).

Chapter 3

A New Induction Algorithm for Dempster-Shafer Belief Networks

Dempster-Shafer belief network is an inference methodology for prediction under uncertainty. The network structure can be inducted automatically and dynamically from a relatively small data set. Based on upcoming evidence, the inducted belief network can be updated by the Dempster-Shafer scheme. The first induction algorithm for the Dempster-Shafer networks was proposed by Liu *et al.* [94] [96], based on binomial distribution. We developed an alternative induction algorithm [57], based on *prediction logic* [64], which is applicable for implication rules in general. We also develop a classification scheme by the Belief Revision Algorithm for updating the Dempster-Shafer networks, which is modified from [94].

3.1 A Novel Algorithm for D-S Belief Network Induction

We use *prediction logic* based on a contingency table of probabilities [64] to induce the D-S belief networks. There are six most important implication rules relating two dichotomous variables (see Figure 2.5). In Figure 2.5, each table is a contingency table. The shaded cells are the errors for the corresponding implication rule. For example, $A \wedge \neg B$ is the error cell for the implication rule $A \Rightarrow B$. $N_{A \wedge \neg B}$ represents the number of error occurrences. We use an optimality method to seek the most precise proposition that meets a required level of prediction success, modified from [64], to derive the implication relation between each pair of attributes in the data set. The implication induction algorithm is shown in Figure 3.1.

In the Implication Induction Algorithm, U_p is the *scope* of an implication rule, represent-

```
The Implication Induction Algorithm
Begin
 Set a significance level $\bigtriangledown_{min}$ and a minimal $U_{min}$
 for $node_i$, $i \in [0, n_{max} - 1]$ and $node_j$, $j \in [i + 1, n_{max}]$
 (Note: $n_{max}$ is the total number of attributes)
   for all empirical case samples $N$
     Compute a contingency table

                    $M_{ij} = \boxed{\begin{matrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{matrix}}$

   for each relation type $k$ find the solution to

                    $Max\ U_p$

          Subject to   $Max\ U_p \geq U_{min}$

                    $\bigtriangledown_p \geq \bigtriangledown_{min}$

 $\omega_{ij} = 1$ or $0$ (if $N_{ij}$ corresponds to an error cell, $\omega_{ij} = 1$; otherwise, $\omega_{ij} = 0$)

          $\bigtriangledown^{(b)} > \bigtriangledown^{(b')}$ if $\omega^{(b)} = 1$ and $\omega^{(b')} = 0$

  if the solution exists, then return a type $k$ relation
End
```

Figure 3.1: The Induction Algorithm for Building D-S Networks

ing the portion of the data set covered by the implication rule. $\bigtriangledown_p$ is the *precision* of an implication rule. An implication rule has high precision, if the number of error occurrences is only a small portion of the data covered by the implication rule. U_{min} and $\bigtriangledown_{min}$ are the minimum *scope* and *precision* required for the formation of an implication rule. They must be positive for a valid implication relation. Users can set thresholds for these two parameters according to their own requirements. They are also the tuning parameters of the D-S networks.

For a single error cell, if N_{ij} is the number of error occurrences, we have:

$$U_p = U_{ij} = \frac{N_{i.} * N_{.j}}{N^2} \tag{3.1}$$

$$\bigtriangledown_p = \bigtriangledown_{ij} = 1 - \frac{N_{ij}}{N * U_p} \tag{3.2}$$

For multiple error cells,

$$U_p = \sum_i \sum_j \omega_{ij} * U_{ij} \tag{3.3}$$

($\omega_{ij} = 1$ for error cells; otherwise, $\omega_{ij} = 0$)

$$\bigtriangledown_p = \sum_i \sum_j (\frac{\omega_{ij} U_{ij}}{U_p}) \bigtriangledown_{ij} \tag{3.4}$$

Our induction algorithm derives an implication rule if it has the maximum U_p value, and satisfies that its U_p and $\bigtriangledown_p$ are greater than the required minimum U_{min} and $\bigtriangledown_{min}$, respectively, and the $\bigtriangledown$ values of all non-error cells are greater than those of the error cells for the corresponding implication rule. The difference between our implication induction algorithm and that of Hildebrand *et al.* [64] is that we set minimum requirements for both *scope* (U_p) and *precision* ($\bigtriangledown_p$), instead of just *precision*. The complexity of the induction algorithm is $\mathcal{O}(Nn^2)$, where N is the sample size and n is the number of the attributes (i.e., nodes in the D-S networks).

For binary data sets, the logically equivalent relations are derived at the same time and carry different weight, represented by weight functions W_I and W_I'. For instance, the weight associated with $A \Rightarrow B$ can be represented as $W_I = \frac{N_{A \wedge B}}{N_{A \wedge B} + N_{A \wedge \neg B}}$, while the weight associated with its logical equivalence $\neg B \Rightarrow \neg A$ can be represented as $W_I' = \frac{N_{\neg A \wedge \neg B}}{N_{\neg A \wedge \neg B} + N_{A \wedge \neg B}}$. We use a quintuple to represent each implication rule I:

$$I \in \ddot{I}, I =< R, N_{ant}, N_{con}, W_I, W_I' >$$

where $\ddot{I}$ is the set of implication rules; W_I and W_I' are weight functions mapping the antecedent-consequent nodes, i.e., N_{ant} and N_{con}, and their negations to a real number between 0 and 1, respectively. That is,

$$W_I : N_{ant} \times N_{con} \rightarrow [0, 1]$$

$$W_I' : \neg N_{con} \times \neg N_{ant} \rightarrow [0, 1]$$

One way to define the weight functions is illustrated in the previous example. A numerical example of the induction algorithm is demonstrated in the following subsection.

3.1.1 A Numerical Example of the Induction Algorithm

Suppose from a data set, we can form a contingency table for attributes A and B as in Table 3.1. Each number in the table represents the number of occurrences of the corresponding event. For example, the number of the occurrences of A is *true* and B is *true* is 10, while the number of the occurrences of A is *true* and B is *false* is 790. We can calculate the U_p and $\bigtriangledown_p$ values for each proposition as follows:

Table 3.1: A Contingency Table

	B	$\overline{B}$
A	10	790
$\overline{A}$	190	10

1. For proposition $A \rightarrow B$,

$$U_{A\rightarrow B} = \frac{N_A N_{\overline{B}}}{N^2} = \frac{800 \times 800}{1000^2} = 0.64$$

$$\bigtriangledown_{A\rightarrow B} = 1 - \frac{N_{A\overline{B}}}{N_A N_{\overline{B}}} N$$

$$= 1 - \frac{790}{800 \times 800} \times 1000 = -0.234$$

2. Similarly, for proposition $A \rightarrow \overline{B}$,

$$U_{A\rightarrow \overline{B}} = 0.16$$

$$\bigtriangledown_{A\rightarrow \overline{B}} = 0.938$$

3. For proposition $\overline{A} \rightarrow B$,

$$U_{\overline{A}\rightarrow B} = 0.16$$

$$\bigtriangledown_{\overline{A}\rightarrow B} = 0.938$$

4. For proposition $\overline{A} \rightarrow \overline{B}$,

$$U_{\overline{A}\rightarrow\overline{B}} = 0.04$$

$$\bigtriangledown_{\overline{A}\rightarrow\overline{B}} = -3.75$$

5. For proposition $A \rightleftharpoons B$,

$$U_{A\rightleftharpoons B} = \frac{N_A N_{\overline{B}} + N_{\overline{A}} N_B}{N^2} = 0.68$$

$$\bigtriangledown_{A\rightleftharpoons B} = 1 - \frac{N_{A\overline{B}} + N_{\overline{A}B}}{N_A N_{\overline{B}} + N_{\overline{A}} N_B} N = -0.442$$

6. For proposition $A \rightleftharpoons \overline{B}$,

$$U_{A\rightleftharpoons\overline{B}} = \frac{N_A N_B + N_{\overline{A}} N_{\overline{B}}}{N^2} = 0.32$$

$$\bigtriangledown_{A\rightleftharpoons\overline{B}} = 1 - \frac{N_{AB} + N_{\overline{AB}}}{N_A N_B + N_{\overline{A}} N_{\overline{B}}} N = 0.938$$

For a valid implication rule, U_{min} and $\bigtriangledown_{min}$ must be positive. If we set $U_{min} = 0.25$ and $\bigtriangledown_{min} = 0.6$, the derived implication rule is $A \rightleftharpoons \overline{B}$. It not only satisfies following criteria: $U_{A\rightleftharpoons\overline{B}} > U_{min}$, $\bigtriangledown_{A\rightleftharpoons\overline{B}} > \bigtriangledown_{min}$, as well as $\bigtriangledown^{(b)} > \bigtriangledown^{(b')}$ if $\bigtriangledown^{(b)}$ represents the precision measure for every error cell contained in the error set and $\bigtriangledown^{(b')}$ represents the precision measure for every cell not in the error set (In this case, $\bigtriangledown_{A\rightarrow\overline{B}} > \bigtriangledown_{A\rightarrow B}$, $\bigtriangledown_{A\rightarrow\overline{B}} > \bigtriangledown_{\overline{A}\rightarrow\overline{B}}$, $\bigtriangledown_{\overline{A}\rightarrow B} > \bigtriangledown_{A\rightarrow B}$, and $\bigtriangledown_{\overline{A}\rightarrow B} > \bigtriangledown_{\overline{A}\rightarrow\overline{B}}$), but also has the maximum U_p value among all those implication rules that satisfy above criteria.

For proposition types $1-4$, a quintuple is used to represent the corresponding implication rule. For proposition types 5 and 6, two quintuples are used to represent the corresponding implication rule. In this example, the induced implication relation is $A \rightleftharpoons \overline{B}$, which is the double implication rule of $A \rightarrow \overline{B}$ and $\overline{B} \rightarrow A$. The weight function associated with $A \rightarrow \overline{B}$ is $W_I = \frac{N_{A\wedge\overline{B}}}{N_A} = \frac{790}{800} = 0.988$, while the weight associated with its logical equivalence $B \rightarrow \overline{A}$ is $W_I' = \frac{N_{\overline{A}\wedge B}}{N_B} = \frac{190}{200} = 0.95$. Similarly, the weight associated with $\overline{B} \rightarrow A$ and its logical equivalence is 0.95 and 0.988, respectively. Hence, I_1 and I_2 are used to represent the implication rule derived in this numerical example:

$$I_1 =< 2, A, B, 0.988, 0.95 >$$
$$I_2 =< 3, A, B, 0.95, 0.988 >$$

3.2 Modifications of the Belief Revision Algorithm

As introduced in §2.2.2, due to the node connectivity, the updated belief can be propagated throughout the network by the Belief Revision Algorithm [94] (see Figure 2.7). Belief revision starts from each observed node (the predictor), and propagates belief to its neighboring nodes based on the implication rules and the weight functions. The branching of an inference process stops whenever the path is terminated or the change in the belief value after updating is less than a threshold, θ (e.g. 0.1 percent). However, this algorithm does not make it clear when a node is considered as *true* or *false*.

We modified the belief revision algorithm from Liu *et al.* [94] by adding the domain specific factor τ to decide if a proposition is *true* or $false$. For each node's frame of discernment, for instance $\{a, \neg a\}$, if $Bel(a) > \tau$ ($\tau > 0.5$), it is considered *true*; if $Bel(\neg a) > \tau$, it is considered $false$. Once the belief updating process is finished, the node to be predicted, y, is classified as *true* if $Bel(y) > \tau$, $false$ if $Bel(\neg y) > \tau$, or *no prediction* if neither of the criteria is satisfied. In this way, the D-S networks can also be applied as classifiers. A numerical example of applying the modified Belief Revision Algorithm is presented below.

3.2.1 A Numerical Example of the Modified Belief Revision Algorithm

At the beginning of the program, each belief function of the nodes is initialized as $Bel(A) = 0.0001$, which is negligible during the computation.

Suppose a D-S network has only two nodes, x_1 and x_2, connected by the implication rule $x_1 \rightarrow x_2$, and the quintuple for the implication rule is $I =< 1, x_1, x_2, 0.95, 0.9 >$. For each node, proposition a is $x = 1$, and proposition $\neg a$ is $x = 0$. When $x_1 = 1$ is observed from the data set, the initial beliefs are assigned as $Bel_{x_1}(a) = 0.9$ and $Bel_{x_1}(\neg a) = 0.1$. The belief revision algorithm works as follows.

First, these two belief functions are combined for x_1 by the Dempster's rule of combination as Equation 2.4:

$$Bel_{x_1}(a)' = \frac{Bel_{x_1}(a)(1 - Bel_{x_1}(\neg a))}{1 - Bel_{x_1}(a)Bel_{x_1}(\neg a)} = 0.89$$

$$Bel_{x_1}(\neg a)' = \frac{Bel_{x_1}(\neg a)(1 - Bel_{x_1}(a))}{1 - Bel_{x_1}(a)Bel_{x_1}(\neg a)} = 0.01$$

If the Domain Specific Factor $\tau = 0.65$, since $Bel(a) > \tau$, proposition a is considered *true* for x_1. Therefore, the belief is propagated through the implication rule I. Since $W_I = P_{x_2|x_1}$, the belief propagated to x_2 is:

$$Bel_{x_2}(a) = Bel_{x_1}(a)'W_I = 0.89 \times 0.95 = 0.85$$

For this simple D-S network, the belief propagation process stops here. Now $x_2 = 1$ is classified as *true*, since $Bel_{x_2}(a) > \tau$.

Chapter 4

Predicting Fault Prone Modules in Software Engineering

Automated detection of fault prone modules during software development process is an important prerequisite for successful verification of large systems. Models, built from metrics collected in past projects or releases, are used to identify fault prone module (a module is equivalent to a C function) candidates in the current project/release and subject them to more rigorous verification activities. Successful models are characterized by high prediction accuracy, thus allowing verification experts to concentrate their attention and resources at problem areas in the system under development.

This chapter describes an original methodology for predicting fault prone modules. This methodology is based on Dempster-Shafer (D-S) belief networks. Our approach has the following characteristics: (1) the methodology is general and not restricted to particular metrics or research objectives. (2) It is objective, highly automatic and computationally efficient. It consists of three major parts: First, building the Dempster-Shafer network from an existing data set by the induction algorithm; Second, selecting the predictors for the Dempster-Shafer network; Third, feeding the predictors describing the modules of the current project into the inducted Dempster-Shafer network and identifying fault prone modules. We applied this methodology in two case studies based on NASA data sets. The prediction accuracy of the proposed methodology is higher than that achieved by logistic regression, discriminant analysis, random forests as well as the algorithms in two machine learning software packages WEKA [153] and See5 [6] for the same data sets. The difference in the performance of the proposed methodology over other methods is statistically significant.

4.1 Introduction to Software Quality Prediction

Software developers have a keen interest in software quality models, which automatically predict fault prone modules and subject them to more rigorous verification activities. Accurate predictions enable verification experts to concentrate their attention and resources at problem areas of the system in early software life cycle.

As reviewed in §2.3.1, many modeling techniques have been developed and applied for software quality prediction. Since most of these studies have been performed using different data sets, reflecting different software development environments and processes, the final judgement on "the best" fault-prone module prediction method is difficult to make. In addition, some papers do not report associated overall prediction accuracy, which makes objective comparisons even more difficult.

In this chapter, we introduce a novel software quality prediction methodology based on the Dempster-Shafer (D-S) belief networks. We compare the proposed methodology with many existing approaches using the same data sets. Our approach to predicting fault prone modules has the following characteristics. (1) The methodology is general and not restricted to particular metrics or research objectives. (2) It is objective and highly automated. Each step of the methodology is performed by custom made computer programs or commercial software. Little human interaction is involved during the experimental procedure. (3) It is computationally efficient. The algorithms, D-S network induction and belief updating, are polynomial in time complexity. The methodology consists of three major parts. First, a Dempster-Shafer network is built by the induction algorithm. Next, predictors are chosen after performing feature selection. Finally, the predictors are fed to the inducted Dempster-Shafer network to make a prediction. The prediction accuracy of the proposed methodology is higher as compared to logistic regression, discriminant analysis, or the algorithms employed in two data ming software packages, WEKA and See5 for the same data sets obtained from NASA. In addition, the proposed methodology entails lower effort for software inspection as compared to another defect module detector, ROCKY [110]. The difference in the performance of the proposed methodology over other methods is statistically significant.

This chapter illustrates a set of experiments performed by the D-S networks with case-based reasoning on data sets obtained from NASA. The information from the previous project was used in making a prediction for the current project. The methodology is useful for large scale projects or projects with multiple releases, since it is impossible to collect the information on all modules early in the software life cycle. The basic hypothesis is that a module currently under development is fault prone, if a module with the similar product or process metrics in an earlier project (or release) was fault prone [79].

The remainder of this chapter is organized as follows. Section 4.2 introduces the proposed methodology. Section 4.3 introduces the data sets used in the case studies, and the measurement parameters used in the experiments. Section 4.4 outlines the major steps of the experiments. Section 4.5 and 4.6 presents two case studies. Section 4.7 provides a comparison of the results obtained from the proposed methodology over related work, i.e. logistic regression, discriminant analysis, random forests, two machine learning software packages, See5 [6] and WEKA [153], and NASA's ROCKY toolset [110] for the same data sets. Finally, Section 7 summarizes the unique aspects and advantages of the proposed methodology.

4.2 Methodology

We developed an inference methodology based on Dempster-Shafer belief networks. As illustrated in Chapter 3, the Implication Induction Algorithm is first applied to a data set to build the D-S network. The inducted D-S network is then updated based on upcoming evidence by the Belief Revision Algorithm. Finally, when the inference propagation stops, we can make a prediction based on the final values of the nodes in the D-S network. The mechanism of Dempster-Shafer Networks is depicted in Figure 4.1.

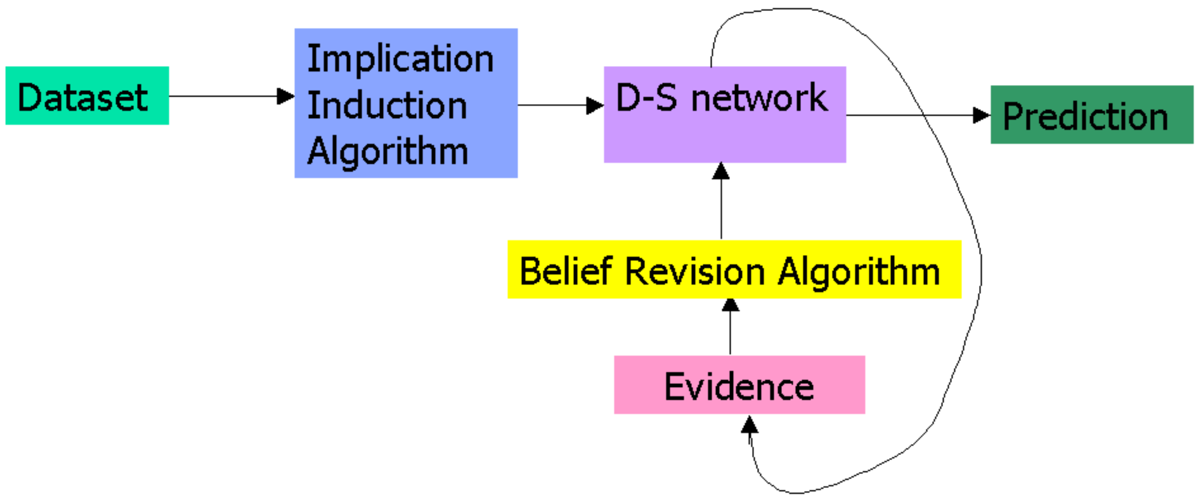


Figure 4.1: The mechanism of Dempster-Shafer Belief Networks

Specifically, our methodology contains three steps:

- building the D-S belief network by the Implication Induction algorithm;

- selecting predictors by feature selection. As Hall and Holmes suggest [58], including irrelevant, redundant or noisy features can lead to data mining algorithms with poor predictive performance. Attribute subset selection is the process of identifying and removing as much of the irrelevant and redundant information as possible. Attribute selection techniques can be categorized into two areas: those which rank individual attributes such as the attribute selection algorithms in WEKA [153], and those which rank *subsets* of attributes for instance the logistic procedure in SAS [5];

- feeding the selected predictors into the inducted D-S network and make a prediction. Now the selected predictors are treated as upcoming evidence for the Belief Revision Algorithm, which is used to update the inducted D-S network. After all the predictors are incorporated into the D-S network, we can make the prediction based on the final values of the nodes in the network.

Dempster-Shafer belief networks have not been applied in software engineering since its introduction by Liu *et al.* In this study, we apply it to predicting fault prone modules in software engineering.

4.3 Data sets and Measurement Parameters

4.3.1 Projects KC2 and JM1

The data sets used in the case studies are two C++ NASA projects, referred to as KC2 and JM1 [109]. KC2 contains over $3,000$ modules (a module is equivalent to a C function). 520 modules are of research interest as they were built by NASA software developers. The remaining modules are COTS software. Of these 520 modules, 106 were found to have between 1 and 13 faults, while the remaining 414 were fault free. KC2 modules have the average size of 37 lines of code (LOC), while the largest module has $1,275$ LOC. JM1 is a much larger software with $10,883$ modules, of which $2,105$ modules have between 1 and 26 faults. The remaining $8,778$ modules are fault free. JM1 modules have the average size of 42 lines of code (LOC), while the largest module has $3,442$ LOC.

Each data set contains 21 software metrics, including McCabe [103] [104] [105], Halstead [59], Line Count, and Branch Count. Metric descriptions are listed in Table 4.1. KC2 data set contains additional three attributes: *Error Rate* (number of defects in the module), *Defect* (whether or not the module has any defects), and *Defect Density* (number of defects per LOC). JM1 contains only two of these attributes: *Error Rate* and *Defect*.

Table 4.1: Metric Descriptions of KC2 and JM1

Metric Type	Metric	Definition
McCabe	v(G)	Cyclomatic Complexity
	ev(G)	Essential Complexity
	iv(G)	Design Complexity
	LOC	Lines of Code
Derived Halstead	N	Length
	V	Volume
	L	Level
	D	Difficulty
	I	Intelligent Count
	E	Effort
	B	Effort Estimate
	T	Programming Time
Line Count	LOCode	Lines of Code
	LOComment	Lines of Comment
	LOBlank	Lines of Blank
	LOCodeAndComment	Lines of Code and Comment
Basic Halstead	UniqOp	Unique Operators
	UniqOpnd	Unique Operands
	TotalOp	Total Operators
	TotalOpnd	Total Operands
Branch	BranchCount	Total Branch Count

4.3.2 Measurement Parameters

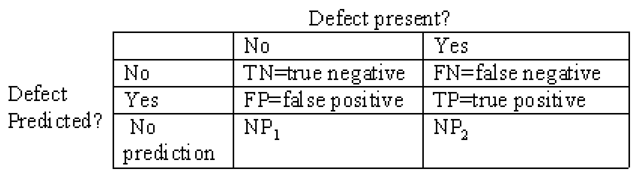
In this study we are interested in predicting whether or not the module contains any defects, instead of how many defects it contains. Software metrics serve as predictors. The predicted variable is *Defect*. Figure 4.2 presents a defect prediction sheet.

Specificity is used to define the rate of the defect module detection. In the literature, it is also referred to as *Probability of Detection (PD)* [110]:

$$Specificity(PD) = \frac{TP}{FN + TP + NP_2} \tag{4.1}$$

Similarly, *Sensitivity* is defined as the portion of the correct classification of non-fault prone modules:

$$Sensitivity = \frac{TN}{TN + FP + NP_1} \tag{4.2}$$



		Defect present?	
		No	Yes
Defect Predicted?	No	TN=true negative	FN=false negative
	Yes	FP=false positive	TP=true positive
	No prediction	NP_1	NP_2

Figure 4.2: A defect prediction sheet

The overall prediction accuracy is measured by *Acc*:

$$Acc = \frac{TN + TP}{TN + FN + FP + TP + NP_1 + NP_2} \tag{4.3}$$

Another parameter is *Probability of False alarm (PF)*. It represents the ratio of non-fault prone modules predicted as fault prone modules:

$$PF = \frac{FP}{TN + FP + NP_1} \tag{4.4}$$

Effort is defined to represent the resources required for the inspection of faulty modules [110]. In our case, we define *Effort* as the percentage of the lines of code of all the modules predicted as *fault prone* or *no prediction*:

$$Effort = \frac{LOC_{FP} + LOC_{TP} + LOC_{NP}}{LOC_{TN} + LOC_{FN} + LOC_{FP} + LOC_{TP} + LOC_{NP}}, \tag{4.5}$$

where $LOC_{NP} = LOC_{NP_1} + LOC_{NP_2}$.

4.4 Description of the Experiments

4.4.1 Primary Data Treatment

KC2 and JM1 are numerical continuous data sets. Since Dempster-Shafer networks deal with discrete data sets, we discretized the continuous data sets into binary ones by AWK programs. We partition the data sets using the mean value or the median of each attribute. If the data value is greater than the mean (median) of the corresponding attribute, it is assigned 1; otherwise, it is 0. The predicted variable, *Defect*, is 1 if the module contains fault(s), or 0 if it is fault free.

4.4.2 Selecting the Predictors

There are 21 predictors in the project data. Some of them are highly correlated. Therefore, using all of them as predictors may not result in optimal prediction. We select predictors by performing feature selection. As discussed in §4.2, attribute selection techniques can be categorized into two areas: those which rank individual attributes, and those which rank *subsets* of attributes. We apply both kinds of attribute selection algorithm in our study.

WEKA provides many attribute selection algorithms which rank individual attributes. There are six standard methods used in the Hall and Holmes study [58]. We applied them to our discretized data sets and selected the top ranked attributes as predictors.

- *Information Gain Attribute Ranking*: It gives the average merit and average rank of each attribute. The attributes with highest merit are selected as predictors.
- *Relief*: This algorithm also gives the average merit and average rank of each attribute. The attributes with highest merit are selected as predictors.
- *CFS* (Correlation-based Feature Selection): It lists how many times each attribute is selected during 10-fold cross validation. The attributes appearing in 10 folds are chosen as predictors.
- *Consistency-based Subset Evaluation*: It also lists how many times each attribute is selected during 10-fold cross validation. Different from *CFS*, it selects more attributes than required. For discretized JM1 data sets, for example, every attribute except two appears in 4 folds during 10-fold cross validation. We select the attributes with most frequent appearance during 10-fold cross validation as predictors.
- *PCA* (Principle Component Analysis): It returns eigenvectors of weighted attributes. The selected attributes by this algorithm are used as predictors.
- *Wrapper*: This algorithm does not produce any results for our data sets, due to the enormous computing resources needed by the algorithm.

The five WEKA attribute selection algorithms can produce ranked individual attributes within several seconds for KC2 data set, and within a couple of minutes for JM1 data set. The selected predictors constitute the input to the Dempster-Shafer networks.

The logistic regression procedure in SAS (a commercial statistical software) [5] is an attribute selection algorithm that selects *subsets* of attributes. We also used this method to select predictors. The transformed discrete data sets are input files to the logistic regression procedure.

The logistic regression procedure in SAS generated 20 score tables of the candidate predictors for each input file in a second. It ranks the Chi-Square scores for each combination of the predictors. The number of the predictors in the score tables increases from 1 to 20. Each score table contains best combinations with the same number of predictors. For example, score table 1 contains best single predictors; score table 2 contains best combinations of 2 predictors, etc. The top 5 combinations from each score table were chosen as the candidate predictor sets to the Dempster-Shafer networks. The predictors are input into the Dempster-Shafer networks incrementally. If increasing the number of predictors results in reduced prediction accuracy, we stop trying subsets with more predictors. Therefore, the best subsets of predictors are selected.

4.4.3 Empirical Validation

We used 10-fold cross-validation to evaluate our prediction of fault prone modules for KC2 project. The data set was randomly partitioned into 10 folds of equal size. The D-S network was trained and tested 10 times. Each time 9 folds are picked to build the Dempster-Shafer network by the induction algorithm, while the remaining fold is validated by the belief revision algorithm. The experiment is complete when all the 10 folds are validated. The cross-validation was run for at least 60 times in each experiment. The result with the least variance was chosen as the final result. In this way, we are confident that the accuracy estimation has low bias and low variance [83]. For JM1 project, we randomly picked some data for learning, and used the remaining data for validation. Each experiment was performed 10 times. The average of 10 consecutive runs was used as the final result. The details are explained in the next section.

During the validation, the predictors picked in Section IV.B were used by the inducted Dempster-Shafer networks as the observations. Since the order of the observations matters, different sequences of the predictors were tried, and the best sequences were recorded. In addition to the sequence of the predictors, there are five other tuning parameters in the D-S networks: U_{min} and $\bigtriangledown_{min}$ in the induction algorithm, the domain specific parameter τ in the belief revision algorithm, and the initial beliefs assigned to the observed nodes based on their values in the data set. Different tuning of these parameters results in different prediction. According to different real-world requirements, for example, to achieve maximal *Specificity (PD)* and *Accuracy (Acc)*, or to use minimal *Effort* while achieving maximal *PD*, the system can be tuned to output the optimal results that meet these criteria. The optimal results for each set of criteria are selected by dominant rule, which first sorts the results in order and then discards the results dominated by others. For example, if the measurement parameters of interest are *Acc* and *PD*, we use $r =< Acc, PD >$ to represent each result. Suppose we have $r_1 =< 0.6, 0.7 >$, $r_2 =< 0.5, 0.8 >$, $r_3 =< 0.6, 0.6 >$. Since r_3 is dominated by r_1, it is discarded. Therefore, the optimal results are r_1 and r_2.

4.5 Case Study 1

The original KC2 data set was transformed into two binary data sets, KC2a and KC2b. KC2a was generated by partitioning with the mean value of each attribute. If the data value is greater than the mean value of the corresponding attribute, it is assigned value 1; otherwise, it is 0. KC2b was obtained by stratifying with the median of each attribute. If the data value is greater than the median of the corresponding attribute, it is 1; otherwise, it is 0. For both data sets, the predicted variable, *Defect*, is 1 if the module contains any fault(s), or 0 if it is fault free.

Each transformed data set served as input to the WEKA attribute selectors and the LOGISTIC procedure in SAS, which generated candidate predictors. We tuned the system to meet two sets of real-world requirements: one is to maximize *Accuracy (Acc)* and *Specificity (PD)*; the other is to minimize *Effort* and maximize *PD*.

The prediction results tuned for maximal *Acc* and *Specificity* are depicted in Figure 4.3. Experiments 1–10 are the results from KC2a and 11–13 are the results from KC2b. Figure 4.3 indicates that different data treatments give different ranges of prediction accuracy. Data partitioned by the mean values have higher overall accuracy (*Acc*), while data partitioned by the median tend to give higher rate of defect detection (*Specificity*), up to 91.5%.

The prediction tuned for minimal *Effort* and maximal *PD* are depicted in Figure 4.4 for KC2a, and in Figure 4.5 for KC2b. Both figures demonstrate that the required percentage of code that is recommended for inspection is below the percentage of detected fault-prone modules. The results from KC2b indicates that we could, for example, detect 91.5% of defects by reading 71.8% of the source code. On the average, *PD* is higher than *Effort* by 18.2% on KC2b.

From the record of the best sequences of the predictors, we found that, generally, using 2 to 4 predictors results in optimal prediction for KC2 project, and the best combinations generally come from the top three candidates from the score tables produced by the logistic procedure. The *CFS* algorithm in WEKA generates the same predictors as the logistic procedure for KC2b. The predictors selected by the rest of the WEKA attribute selection

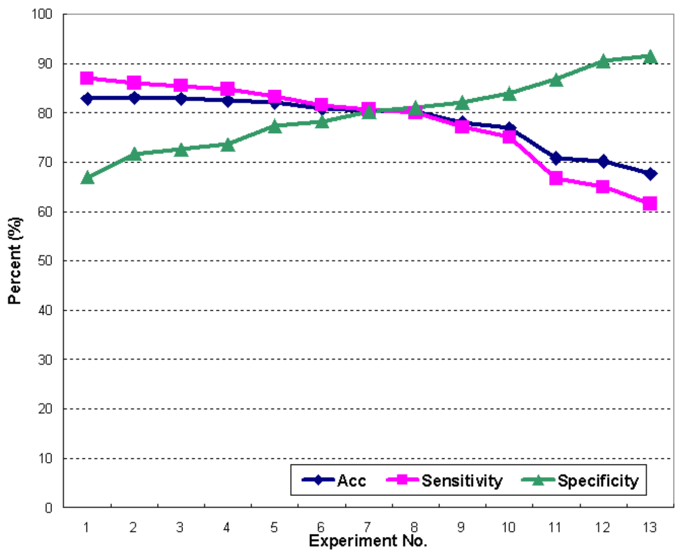


Figure 4.3: Prediction of fault-prone modules by the D-S networks on KC2

algorithms are unable to give optimal performance. The tuning parameters for Figure 4.3 are listed in Table 4.2. From Table 4.2, we can observe that E(Effort), V(Volume), and *LOComment* (Lines of comment) are good defect predictors for KC2, which are selected most frequently in the tuning process.

4.6 Case Study 2

JM1 is a very large project containing 10,883 modules. The practical way to make early prediction of defect modules in a large project is to learn from the previous project and/or a small part of the current one. Therefore, in some of our experiments, we used KC2 project as the library for case-based reasoning rules and devised the experiments for JM1 as follows.

First, the JM1 data set was transformed into two binary data sets, JM1a and JM1b. JM1a was generated by partitioning with the mean value of the corresponding attribute from KC2a. If the data value is greater than the KC2a mean value, it is assigned value 1; otherwise, it is 0. JM1b was obtained by stratifying with the median of the corresponding attribute from KC2b. If the data value is greater than the KC2b median, it is 1; otherwise, it is 0. For both data sets, the predicted variable, *Defect*, is 1 if the module contains any fault(s), or 0 if it is fault free.

JM1a and JM1b were input into the WEKA attribute selectors as well as the LOGIS-

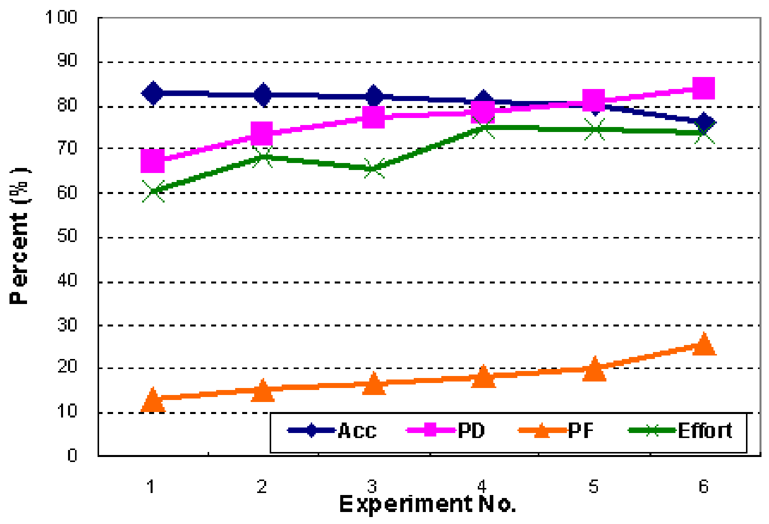


Figure 4.4: Prediction of fault-prone modules by the D-S networks on KC2a

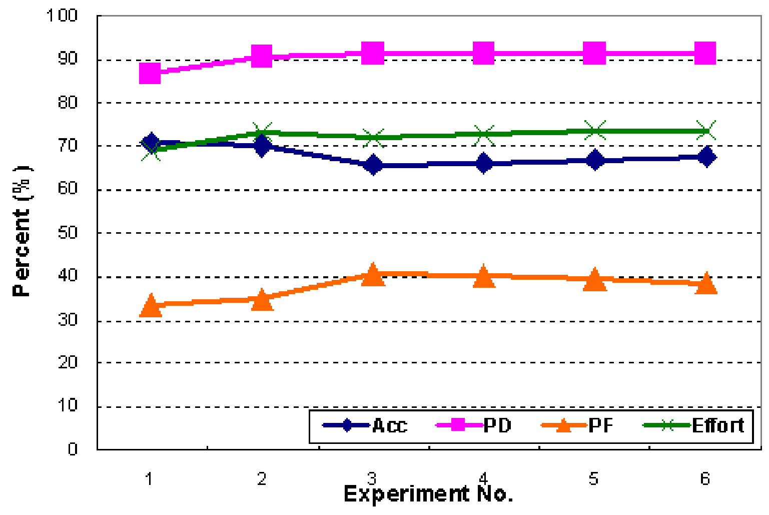


Figure 4.5: Prediction of fault-prone modules by the D-S networks on KC2b

Table 4.2: Tuning parameters of the D-S networks on KC2 (for Figure 4.3)

Experi-ment No.	Predictors (in order)	File	U_{min}	∇_{min}	Domain Factor τ	$Bel_x(1)$ $(x=1)$	$Bel_x(0)$ $(x=1)$	$Bel_x(1)$ $(x=0)$	$Bel_x(0)$ $(x=0)$
1	E, UniqOp, V	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
2	V,UniqOpnd,LOComment,T	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
3	E,UniqOpnd,LOComment,V	KC2a	0.04	0.25	0.65	0.9	0.1	0.1	0.7
4	E,UniqOpnd,LOComment,V	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
5	UniqOpnd,t,LOComment,V	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
6	UniqOpnd,V,T,LOComment	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
7	UniqOpnd,E,V	KC2a	0.04	0.25	0.65	0.9	0.1	0.1	0.7
8	UniqOpnd,E,V	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
9	I,D,UniqOpnd,v(G),iv(G), V,T,BranchCount, LOComment	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
10	I,V,E,D	KC2a	0.05	0.25	0.65	0.9	0.1	0.1	0.7
11	ev(G),LOComment,I	KC2b	0.02	0.25	0.65	0.9	0.1	0.1	0.7
12	ev(G),I,LOComment	KC2b	0.02	0.25	0.65	0.9	0.1	0.1	0.7
13	ev(G),TotalOpnd	KC2b	0.02	0.25	0.65	0.9	0.1	0.1	0.7

TIC procedure in SAS to generate the candidate predictors. We conducted three different experiments, organized as follows:

1. Randomly pick 0 or 300 data points from JM1a, plus all data from KC2a to build the D-S networks. Use the remaining data of JM1a for validation.

2. Randomly choose 9,795 data points from the transformed JM1 data sets to induce the D-S networks. Use the remaining data for validation.

3. Randomly pick 0, 300, or 1,000 data points from JM1b, plus all data from KC2b to build the D-S networks. Use the remaining data of JM1b for validation.

In JM1, the optimal results for maximal *Acc* and *PD*, as well as minimal *Effort* and maximal *PD* have been achieved by the tuning parameters shown in Table 4.3. Figure 4.6 shows the prediction results of the D-S networks on JM1. Experiments 1–3 followed experimental procedure 1 on JM1a. Experiments 4 and 5 followed the procedure 2 on JM1a. Experiments 6–13 followed the procedure 3 on JM1b. It is worth noticing that Experiments 1 and 2 used only KC2a data for building the D-S networks, and achieved above 75% of overall prediction accuracy *Acc*; Experiments 10–13 used only KC2b data for the D-S network induction, and achieved 89.5% to 94.8% of *PD* on JM1.

All the tuning parameters are listed in Table 4.3. For JM1 project, we need more than 9 predictors to make good prediction. The reason might be that for a large project like

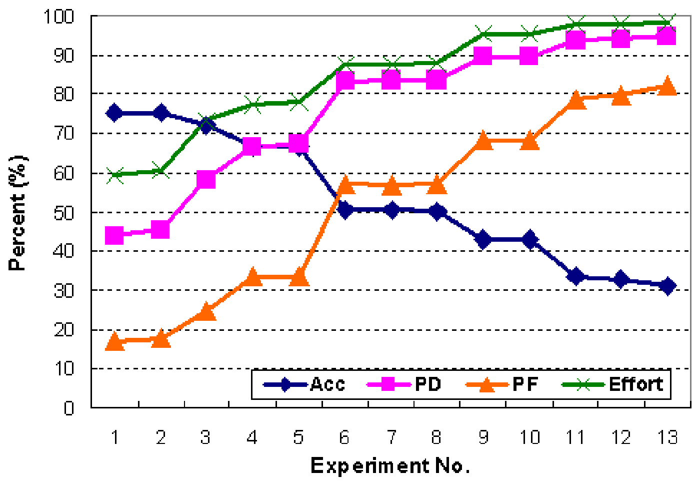


Figure 4.6: Prediction of fault-prone modules by the D-S networks on JM1

JM1, the information needed for overall accurate prediction is more intricate than that for a relatively small project like KC2. Again, the best predictors come from the logistic procedure. The *Information Gain Attribute Ranking* algorithm in WEKA produces similar results as the logistic procedure on JM1b; while the predictors selected by the rest of WEKA attribute selection algorithms do not result in optimal prediction on JM1.

4.7 Evaluation

In this section, we compare the performance of D-S networks with that of other statistical methods. In §4.7.1 through §4.7.5 and §4.7.6, the comparison criteria are overall accuracy *(Acc)* and defect detection rate *(PD)*. The D-S results are chosen for comparison by the following rules. For a pair of results, the predictions by the D-S network and the compared method, if one prediction is dominant over the other (see §4.4.3 for dominant rule), this pair of results is selected for evaluation. Otherwise, the pair with the closest distance between the compared criterion, either Acc or PD, is selected. For clarity, we use $< Acc_{DS}, PD_{DS} >$ to represent a result from D-S prediction, and $< Acc_{other}, PD_{other} >$ to represent a result from other methods compared. We define $Distance_{Acc}$, $Distance_{PD}$, and $Distance$ as follows:

$$Distance_{Acc} = |Acc_{DS} - Acc_{other}|$$

$$Distance_{PD} = |PD_{DS} - PD_{other}|$$

$$Distance = \min(Distance_{Acc}, Distance_{PD})$$

Table 4.3: Tuning parameters of the D-S networks on JM1 (for Figure 4.6)

Exp. No.	Predictors (in order)	File	Learning Data	U_{min}	∇_{min}	τ	$Bel_x(1)$ $(x=1)$	$Bel_x(0)$ $(x=1)$	$Bel_x(1)$ $(x=0)$	$Bel_x(0)$ $(x=0)$
1	iv(G),LOC,I,LOCodeAndComment, LOCode,L,D,ev(G),T,LOBlank	JM1a	520KC2a	0.03	0.35	0.6	0.65	0.1	0.1	0.9
2	LOC,iv(G),I,LOCodeAndComment, LOCode,L,D,ev(G),T,LOBlank	JM1a	520KC2a	0.03	0.35	0.6	0.65	0.1	0.1	0.9
3	LOC,iv(G),LOBlank,LOCodeAnd-Comment,LOCode,L,D,ev(G),T,I	JM1a	520KC2a +300JM1a	0.03	0.35	0.6	0.9	0.1	0.1	0.65
4	LOC,v(G),ev(G),iv(G),L,D,E,T, LOCode,LOBlank,LOCodeAnd-Comment,I,UniqOp,UniqOpnd	JM1a	9,795 JM1a	0.01	0.1	0.65	0.9	0.1	0.1	0.7
5	LOC,v(G),ev(G),iv(G),L,D,E,T, LOCode,LOBlank,LOCodeAnd-Comment,I,UniqOp,UniqOpnd	JM1a	9,795 JM1a	0.01	0.11	0.65	0.9	0.1	0.1	0.7
6	LOBlank,iv(G),LOC,ev(G),I, TotalOpnd,LOCode,LOCodeAnd-Comment,LOComment	JM1b	1,000 JM1b + 520KC2b	0.02	0.1	0.55	0.85	0.15	0.15	0.6
7	LOBlank,iv(G),LOC,ev(G),I, TotalOpnd,LOCode,LOCodeAnd-Comment,LOComment	JM1b	1,000 JM1b + 520KC2b	0.05	0.16	0.6	0.9	0.1	0.1	0.65
8	LOBlank,iv(G),LOC,ev(G),I, TotalOpnd,LOCode,LOCodeAnd-Comment,LOComment	JM1b	1,000 JM1b + 520KC2b	0.055	0.16	0.6	0.9	0.1	0.1	0.65
9	LOC,v(G),iv(G),ev(G),V,N,L,B, LOBlank,E,I,D,LOCode,LO-Comment,T,LOCodeAndComment, UniqOp, UniqOpnd,TotalOp, TotalOpnd,BranchCount	JM1b	300 JM1b + 520KC2b	0.055	0.16	0.6	0.9	0.1	0.1	0.65
10	LOC,v(G),iv(G),ev(G),V,N,L,B, LOBlank,E,I,D,LOCode,LO-Comment,T,LOCodeAndComment, UniqOp, UniqOpnd,TotalOp, TotalOpnd,BranchCount	JM1b	520KC2b	0.055	0.16	0.6	0.9	0.1	0.1	0.65
11	iv(G),LOBlank,LOCodeAnd-Comment,LOC,LOCode,N,I, TotalOpnd,LOComment,L,D, BranchCount,TotalOp,ev(G)	JM1b	520KC2b	0.055	0.16	0.6	0.9	0.1	0.1	0.65
12	iv(G),LOCodeAndComment, LOBlank,LOC,UniqOp,N,TotalOp, TotalOpnd,LOComment,L,D, BranchCount,I,ev(G),LOCode	JM1b	520KC2b	0.055	0.16	0.6	0.9	0.1	0.1	0.65
13	LOBlank,iv(G),LOC,ev(G),I, TotalOpnd,LOCode,LOCodeAnd-Comment,LOComment,E,v(G),D,N, V,T,B,UniqOp,UniqOpnd,TotalOp, L,BranchCount	JM1b	520KC2b	0.055	0.16	0.6	0.9	0.1	0.1	0.65

The pair of results that have minimal *Distance* are chosen for evaluation.

In §4.7.5, we compare D-S predictions with the results of ROCKY toolset based on three criteria: *Acc*, *PD*, and *Effort*. The two results are chosen for comparison at operational points where D-S and ROCKY yield the same PD.

4.7.1 D-S Networks vs. Logistic Regression

Logistic regression [107] [66] is useful to predict a dependent variable on the basis of independents (predictors). Logistic regression first transforms the dependent into a logit variable (the natural log of the odds of the dependent occurring or not), and then applies maximum likelihood estimation to determine logit coefficients in the model. In this way, logistic regression estimates the probability of a certain even occurring.

For comparison, The LOGISTIC procedure in SAS was used as a classifier to predict fault prone modules for KC2 and JM1. The original KC2 and JM1 data sets were input into the statistical software. The prediction results are shown in Figure 4.7 and Figure 4.8. The comparison of software quality prediction by D-S networks and logistic regression is illustrated in Figure 4.9 and Figure 4.10 on KC2 and JM1, respectively.

For KC2 (see Figure 4.9), the prediction by the D-S network has higher overall accuracy than that of logistic regression. The defect detection rate (PD) of the D-S networks is 1.9% to 5.7% higher than that of logistic regression. On the average, the defect detection rate of the D-S networks is 4.0% higher than that of logistic regression, while the overall accuracy (*Acc*) is 2.3% higher.

For JM1 (see Figure 4.10), the prediction accuracy of the D-S networks is generally higher than that of logistic regression. The defect detection rate (PD) of the D-S networks is 0.3% to 1.8% higher than that of logistic regression, while *Acc* is 0.5% to 0.9% higher. Considering the scale of JM1 project, the D-S networks correctly predict almost a hundred modules more than logistic regression.

4.7.2 D-S Networks vs. Discriminant Analysis

Discriminant analysis [68] is a very useful tool to determine which variables discriminant between two or more naturally occurring groups. It can also be used to classify cases into two or more groups with a better than chance accuracy. The basic idea is to determine whether groups differ with regard to the mean of a variable, and then to use that variable to predict group membership.

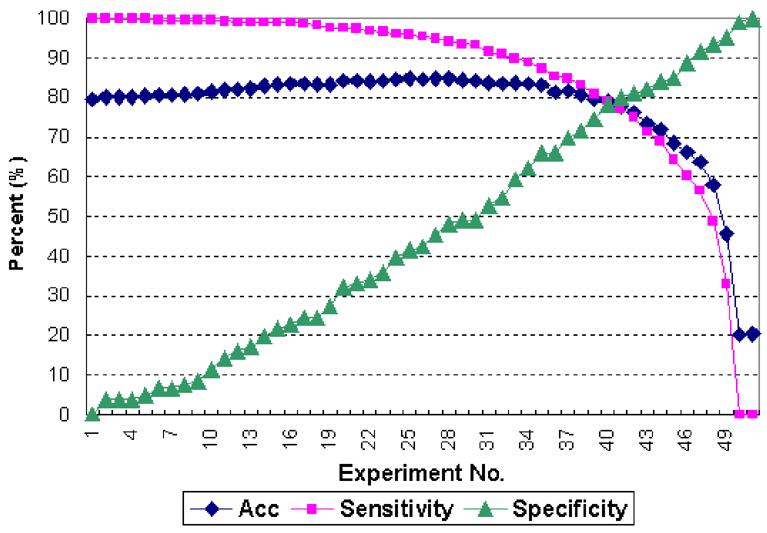


Figure 4.7: Prediction of fault-prone modules by logistic regression on KC2

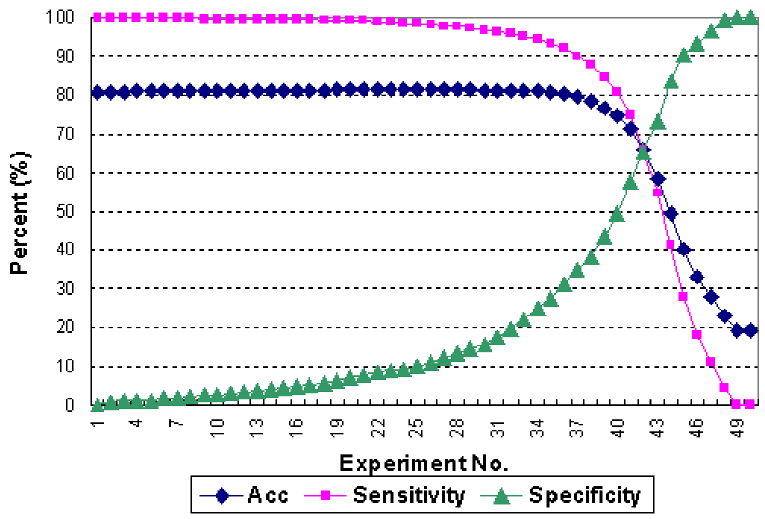


Figure 4.8: Prediction of fault-prone modules by logistic regression on JM1

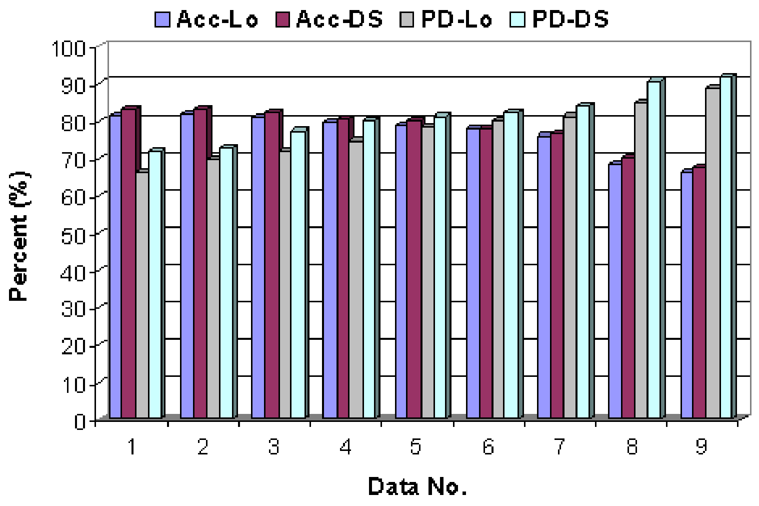


Figure 4.9: Prediction by the D-S networks vs. logistic regression on KC2

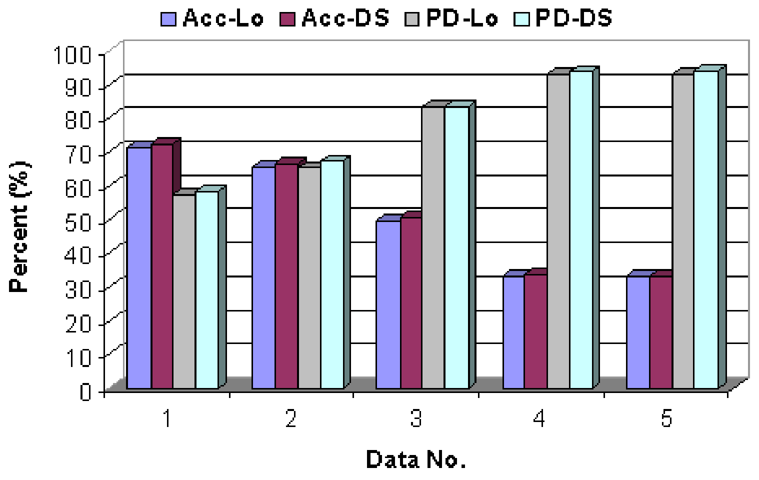


Figure 4.10: Prediction by the D-S networks vs. logistic regression on JM1

The DISCRIM procedure in SAS (linear discriminant function) was employed as a classifier on the original KC2 and JM1 data sets. The STEPDISC procedure in SAS was first used to perform stepwise discriminant analysis to pick the best predictors. Based on the predictors selected, the DISCRIM procedure gave the best prediction by discriminant analysis.

The optimal prediction of discriminant analysis on KC2 is compared with that of the D-S networks in Figure 4.11. These two methods have the same *Acc* 83.1%. However, the defect detection rate (*Specificity*) of the D-S networks is 4.7% higher than that of discriminant analysis.

The optimal prediction of discriminant analysis on JM1 is compared with that of the D-S networks in Figure 4.12. While *Acc* of discriminant analysis is slightly (1.6%) higher than that of the D-S networks, the defect detection rate (*Specificity*) of the D-S networks is 7.3% higher than that of discriminant analysis.

Assuming that the cost to release a defect into the later phase of the software life cycle caused by imprecise prediction of fault-prone modules is higher than the cost of software inspection, the D-S networks do have an advantage over discriminant analysis.

4.7.3 D-S Networks vs. See5/C5

See5/C5 is a commercial machine learning software [6]. Its earlier version is called C4.5. There are three classifiers in See5: *DecisionTree, RuleSet*, and *Boosting*. When See5 is invoked with the default values of all options, it constructs a decision tree for classification. Decision trees can sometimes be quite difficult to understand. An important feature of See5 is its ability to generate classifiers called *RuleSets* that consist of unordered collections of (relatively) simple if-then rules, derived from the constructed decision trees. Another innovation incorporated in See5 is *adaptive boosting*. The idea is to generate several classifiers (either decision trees or rulesets) rather than just one. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class.

The three classifiers of See5 were used to predict fault prone modules for KC2 and JM1. The original KC2 and JM1 data sets were input to See5 software. The prediction results are compared with D-S predictions in Figure 4.13 on KC2 and Figure 4.14 on JM1.

For KC2 project, D-S prediction has higher overall accuracy *Acc* (1.2%) and higher defect detection rate *PD* (17%) than the *Decision Tree* classifier of See5. The overall accuracy of D-S prediction (*Acc*) is comparable to that of *Rule Set* and *Boosting* classifiers of See5. However, the defect detection rate *PD* of the D-S network is 21.7% and 28.3% higher than the *Rules Set* and *Boosting* classifiers, respectively.

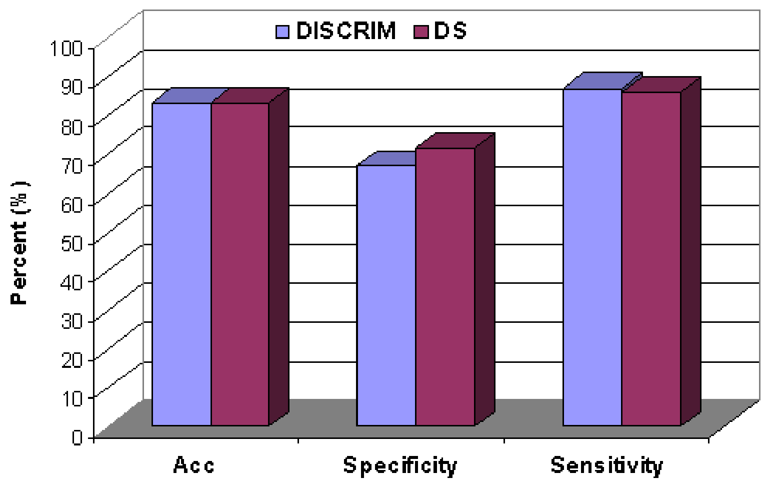


Figure 4.11: Prediction by the D-S networks vs. discriminant analysis on KC2

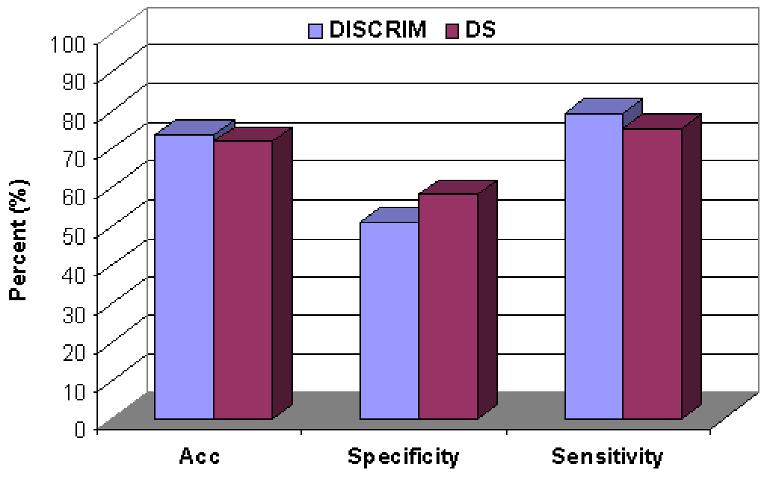


Figure 4.12: Prediction by the D-S networks vs. discriminant analysis on JM1

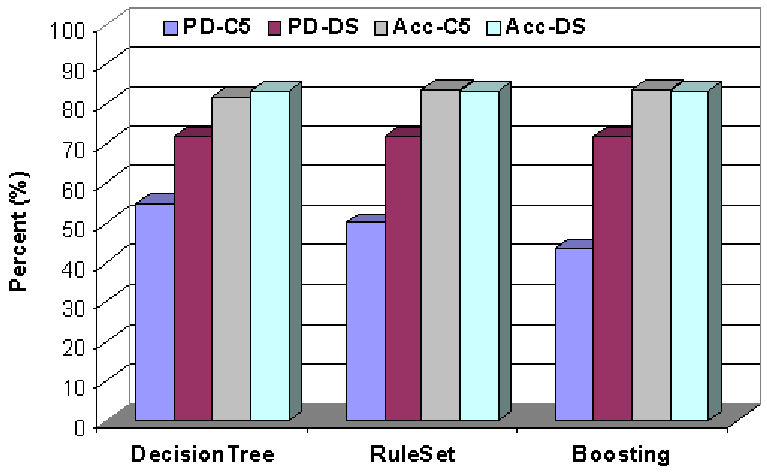


Figure 4.13: Prediction by the D-S networks vs. See5 Classifiers on KC2

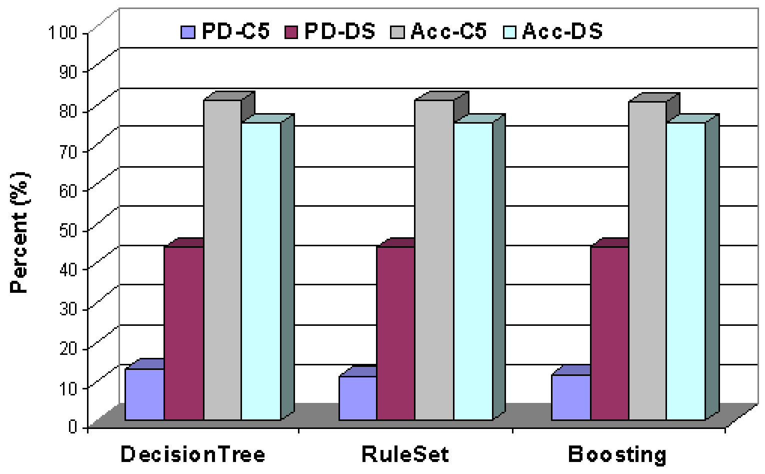


Figure 4.14: Prediction by the D-S networks vs. See5 Classifiers on JM1

For JM1 project, the overall accuracy of D-S prediction (*Acc*) is 5.6% less than See5 classifiers on the average. However, the defect detection rate (*PD*) of the D-S network is 31.8% higher than See5 classifiers on the average. In contrast to the See5 classifiers that learn from 9/10 of JM1 data, the D-S network is built from KC2 data only, which is more meaningful in practical applications.

4.7.4 D-S Networks vs. WEKA Learners

Table 4.4: WEKA Classifiers

Applied WEKA Classifiers	Generally Recommended	Best in (*) Our Study
DecisionStump	X	X
DecisionTable	X	
HyperPipes		X
IB1		X
IBk	X	XX
j48.J48	X	X
j48.PART	X	
KernelDensity		X
KStar		
Logistic		X
NaiveBayesSimple		
NaiveBayes	X	X
ZeroR	X	
OneR	X	
SMO	X	
VotedPerceptron		XX
VF1		XX
LogitBoost		X
NeuralNetwork		
ADTree		

(*) X: among the best on one project; XX: among the best on two projects

WEKA is a collection of machine learning algorithms for solving real-world data mining problems [153]. It contains 41 algorithms for classification and numeric prediction (the most important ones are explained in [154]). We applied all the 41 classifiers to KC2 and JM1. 20 classifiers are applicable to our data sets. For each data set, the best classifiers are selected according to $< Acc, PD >$ pair by the domination rule. The applied WEKA classifiers, and the recommended important ones [154], as well as the best performers in our case studies are listed in Table 4.4. It can be observed from Table 4.4 that, although not recommended as the most reliable classifiers in [154], some algorithms such as *VotedPerceptron* and *VF1* turn out to be the best performers on our data sets. The best classifiers of WEKA are compared with D-S predictions in Figure 4.15 on KC2 and Figure 4.16 on JM1.

For KC2 project, compared with the *LogitBoost* classifier of WEKA, the overall prediction accuracy of the D-S network is 2% lower. However, the defect detection rate (*PD*) of D-S prediction is 36% higher. For the rest of the selected WEKA classifiers, D-S predictions have both higher *Acc* and *PD*.

For JM1 project, compared with the *Logistic*, *KernelDensity*, *NaiveBayesSimple*, and *J48* classifiers, the overall prediction accuracy *Acc* of the D-S prediction is 5.3% lower, and the defect detection rate *PD* is 25% higher, on the average. The D-S prediction has comparable overall accuracy as the *IBk* and *IB1* classifiers. However, the defect detection rate of D-S predictions is 6.3% and 7% higher, respectively. The D-S networks have higher overall prediction accuracy and defect detection rate than the *VotedPerceptron* and *VF1* classifiers. It is striking that the *HyperPipes* classifier detects 100% of defect modules. The D-S network detects about 5% less defect modules than *HyperPipes*, but the overall accuracy is 21% higher. In conclusion, D-S networks can achieve better predictions than most WEKA classifiers by learning from another project, KC2, only. D-S predictions are not outperformed by any of the WEKA classifiers.

4.7.5 D-S Networks vs. ROCKY

ROCKY is a defect detector toolset used in experimental selection of modules for software inspection at NASA IV&V facility in Fairmont, West Virginia [110]. The main aim of ROCKY is to facilitate minimal inspection effort (recommend inspecting the minimal number of code lines) while achieving as good as possible defect detection rate *PD*. ROCKY detectors are built by exhaustively exploring all singleton rules of the form:

$$attribute \geq threshold$$

where *attribute* is every numeric attribute present in a data set, and *threshold* is certain percentile value of the corresponding *attribute* [110]. ROCKY was applied to predicting fault prone modules in KC2 and JM1 data sets with all McCabe and Halstead metrics. There is no observation derived concerning which metrics perform better than the others, in

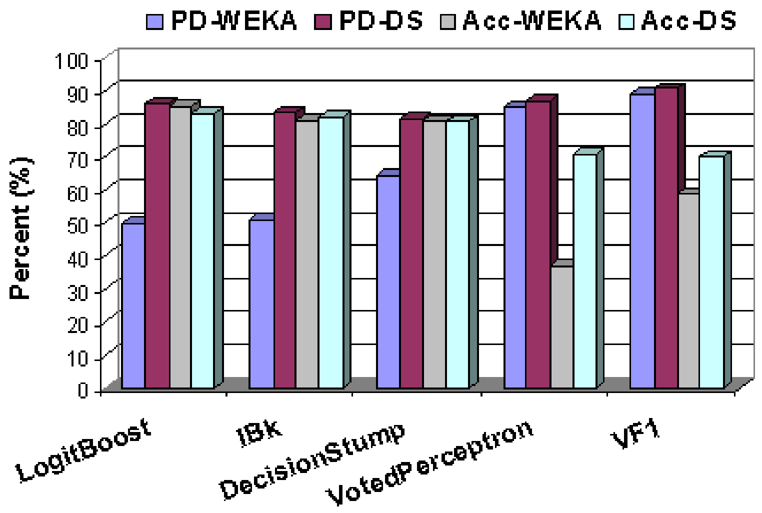


Figure 4.15: Prediction by the D-S networks vs. WEKA Classifiers on KC2

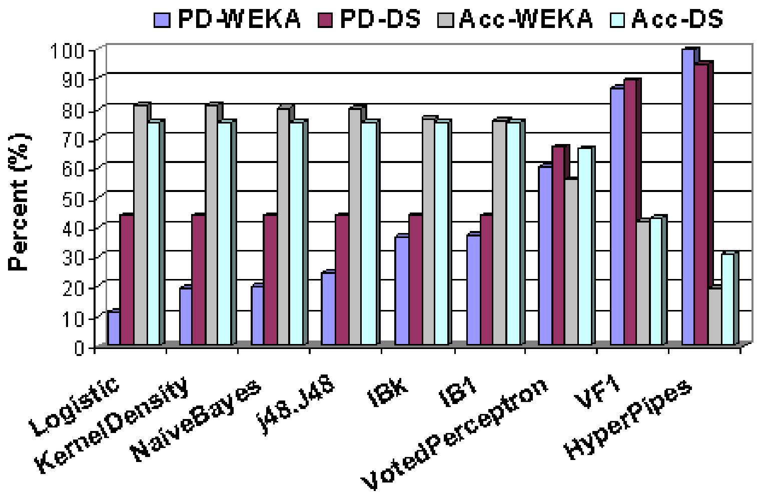


Figure 4.16: Prediction by the D-S networks vs. WEKA Classifiers on JM1

general. Predictions based on individual metrics were presented in [110].

Compared with the optimal performance of ROCKY on KC2 project, we notice several advantages of D-S networks. First of all, D-S predictions have higher overall accuracy *Acc* than ROCKY in entire data range, to achieve the same defect detection rate *PD*. Secondly, *Effort* is generally higher than *PD*, except for one or two data points for ROCKY [110]. On the other hand, *Effort* is generally lower than *PD* for the entire data range for D-S networks. Especially significant advantage of D-S networks can be seen in Figure 4.5 depicting the effort on KC2b. There, *Effort* is 18.2% lower than *PD* on the average in the high *PD* region (86.8% to 91.5%). In other words, as mentioned earlier, inspecting approximately 72% of the code (LOC) would expose over 91% of the fault prone modules. In contrast, ROCKY recommended reading 94% of the code to discover 91% of fault prone models. The comparison of the *Effort* resulting from D-S predictions and ROCKY toolset predictions is shown in Figure 4.17. Based on the available data [110], ROCKY predictions lead to higher levels of effort than D-S network predictions.

For JM1 project, ROCKY used all information available for building models, which is a different scenario from ours. The overall accuracy *Acc* of the D-S networks is higher than that of ROCKY to achieve the same defect detection rate *PD*. The *Effort* of the D-S networks is comparable to that of ROCKY for the same *PD*. Detailed comparison is shown in Figure 4.18. The optimal performance of ROCKY was selected for comparison.

4.7.6 D-S Networks vs. Random Forests

Random forests, proposed by Breiman in 2001 [16] [4], are a recent extension of decision tree learning. Instead of generating one decision tree, this methodology generates hundreds or even thousands of trees using subset of the training data. Classification decision is obtained by voting. Specifically, the random forest classifies a new object from an input vector by examining the input vector down each of the trees in the forest. Each tree casts a unit vote at the input vector by giving a classification. The forest selects the classification having the most votes over all the trees in the forest.

Each tree is grown as follows:

- If the number of cases in the training set is N, sample N cases at random, with *replacement* from the original data. This sample will be the training set for growing the tree.

- At each node, m predictors are randomly selected out of the M input variables ($m \ll M$) and the best split on these m predictors is used to split the node. The value of

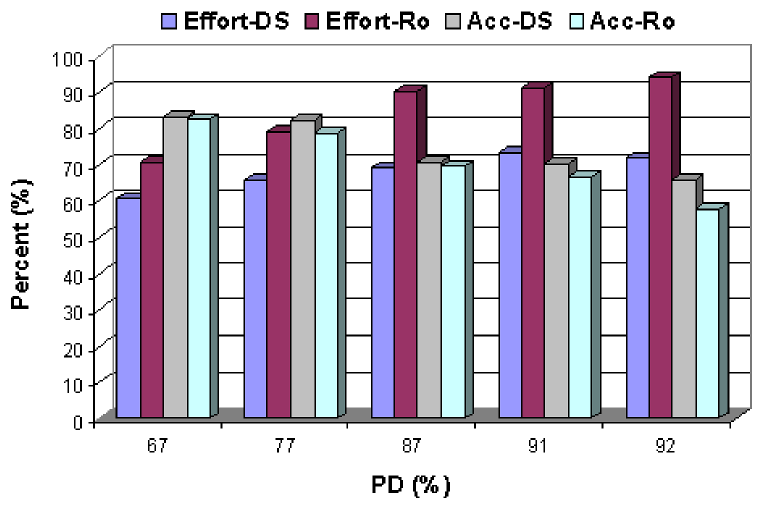


Figure 4.17: Prediction by the D-S networks vs. ROCKY on KC2 (performance of ROCKY taken from [110])

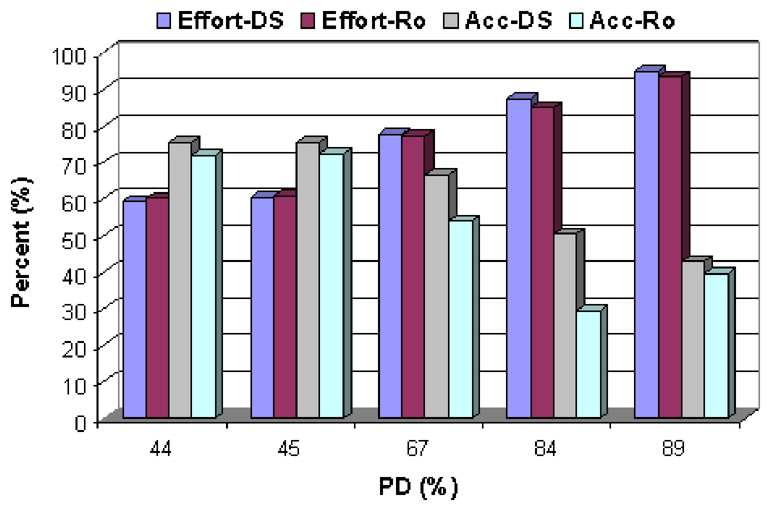


Figure 4.18: Prediction by the D-S networks vs. ROCKY on JM1 (performance of ROCKY taken from [110])

m is held constant during the forest growing. By default, $m = \sqrt{M}$ (to achieve near optimal results).

- Each tree is grown to the largest extent possible. There is no pruning.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This **oob (out-of-bag) data** is used to get a running unbiased estimate of the classification error as trees are added to the forest. In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. The out-of-bag estimates are unbiased [16].

We applied the random forest classifier to predicting fault prone modules. In our case studies, the two NASA data sets are mission critical projects. The fault-prone modules constitute only a small portion in the data sets. As mentioned earlier, there is a tradeoff between the overall accuracy and the defect detection rate. Random forests, trying to minimize overall error rate, will keep the error rate low on the large class while letting the smaller classes have a large error rate. This will obviously impose problems for software quality prediction, because many fault-prone modules will be misclassified as non-fault prone ones and hence might be released into the later phase of the software life cycle.

We solve this problem by changing the default *cutoff* value of random forests. As a result, different weights are setting for the classes and the prediction errors can be balanced toward the way it is preferred. In this way, random forests can be tuned to achieve a wide range of overall accuracy and defect detection rate, and to obtain optimal results.

In our study, we first generate a two dimensional vector for *cutoff* values, and then apply the random forest classifier with the individual *cutoff* value to predicting fault prone modules. Each tree in the forest is built from randomly selected two-thirds of the data set, and the remaining one-third of the data are the test set for validation.

For each available data set, we generated 45 cutoff vectors in the form of 45×2 matrix. All values in the matrix are of the form $k * 0.1$, where $k \in \{1, 2, ..., 9\}$. The sum of the two cutoff values in the same row can be no greater than 1, to meet the requirement for *cutoff* values in random forests. Each row in the matrix represents a pair of thresholds for one classification result. In this way, we get combinations of *cutoff* values that result in different classifications by the random forest. Through experimentation, we determined that additional cutoff vectors (with smaller increment values) give us redundant results.

By using 45 rows of the cutoff vector as the *cutoff* values for random forests, we get 45 experimental results for each data set. Each result is generated by a random forest with 500 trees (the default value in random forests). During the classification, 5 variables (also the default value) were randomly selected to split each node in the tree.

Random Forests in R [3] (a statistical software) was applied to KC2 and JM1 as a classifier. The comparison of the performance of D-S Networks vs. Random Forests are depicted in Figure 4.19 and Figure 4.20 for KC2 and JM1, respectively.

For KC2 project (in Figure 4.19), we can observe that, in the whole range, D-S networks detect more fault-prone modules than Random Forests with the same overall accuracy *Acc*. The defect detection rate PD of D-S networks is 16% to 51% higher than that of Random Forests. On the average, D-S networks achieves 29.5% higher defect detection rate PD than Random Forests with the same overall accuracy *Acc*.

For JM1 project(in Figure 4.20), D-S networks detect 3% more fault-prone modules than Random Forests, while Random Forests achieve 6% higher overall accuracy *Acc*. The highest defect detection rate (*Specificity* or PD) that Random Forests can achieve is 80.4%. In contrast, D-S networks can reach up to 94.8% defect detection rate.

4.8 Discussion

An analysis of several recent projects revealed that 20% of the modules are responsible for 80% of the malfunctions of the whole project [40]. The two data sets in our case studies demonstrate interesting extension of this rule to static artifacts, code defects. The goal of software quality prediction is to identify these critical modules as early as possible.

This chapter compares the performance of machine learning algorithms on software quality data sets. Some of them are not suitable for software quality prediction because of their low defect detection rate *PD*. Others, able to achieve over 60% defect detection rate, are candidate methods for software quality prediction. These methods include: logistic regression and discriminant analysis of SAS, as well as the *DecisionStump*, *VotedPerceptron*, *VF1*, and *HyperPipes* classifiers of WEKA, and Random Forests of R. Compared with these methods, D-S networks can achieve higher overall prediction accuracy *Acc* and defect detection rate *PD* (*Specificity*). D-S predictions are not outperformed by any of the methods discussed in this chapter. we are also not aware of any other methods that outperform D-S networks. The results of statistical significant difference test (normal distribution test) of D-S predictions vs. other methods using 0.05 level of significance are listed in Table 4.5 for KC2 project and Table 4.6 for JM1 project. In each table, a (+) sign means the D-S prediction is significantly better than the compared method; a (−) sign means that the D-S prediction is significantly worse than the compared method. From the results, D-S predictions have significantly higher *Acc* and/or PD than almost all the other methods. Actually, the difference between D-S

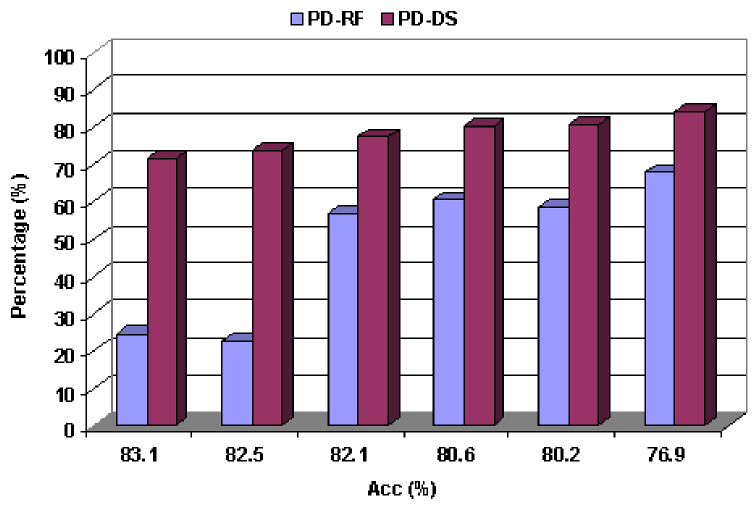


Figure 4.19: Prediction by the D-S networks vs. Random Forests on KC2

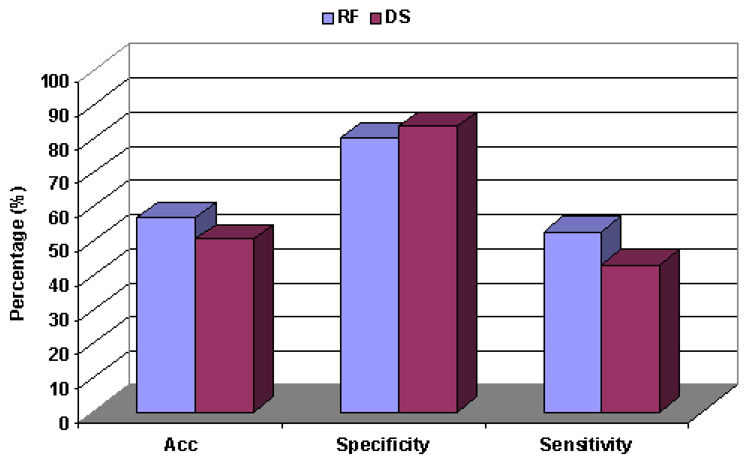


Figure 4.20: Prediction by the D-S networks vs. Random Forests on JM1

Table 4.5: Statistical Significant Difference Test of D-S vs. Other Methods Using 0.05 Level of Significance on KC2

Method Compared	Software	Acc	Acc_{DS}	$\lvert Z_{Acc}\rvert$	Z_{Acc} Significant?	PD	PD_{DS}	$\lvert Z_{PD}\rvert$	Z_{PD} Significant?
Logistic	SAS	0.762	0.802	1.56	No (+)†	0.811	0.811	0.00	No
Discriminant	SAS	0.831	0.831	0.00	No	0.670	0.717	0.74	No
DecisionTree	See5	0.819	0.831	0.51	No	0.547	0.717	2.57	Yes (+)
RuleSet	See5	0.836	0.831	0.22	No	0.500	0.717	3.24	Yes (+)
Boosting	See5	0.835	0.831	0.17	No	0.434	0.717	4.17	Yes (+)
LogitBoost	WEKA	0.852	0.831	0.93	No	0.500	0.860	5.62	Yes (+)
IBk	WEKA	0.812	0.821	0.37	No	0.509	0.833	5.01	Yes (+)
DecisionStump	WEKA	0.808	0.808	0.00	No	0.642	0.816	2.86	Yes (+)
VotedPerceptron	WEKA	0.367	0.708	11.02	Yes (+)	0.849	0.868	0.40	No
VF1	WEKA	0.586	0.702	3.89	Yes (+)	0.887	0.906	0.45	No
ROCKY	ROCKY	0.832	0.831	0.04	No	0.556	0.717	2.44	Yes (+)
ROCKY	ROCKY	0.576	0.658	2.72	Yes (+)	0.917	0.915	0.05	No
Random Forests	R	0.831	0.831	0.00	No	0.245	0.717	6.88	Yes (+)

(†) statistical significant at 0.05 level of significance

predictions and that of the other methods is very large in many cases, indicated by the large Z values ($|Z| \geq 1.96$ corresponds to statistical significant difference at 0.05 level of significance). D-S predictions are not outperformed by any other methods (no double $(-)$ signs).

The work presented in this chapter is the first attempt to apply Dempster-Shafer belief networks to software quality prediction. This novel methodology has following unique aspects and advantages:

1. It can be tuned to meet different optimization requirements, such as maximizing *Acc* and *PD*, or minimizing *Effort* while maximizing *PD*. Other methods mentioned in this chapter cannot be tuned in such a way.

2. It can be tuned to output a wide spectrum of overall accuracy *Acc* and defect detection rate *PD*. The defect detection rate can reach over 90%. It allows software testers to choose their preferable range according to their time and budget constraints.

3. It is highly efficient. It takes several milliseconds to build a D-S network and perform 10-fold cross validation, even on a large data set like JM1. On the contrary, some algorithms in WEKA software need much more computing time to perform 10-fold cross validation on JM1. For example, the *IB1* classifier of WEKA needs 20 minutes, *IBk* 19 minutes, *KernelDensity* 31 minutes, and *Neural Network* over 90 minutes on

Table 4.6: Statistical Significant Difference Test of D-S vs. Other Methods Using 0.05 level of Significance on JM1

Method Compared	Software	Acc	Acc_{DS}	$\|Z_{Acc}\|$	Z_{Acc} Significant?	PD	PD_{DS}	$\|Z_{PD}\|$	Z_{PD} Significant?	Exp. No.*
Logistic	SAS	0.658	0.666	1.25	No (+)†	0.654	0.672	1.24	No (+)†	5
Discriminant	SAS	0.736	0.720	2.65	Yes (−)	0.509	0.582	4.76	Yes (+)	3
DecisionTree	See5	0.811	0.754	10.19	Yes (−)	0.131	0.439	22.14	Yes (+)	1
RuleSet	See5	0.810	0.754	10.00	Yes (−)	0.115	0.439	23.49	Yes (+)	1
Boosting	See5	0.808	0.754	9.63	Yes (−)	0.118	0.439	23.23	Yes (+)	1
Logistic	WEKA	0.813	0.754	10.57	Yes (−)	0.117	0.439	23.32	Yes (+)	1
KernelDensity	WEKA	0.810	0.754	10.00	Yes (−)	0.194	0.439	17.09	Yes (+)	1
NaiveBayes	WEKA	0.803	0.754	8.70	Yes (−)	0.197	0.439	16.86	Yes (+)	1
j48.J48	WEKA	0.802	0.754	8.52	Yes (−)	0.247	0.439	13.12	Yes (+)	1
IBk	WEKA	0.761	0.754	1.20	No	0.369	0.439	4.63	Yes (+)	1
IB1	WEKA	0.756	0.754	0.34	No	0.376	0.439	4.16	Yes (+)	1
VotedPerceptron	WEKA	0.560	0.666	16.05	Yes (+)	0.604	0.672	4.59	Yes (+)	5
VF1	WEKA	0.418	0.429	1.64	No (+)‡	0.868	0.894	2.61	Yes (+)	9
HyperPipes	WEKA	0.195	0.310	19.53	Yes (+)	1.000	0.948	10.60	Yes (+)	13
ROCKY	ROCKY	0.752	0.752	0.00	No	0.338	0.453	7.63	Yes (+)	2
ROCKY	ROCKY	0.540	0.666	19.00	Yes (+)	0.671	0.672	0.07	No	5
Random Forests	R	0.570	0.504	8.77	Yes (−)	0.804	0.836	2.70	Yes (+)	8

(*) in Table 4.3
(†) statistical significant at 0.2 level of significance; (‡)statistical significant at 0.1 level of significance

a Windows XP machine with a 1.60 GHz Pentium 4 processor and 256 MB of RAM. Random Forests of R can not even run on this Windows XP machine. We ran the JM1 experiments on a server with a 248MHz Sun Microsystem's Ultra SPARC-II sun4u processor and 2 GB of RAM.

4. It has higher overall prediction accuracy *Acc* and defect detection rate *PD* than logistic regression, discriminant analysis, as well as the algorithms in two machine learning software packages WEKA and See5.

5. When applied with case-based reasoning, the implication network can be constructed by learning from a smaller project's data set and achieve higher overall prediction accuracy and/or defect detection rate than most of other methods, which learn from 9/10 of the data describing the same project. From Table 4.6, we can observe that only two comparisons used the result from Experiment 5, which learned D-S networks from JM1 data. Other comparisons used results from experiments that learned D-S networks from KC2 data only (except Experiment 9 that used additional 300 JM1 data). These results are presented in §4.6.

6. It is more effort economic when recommending fault prone modules for software inspection than NASA's toolset ROCKY.

It is a common practice to use categorized data sets to predict software quality and reliability by belief networks such as the Bayesian networks [45]. D-S networks deal with discrete data sets. The case studies in this chapter used two different ways to discretize a continuous data set: partition by mean, and partition by median. The experiment results indicate that data partitioned by mean have higher overall accuracy *Acc*, while data stratified by median have higher defect detection rate *PD* for software quality prediction. Currently, we do not have an explanation for it. However, if this phenomenon is observed on other data sets in future studies, its possible explanation will provide a useful guidance for data set treatment, aiming at meeting different real world requirements (overall *Acc* or *PD*). As illustrated earlier, when the D-S networks were applied to the transformed data sets, they generally achieved higher prediction accuracy and defect detection rate than the other methods in our experiments. This indicates the great potential of the D-S belief networks.

The logistic procedure was used to select *subsets* of attributes. The results demonstrate that the predictors selected by the logistic procedure have better predictive performance than those selected by the attribute selection algorithms in WEKA, which rank the individual attributes. A possible explanation is that highest ranked single attributes together may not result in the best prediction, while the subsets of attributes selected by the logistic procedure result in the best combination for the optimal prediction.

The selected software metrics are generally different from project to project. In KC2, the selected good predictors for fault prone modules are: E (Effort), V (Volume), and LOComment (Lines of Comment). These three software metrics are also among those selected in JM1. We need more software metrics to make good predictions for JM1, which might be an indictor that JM1 is a more complicated project than KC2 (at least much larger in size). There may also exist differences between these two projects in the software development life cycle, which are not represented by the software metrics collected in the case studies.

Our methodology is distinguished by its ability of tuning to meet different requirements. We observed that the sequence of the predictors taken into the D-S networks has effect on optimal network inference, which is consistent with the observation presented in [96]. We found that the first four or five predictors have the greatest impact on the prediction accuracy. Currently, we do not have an algorithm to identify the "magic" sequence. For projects like KC2, where 2-5 predictors yield the optimal prediction, identifying the "magic" sequence is not a serious problem. However, for projects like JM1, where at least nine predictors are needed for the optimal prediction, we could only rely on experience and intuition to find out the best predictors sequence. One possible research direction is to explore the relationship between the sequence of the predictors and the entropy (the measure of uncertainty) of the inducted D-S network. If each time the predictor taken into the D-S network is the one that is most likely to reduce the entropy of the entire network, the algorithm to decide the sequence of the predictors is then an optimization algorithm discussed in [96].

Chapter 5

Information Fusion of Correlated Evidence

5.1 Introduction

Information fusion is a very important area of evidence combination. The goal is to fuse different sources of information for a more precise probabilistic assessment of real-world representations. Most fusion techniques share a common assumption that different sources of information are independent from each other, which is restrictive and unrealistic in many situations. As mentioned earlier, the Murphy's rule of combination is suitable for situations where the evidence is from the *same* source. However, there are other situations where sources of information are *not* distinct, but is *not* from the same source, either. Correlated evidence is one such example. Currently, no framework exists to combine evidence and take correlation in to account.

In this chapter, we propose a methodology based on the Murphy's rule of combination and fuzzy logic to combine correlated information. It is a general methodology for combining evidence, either correlated or distinct. Our approach contains following steps: (1) deciding which sources of information are correlated with each other; (2) calculating correlation coefficient between possibly correlated information; (3) ranking the importance of each source of information; (4) developing a fuzzy rule set for the contextual weighting parameter of the Murphy's rule of combination, based on the importance and correlation of each source of information; (5) applying the Murphy's rule to combine available information.

We also provide an algorithm and proof for the upper and lower bound of the belief function of the combination results for the Murphy's rule, when the contextual weighting parameter remains constant for each source of information (meaning that each source of information carries the same weight during the fusion). To the best of our knowledge, this is the first attempt to provide such an algorithm or a proof. This theoretical aspect is applied to constructing an Online safety Monitor for adaptive intelligent flight control systems, which will be introduced in Chapter 6 .

The reminder of the chapter is organized as follows. The preliminaries of the D-S framework is introduced in §5.2. The proposed methodology is described in §5.3. The algorithm to obtain the upper and lower bound of combination results and its proof are provided in §5.4. Finally, §5.5 summarizes the chapter.

5.2 Methodology

Currently, no framework exists to combine evidence and take correlation in to account. However, such a framework is needed in many applications. For instance, there are many available software reliability assessment methods. There is, however, no methodology to combine them for a more precise prediction, because many reliability assessment methods are correlated with each other. In this section, we propose a methodology based on the Murphy's rule and fuzzy logic to combine evidence, including correlated information. We choose the Murphy's rule of combination because:

- It is a general form of the Dempster's rule of combination. It can combine both dependent and independent sources of evidence.

- It allows to control how much a new source of evidence should influence the total belief, based on whether the new evidence is more *informed* than the previously observed evidence.

- Same as the Dempster's rule of combination, contradictory evidence cancels out in the Murphy's rule, which is advantageous in many applications because it smoothes out the noise.

- The non-commutativity of the Murphy's rule of combination is especially suitable for many applications, for instance realtime applications where the order of the observations should affect the outcome [113]. In addition, the non-commutativity of the Murphy's rule can give the upper and lower bound of the belief function in evidence combination, which is beneficial in many applications. One such application is discussed in our case studies.

Specifically, our approach contains following steps:

1. *Deciding whether or not the sources of information to be combined are correlated with each other*: It requires the analysis based on domain knowledge. For example, if the sources of information are different reliability estimation methods, we need to explore how each method is developed and what factors are involved, and thus decide whether or not is dependent on other methods.

2. *Calculating the correlation coefficient between possibly correlated sources of information*: In this step, either field data or data from simulation can be used to calculate correlation coefficient. Pearson's correlational method is used in our study.

3. *Ranking the importance of each source of information*: Credibility of each source of information is a prior consideration of information fusion. Highly credible information source should carry more weight then less credible ones.

4. *Developing a fuzzy rule set to determine the contextual weighting parameter n for the belief revision function* (Equation 2.10) of the Murphy's rule of combination, based on the importance and correlation of each source of information. Note that if none of the available sources of information are correlated with each other and they are equally important, we simply assign $n = 1$ in the belief revision function. In this case, it is the same as using the Dempster's rule of combination to combine independent sources of information. In other cases, the general rules are (1) the more important (credible) an information source, the larger the n value; (2) the more independent (distinct) an information source, the larger the n value.

5. *Applying the Murphy's rule of combination based on the fuzzy rule set to combining available sources of information, uncorrelated or correlated*: Based on the value of the contextual weighting parameter n defined from the previous step, the Murphy's rule of combination can be readily applied to fusing various information sources, and giving each source its corresponding weight during the combination.

This is a very general outline of the proposed methodology. We will illustrate two applications of our methodology in the following chapter. As mentioned above, the Murphy's rule of combination is non-commutative with $n < 1$. Different order of evidence combination results differently. We will give an algorithm to find the upper and lower bound of the belief function in the combination results.

5.3 Upper and Lower Bound of the Combination Belief Function

In applications where available sources of information are equally important and correlated with each other, we need to treat these sources of evidence equally. In these cases, the contextual weighting parameter n of the belief revision function should be held constant, as each source of evidence is taken into the combination. As mentioned in the previous section, the Murphy's rule of combination is non-commutative. Therefore, different order of evidence combination has different result. We develop an algorithm to obtain the upper and lower bound of the combination results if the contextual weighting parameter of the belief revision function, n, is constant. Our algorithm has following assumptions:

- Since the number of all possible combination forms is exponential of the number of evidence sources, we restrict to one combination form: each source of evidence is taken into the combination one at a time. In other words, available sources of evidence are combined as $(((Bel_i \oplus Bel_j) \oplus Bel_k) \oplus Bel_l) \cdots \oplus Bel_n$. We do not consider other combination forms for instance $((Bel_i \oplus Bel_j) \oplus (Bel_k \oplus Bel_l)) \cdots \oplus (Bel_m \oplus Bel_n)$.

- Available sources of evidence are given equal credit, which means that the n parameter in the Murphy's rule of combination (Equation 2.10) is constant.

- The evidence combination process in Dempster-Shafer theory is exponential of the number of the elements in the frame of discernment. In order to simplify the situation, we restrict ourselves to the situation where the frame of discernment has only two elements, $\Theta = \{A, B\}$, and each source of evidence assigns $m(A) + m(B) = 1$ as the basic probability function.

Based on above assumptions, our algorithm states that if we sort basic probability assignments $m_i(A) = a_i$ in increasing order, i.e., $a_1 < a_2 < a_3 < a_4 \cdots < a_n$, and combine them in this order, i.e., $(((m_1 \oplus m_2) \oplus m_3) \oplus m_4) \cdots \oplus m_n$, then the combination result $m(A)$ gives the maximum value of the belief function $m(A)$; if we sort basic probability assignments $m_i(A) = a_i$ in decreasing order, i.e., $a_1 > a_2 > a_3 \cdots > a_n$, and combine them in this order, i.e., $(((m_1 \oplus m_2) \oplus m_3) \oplus m_4) \cdots \oplus m_n$, then the combination result $m(A)$ is minimum. For two sources of evidence with $m_i(A) = m_j(A)$, the order of these two sources of evidence during the combination does not matter.

The proof is provided as follows. We use mathematical induction in the following proof.

Theorem 1. *In the Murphy's rule of combination, if the contextual weighting parameter in the belief revision function is a constant not equal to 1, and the evidence combination is conducted as* $(((m_1 \oplus m_2) \oplus m_3) \oplus m_4) \cdots \oplus m_n$*, for all sources of evidence that assign* $m(A) + m(B) = 1$ *for the frame of discernment* $\Theta = \{A, B\}$*, the combination in the order of* $m_1(A) < m_2(A) < m_3(A) \cdots < m_n(A)$ *leads to the maximum combined probability* $m(A)$ *assigned to* A*, and the combination in the order of* $m_1(A) > m_2(A) > m_3(A) \cdots > m_n(A)$ *leads to the minimum combined probability* $m(A)$ *assigned to* A*.*

Proof. For the Murphy's rule of combination, if the n parameter is constant in the belief revision function, then the combination order of two sources of evidence does not matter. Basically, $m_1 \oplus m_2$ is the same as $m_2 \oplus m_1$. The combination order only matters if there are at least three sources of evidence. So we start the induction with three sources of evidence, m_1, m_2, m_3.

1. **Initial Step.**

   For three sources of evidence, suppose $m_i(A) = a_i$ is sorted in increasing order, represented by $a_1 < a_2 < a_3$. Let m_{12} represent the combination result of m_1 and m_2, $m_1 \oplus m_2$. According to the Murphy's rule of combination (see Table 5.1),

Table 5.1: Combination of two sources of evidence m_1 and m_2

m_2/m_1	$\{A\}$ a_1	$\{B\}$ $1-a_1$
$\{A\}$ a_2	$\{A\}$ a_1a_2	$\{\emptyset\}$ $a_2(1-a_1)$
$\{B\}$ $1-a_2$	$\{\emptyset\}$ $a_1(1-a_2)$	$\{B\}$ $(1-a_1)(1-a_2)$

$$m_{12}(A) = \frac{(a_1a_2)^n}{(a_1a_2)^n + (1-a_1)^n(1-a_2)^n}$$

$$m_{12}(B) = \frac{(1-a_1)^n(1-a_2)^n}{(a_1a_2)^n + (1-a_1)^n(1-a_2)^n}$$

   Let m_{123} represent the combination result of m_{12} and m_3, representing $(m_1 \oplus m_2) \oplus m_3$. According to the Murphy's rule of combination,

$$m_{123}(A) = \frac{(a_1^n a_2^n a_3)^n}{(a_1^n a_2^n a_3)^n + ((1-a_1)^n(1-a_2)^n(1-a_3))^n}$$

Similarly, if we combine m_1 with m_3 first and then combine m_2, representing $(m_1 \oplus m_3) \oplus m_2$, we have m_{132} representing the combination result:

$$m_{132}(A) = \frac{(a_1^n a_3^n a_2)^n}{(a_1^n a_3^n a_2)^n + ((1-a_1)^n(1-a_3)^n(1-a_2))^n}.$$

Note that

$$\frac{1}{m_{123}(A)} = 1 + \frac{((1-a_1)^n(1-a_2)^n(1-a_3))^n}{(a_1^n a_2^n a_3)^n},$$

$$\frac{1}{m_{132}(A)} = 1 + \frac{((1-a_1)^n(1-a_3)^n(1-a_2))^n}{(a_1^n a_3^n a_2)^n}.$$

Let

$$C_1 = \frac{(1-a_1)^n(1-a_2)^n(1-a_3)}{a_1^n a_2^n a_3},$$

$$C_2 = \frac{(1-a_1)^n(1-a_3)^n(1-a_2)}{a_1^n a_3^n a_2},$$

Then

$$\frac{1}{m_{123}(A)} = 1 + C_1^n,$$

$$\frac{1}{m_{132}(A)} = 1 + C_2^n.$$

We can get

$$\frac{C_1}{C_2} = \frac{(1-a_2)^{n-1}a_3^{n-1}}{a_2^{n-1}(1-a_3)^{n-1}} = \frac{(\frac{1-a_2}{a_2})^{n-1}}{(\frac{1-a_3}{a_3})^{n-1}} = (\frac{\frac{1}{a_2}-1}{\frac{1}{a_3}-1})^{n-1} = (\frac{\frac{1}{a_3}-1}{\frac{1}{a_2}-1})^{1-n}$$

Since $a_2 < a_3 < 1$, we have

$$\frac{1}{a_2} > \frac{1}{a_3} > 1.$$

With $1-n > 0$ and $\frac{1}{a_2} - 1 > \frac{1}{a_3} - 1 > 0$, we get

$$\frac{C_1}{C_2} < 1.$$

So

$$\frac{1}{m_{123}(A)} < \frac{1}{m_{132}(A)},$$

and

$$m_{123}(A) > m_{132}(A).$$

Similarly, we can prove that

$$m_{312}(A) > m_{321}(A).$$

Note that $m_{132}(A) = m_{312}(A)$, so we have

$$m_{123}(A) > m_{132}(A) > m_{321}(A)$$

Thus, we proved that for three sources of evidence, if we sort them in increasing order $m_1(A) < m_2(A) < m_3(A)$ and combine them, the result m_{123} is the maximum combination result for the belief function m_A. Meanwhile, we proved that the decreasing order m_{321} results in the minimum combination result for the belief function m_A.

It is easy to derive from the above proof that for three sources of evidence, if $m_2(A) > m_1(A)$, then combining them in the order of $(m_3(A) \oplus m_1(A)) \oplus m_2(A) > (m_3(A) \oplus m_2(A)) \oplus m_1(A)$, no matter what value $m_3(A)$ has.

2. **Inductive Step.**

Based on the proof on three sources of evidence, we postulate that for k ($k \geq 3$) sources of evidence $m_1(A) \cdots m_k(A)$, if we sort them in the increasing order of $m_1(A) < m_2(A) \cdots < m_k(A)$ and combine them in this order, then we obtain the maximum combination result for $m(A)$. If we sort them in the decreasing order of $m_1(A) > m_2(A) \cdots > m_k(A)$ and combine them in this order, then we obtain the minimum combination result for $m(A)$.

Now we need to prove that if we have one new source of evidence $m_{k+1}(A)$, the inductive step holds.

For $k+1$ sources of evidence, we can sort them in increasing order, and obtain $m'_1(A) < m'_2(A) \cdots < m'_k(A) < m'_{k+1}(A)$. Now suppose we can use a combination order to integrate $k+1$ sources of evidence, which results in the maximum combination result for $m'_{1 \cdots k+1}(A)$. There are two cases for this combination order: either $m'_{k+1}(A)$ is the last one $(k+1)^{th}$ to be combined, or $m'_{k+1}(A)$ is *not* the last one $(k+1)^{th}$ to be combined.

In the first case that $m'_{k+1}(A)$ is the last one $(k+1)^{th}$ to be combined, we know that the first k sources of evidence should be combined in the increasing order of $m'_1(A) < m'_2(A) \cdots < m'_k(A)$ to get the maximum result for $m'_{1 \ldots k}(A)$. So the combination order for $k+1$ sources of evidence should be the same as the increasing order of $m'_1(A) < m'_2(A) \cdots < m'_k(A) < m'_{k+1}(A)$, to achieve the maximum combination result for $m'_{1 \cdots k+1}(A)$.

In the second case that $m'_{k+1}(A)$ is *not* the last one $(k+1)^{th}$ to be combined, then the position of $m'_{k+1}(A)$ in the combination should be within $[1,k]$, and there is a source of evidence $m'_i(A)$ with i within $[1,k]$ should be the last to be combined. In this case, for the first k sources of evidence to be combined, $m'_{k+1}(A)$ is the largest, so it should occupy the k^{th} position during the combination; while $m'_i(A)$ occupies the $(k+1)^{th}$ position since it is the last one to be combined. After we combine the first $k-1$ sources of evidence, we have $m'_{1 \cdots k-1}(A)$, $m'_{k+1}(A)$, and $m'_i(A)$ to be combined in such order, which would lead to the maximum combination result for $m'_{1 \cdots k+1}(A)$. However, we know from Step 1 that, if $m'_{k+1}(A) > m'_i(A)$, then $(m'_{1 \cdots k-1}(A) \oplus m'_{k+1}(A)) \oplus m'_i(A) < (m'_{1 \cdots k-1}(A) \oplus m'_i(A)) \oplus m'_{k+1}(A)$. Therefore, the second case will never lead to the maximum combination result for $m'_{1 \cdots k+1}(A)$.

Hence, we prove that for $k+1$ $(k \geq 3)$ sources of evidence $m_1(A) \cdots m_{k+1}(A)$, if we sort them in the increasing order of $m_1(A) < m_2(A) \cdots < m_{k+1}(A)$ and combine them in this order, we obtain the maximum combination result for the belief function $m(A)$. Similarly, we can prove that if we sort them in the decreasing order of $m_1(A) > m_2(A) \cdots > m_{k+1}(A)$ and combine them in this order, we obtain the minimum combination result for the belief function $m(A)$. □

5.4 Summary

In this chapter, we proposed a general methodology for information fusion, which can take into account the correlation and importance of each source of information during the fusion. It is based on the Murphy's rule of combination and fuzzy logic. This framework is flexible and is able to cope with situations where different sources of information carry the same weight or different weight, correlated or uncorrelated. This general methodology is an extension to the D-S framework and has a great potential to be applied in many application domains. We will introduce two applications of the proposed methodology in the next chapter. One is a in realtime intelligent flight control system; the other is in software reliability prediction.

We also provided an algorithm and a proof for the upper and lower bound of the belief function of combination results for the Murphy's rule, when the contextual weighting parameter remains constant for each source of information (meaning that each source of information carries the same weight during the fusion). To the best of our knowledge, this is the first such algorithm for the Murphy's rule of combination. This theoretical aspect is applied in constructing an Online safety Monitor for adaptive intelligent flight control systems in Chapter 6.

Chapter 6

Software Engineering Applications of the Dempster-Shafer Information Fusion Methodology

Information fusion technologies have been widely applied in sensor fusion [128] [162] [114] [123] [156], event or object detection [49] [50] [56] [71] [120] [147], information retrieval [32] [97] [106] [151] [161], computation of the absolute orientation [61], classifier fusion [77] [84] [129], image analysis [27] [125] [131] [142], multimodal (multimedia) systems [15] [23] [155], and medical applications [43] [95] [155].

In Chapter 5, we proposed a methodology based on the Murphy's rule of combination and fuzzy logic to combine correlated information. It is a general methodology for combining evidence, either correlated or distinct. Our approach contains following steps: (1) deciding correlated sources of information; (2) calculating correlation coefficient between possibly correlated information; (3) ranking the importance of each source of information; (4) developing a fuzzy rule set for the contextual weighting parameter, based on the importance and correlation of each source of information; (5) applying the Murphy's rule to combine available information. We also provided an algorithm and proof for the upper and lower bound of combination results for the Murphy's rule, when the contextual weighting parameter remains constant for each source of information (meaning that each source of information carries the same weight during the fusion).

In this chapter, we apply the proposed information fusion methodology to two applications: (1) constructing an Online safety Monitor for adaptive intelligent flight control systems to detect abnormal events; (2) fusion of various software reliability prediction systems for a more precise prediction of software reliability. These two case studies are introduced in §6.1

and §6.2, respectively. §6.3 summarizes the chapter.

6.1 Constructing an Online Safety Monitor for Intelligent Flight Control Systems

In this case study, we apply the proposed methodology in an adaptive intelligent flight control system. Specifically, we construct an Online Safety Monitor to detect deviations of state that could lead to unsafe behavior.

6.1.1 Project Introduction

An adaptive flight control system that is capable of sensing its environment, processing information, reducing uncertainty, planning, generating and executing control actions is considered an Intelligent Flight Control System (IFCS). The goal of IFCS is to develop and evaluate flight control concepts that incorporate emerging algorithms and techniques to provide an extremely robust system capable of handling multiple accident and off-nominal flight scenarios. Adaptive control is the latest trend in the application of Neural Networks (NN) in realtime automation. Figure 6.1 shows the architectural overview of an IFCS consisting of an Online Learning Neural Network (OLNN) that accounts for dramatic changes of the aircraft exceeding robustness limits [159].

The performance of the Online Neural Network within the dotted box in Figure 6.1 is monitored by an Online Stability Monitor. It is important to understand if the neural network adaptation converges, meaning that learning trajectories converge to a stationary state. In other words, if the Online Neural Network encounters unusual data patterns that force the state of the system to deviate *away* from the its current pattern, monitors will ensure that it always converges back to a stable equilibrium within a *finite* amount of time. Figure 6.2 shows the behavior of an Online Stability Monitor in an off-nominal (failure) mode. In contrast, Figure 6.3 shows the behavior of the Online Stability Monitor in a normal mode.

Currently, there are four Online Stability Monitors used simultaneously in the project, monitoring different parameters. The goal of this project is to integrate all the Online Stability Monitors into an *Online Safety Monitor*, which can detect safety violations. The objective of the Online Safety Monitor is to detect deviations of state that could lead to unstable behavior by monitoring the behavior of all Online Stability Monitors [159]. We apply the proposed D-S fusion methodology to this application.

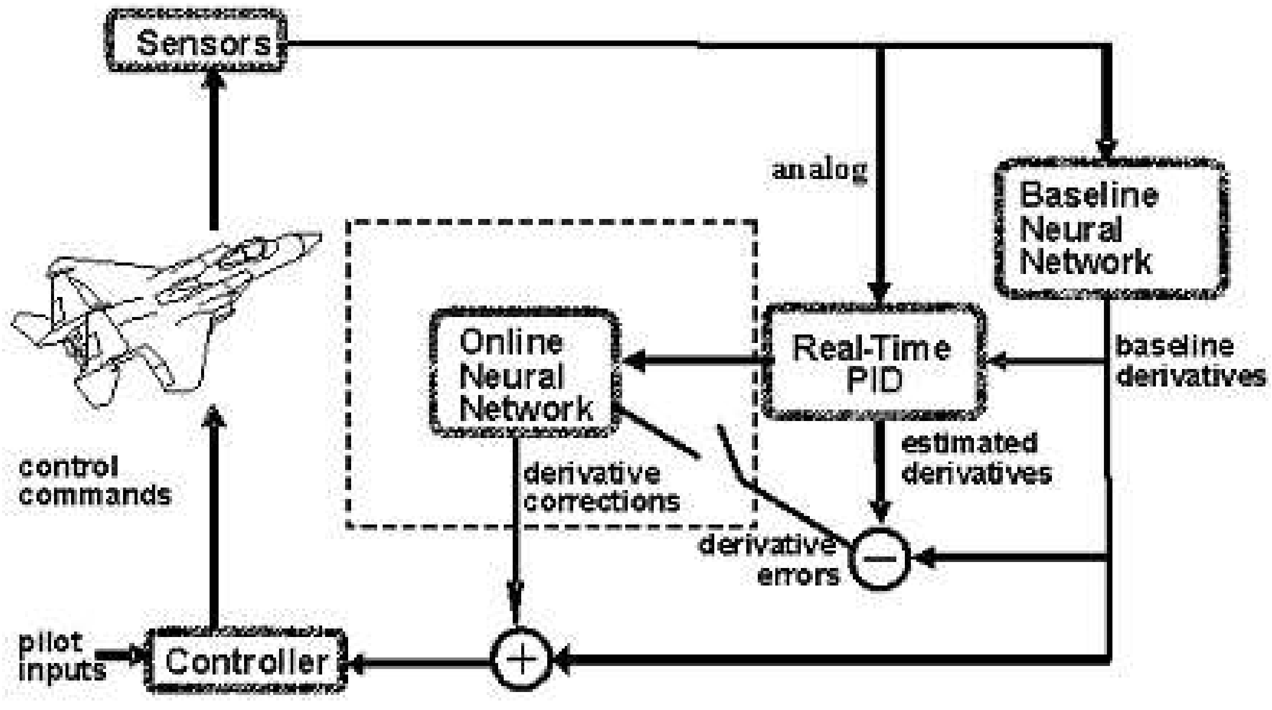


Figure 6.1: IFCS Architecture

6.1.2 Experimental Procedure

In our experiment, we first calculated the correlation coefficient between each pair of Online Stability Monitors. Based on the correlation coefficient and domain knowledge, we developed an algorithm to construct the Online Safety Monitor based on the Murphy's rule of combination. The experiments were performed on nine simulations that simulate seven failure modes and two normal modes, reflecting different flight data patterns. Each simulation has 200 data points, representing 200 time frames. The experimental procedure is detailed as below.

Calculating Correlation Coefficient

The correlation coefficient between each pair of Online Stability Monitors is calculated from the seven simulations for failure modes. As mentioned before, each Online Stability Monitor observes the learning of the Online Neural Network. The behavior of each neural network is consistent in normal modes and stable equilibria. Therefore, each Online Stability Monitor

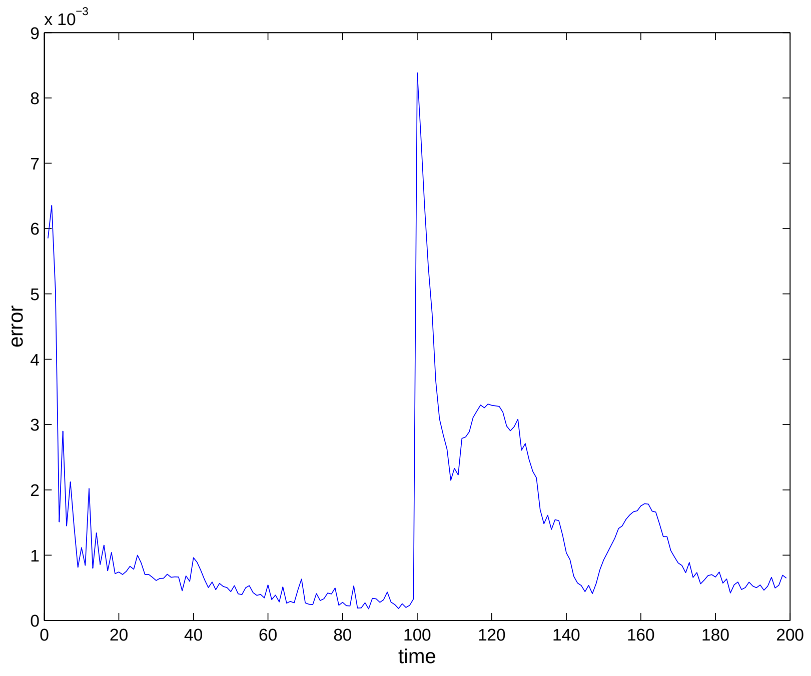


Figure 6.2: Online Stability Monitor in failure mode (the failure occurs at time 100)

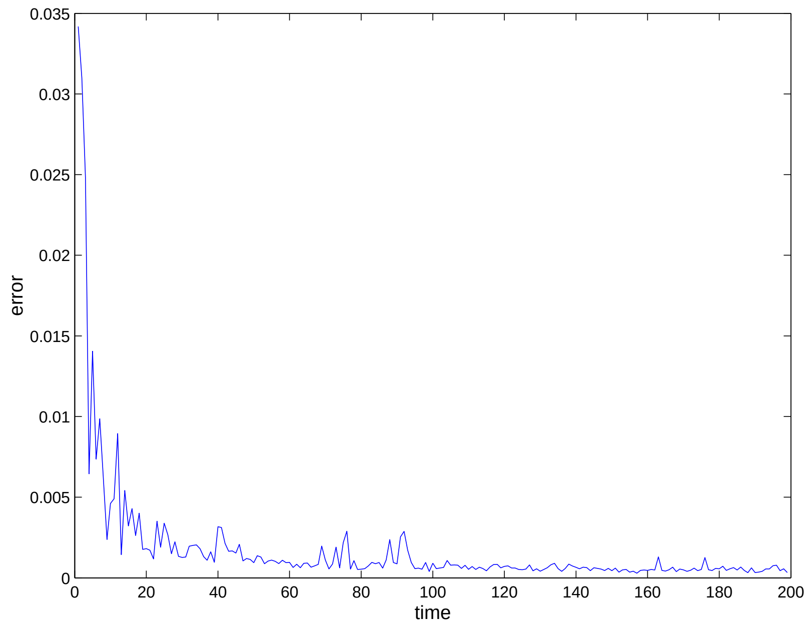


Figure 6.3: Online Stability Monitor in normal mode

is correlated with others in these states. Our focus is the behavior of each Online Stability Monitor when it is encountering a failure (before it converges to a stable equilibrium) as observed in the flight data patterns. Thus, the time period of reacting to a failure in each Online Stability Monitor is taken for the computation of correlation coefficients. As such, in each failure simulation data set, the period starting from 10 time frames before a failure occurrence to 80 time frames after the failure occurrence is used in the computation of correlation coefficients.

For each data set, the correlation coefficient between each pair of Online Stability Monitors is greater than 0.99, indicating that all Online Stability Monitors are highly correlated with each other. The correlation coefficients in two surface failure modes and five control failure modes are listed in tables 6.1 through 6.7. The average correlation coefficient between each pair of monitors in seven tables is computed as shown in Table 6.8. From Table 6.8, it can be observed that each pair of Online stability Monitors are highly correlated with each other with an average correlation coefficient greater than 0.99 in the failure modes.

Constructing Online Safety Monitor

We combine the four Online Stability Monitors into an Online Safety Monitor by applying the proposed methodology illustrated in §5.3. In this case, each source of evidence is an Online Stability Monitor. The output of each Online Stability Monitor is an error measure by the corresponding neural network in time series. The output from different monitors have different scales. Therefore, in order to combine them, the output of each Online Stability Monitor is first normalized into a real number within the [0.0, 1.0] interval, by dividing the current error measure by the maximum error measure up to this time frame of this Online Stability Monitor. The Online Safety Monitor is constructed by combining the outputs from all Online Stability Monitors. In this case, the frame of discernment Θ is {*Error (E), Confidence (C)*}, with $m(E) + m(C) = 1$. In other words, we define the belief value of *Confidence*, $m(C)$, as $1 - m(E)$, where $m(E)$ is the normalized error output from the Online Stability Monitor. The output of the Online Safety Monitor is the confidence interval. The scheme for constructing the Online Safety Monitor is shown in Figure 6.4.

According to the methodology, the value of the n parameter in the Murphy's rule of combination (Equation 2.10) needs to be defined, such that each Online Stability Monitor is given its corresponding credit during the combination. The n value for each monitor is determined based on two factors: its correlation with other monitors, and its importance as a stability monitor. We know that all the Online Stability Monitors are highly correlated with each other from the previous computation. In addition, we know that all these Online

Table 6.1: Correlation Coefficients in Surface Failure Mode 1

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9994	0.9985	0.9997
Monitor 2			0.9994	0.9994
Monitor 3				0.9984
Monitor 4				

Table 6.2: Correlation Coefficients in Surface Failure Mode 2

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9989	0.9979	0.9998
Monitor 2			0.9996	0.9990
Monitor 3				0.9979
Monitor 4				

Table 6.3: Correlation Coefficients in Control Failure Mode 1

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9979	0.9961	0.9996
Monitor 2			0.9995	0.9981
Monitor 3				0.9965
Monitor 4				

Table 6.4: Correlation Coefficients in Control Failure Mode 2

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9992	0.9982	0.9996
Monitor 2			0.9994	0.9992
Monitor 3				0.9982
Monitor 4				

Table 6.5: Correlation Coefficients in Control Failure Mode 3

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9985	0.9970	0.9997
Monitor 2			0.9996	0.9984
Monitor 3				0.9970
Monitor 4				

Table 6.6: Correlation Coefficients in Control Failure Mode 4

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9983	0.9971	0.9997
Monitor 2			0.9995	0.9986
Monitor 3				0.9973
Monitor 4				

Table 6.7: Correlation Coefficients in Control Failure Mode 5

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9985	0.9978	0.9997
Monitor 2			0.9997	0.9987
Monitor 3				0.9980
Monitor 4				

Table 6.8: Averaged Correlation Coefficients in All Failure Modes

	Monitor 1	Monitor 2	Monitor 3	Monitor 4
Monitor 1		0.9987	0.9975	0.9997
Monitor 2			0.9995	0.9988
Monitor 3				0.9976
Monitor 4				

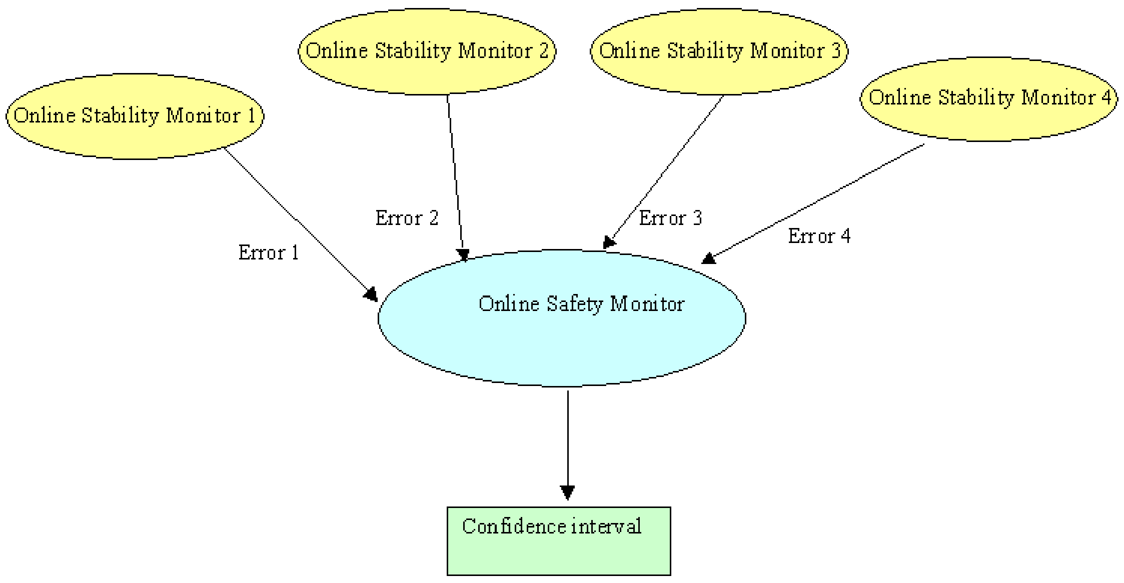

Online Stability Monitor 2
Online Stability Monitor 3
Online Stability Monitor 4
Online Stability Monitor 1
Error 2
Error 3
Error 4
Error 1
Online Safety Monitor
Confidence interval

Figure 6.4: Constructing an Online Safety Monitor

Stability Monitors are equally important based on domain knowledge. Therefore, when each Online Stability Monitor is taken into the combination, it is treated equally and thus $n = 0.5$.

The Murphy's rule of combination is non-commutative. Different order of combination has different result. As demonstrated in §5.4, in the case of constant contextual weighting parameter, when sources of evidence are sorted in increasing and decreasing order, we can get the upper and lower bound for the combination results, respectively. Confidence interval can be obtained from the upper and lower bound of the combination results, which will be the output from the Online Safety Monitor. The algorithm of constructing the Online Safety Monitor is described in Figure 6.5.

Each data set from the nine simulations is input to the Online Safety Monitor according to the algorithm in Figure 6.5. The outputs from the Online Safety Monitor are discussed in the following section.

```
Constructing the Online Safety Monitor
Begin
At each time frame
  for each Online Stability Monitor $i$, $i \; in \; 1 \cdots n$ ($n$ is the number of Online Stability Monitors)
      $Current\_Max\_Error_i = \max(Error_i \; up \; to \; this \; time \; frame)$;
      $m_i(E) = \frac{Error_i}{Current\_Max\_Error_i}$;
      $m_i(C) = 1 - m_i(E)$;
  sort $m_i(C)$ ($i \; in \; 1 \cdots n$) in order;
  Upper confidence $m'(C) \leftarrow$ combine $m_i(C)$ in increasing order;
  Lower confidence $m''(C) \leftarrow$ combine $m_i(C)$ in decreasing order;
  Confidence difference $= m''(C) - m'(C)$;
  output Upper confidence, Lower confidence, Confidence difference;
End
```

Figure 6.5: The Algorithm for Constructing the Online Safety Monitor

6.1.3 Results

We performed the above described experiments for nine flight simulations. The four Online Stability Monitors were used as inputs to the algorithm in Figure 6.5. The constructed Online Safety Monitor is able to discriminate failure modes from normal modes in all simulations. For instance, in one of the simulations for a failure mode, the output from the Online Safety Monitor is depicted in Figure 6.6. In contrast, in one of simulations for a normal mode, the output from the Online Safety Monitor is depicted in Figure 6.7.

The Online Safety Monitor generates output from the four Online Stability Monitors, which are constructed from the Online Neural Network. The error output of the neural network is high when it is in the learning phase. Our methodology can differentiate between learning phase and failure phase by monitoring the confidence difference. When the *confidence difference* is high, it indicates that the neural network is in its learning phase. When the *confidence difference* becomes close to zero, it indicates that the neural network is providing reliable results. In this case, if *confidence* remains high, it means that the system is in normal mode; otherwise, if the *confidence* suddenly decrease, the Online Safety Monitor should give a “red light” and issue a warning that the system is in a failure mode.

The Online Safety Monitor can differentiate between failure modes and normal modes, as well as failure modes and learning phase of the Online Stability Monitors by following rules:

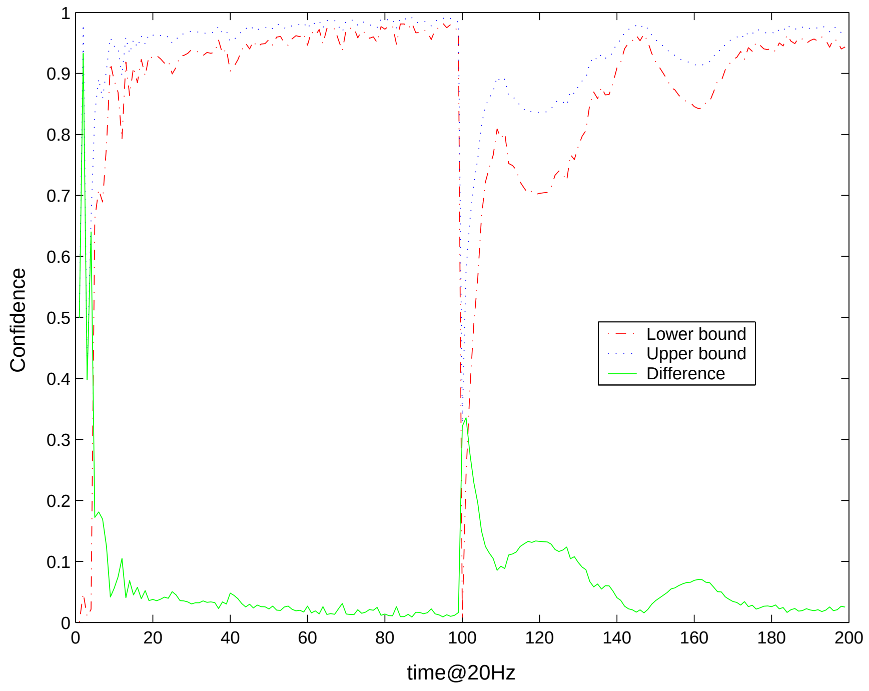


Figure 6.6: Online Safety Monitor in failure mode

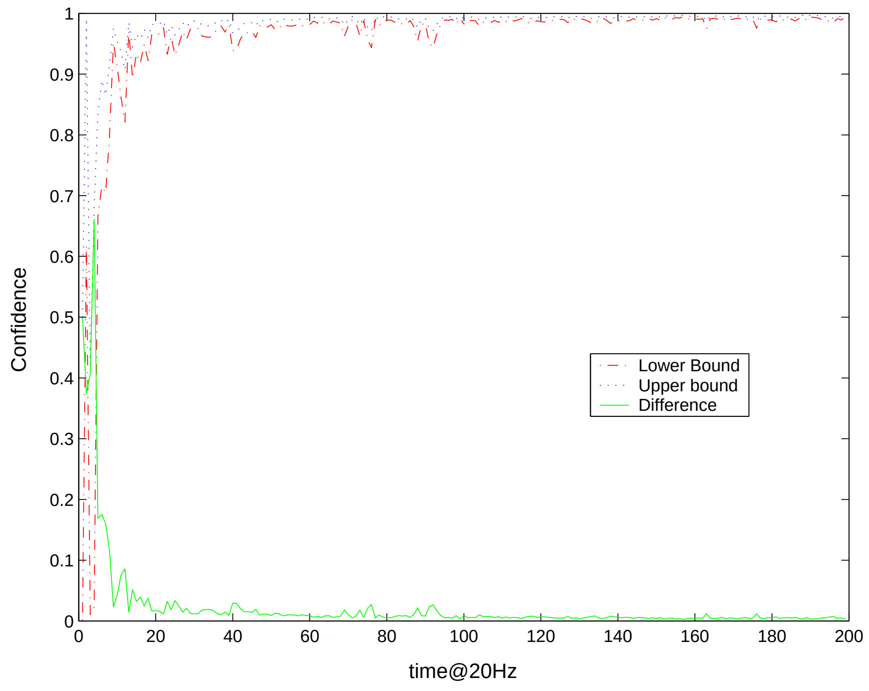


Figure 6.7: Online Safety Monitor in normal mode

- If *Confidence difference* is high, and *confidence* (lower bound) is low, then it is the learning phase of the Online Stability Monitors. In this case, the Online Safety Monitor issues a *yellow light* to represent the potentially unsafe system state;
- If *Confidence difference* remains low, and *confidence* (both upper bound and lower bound) remains high, then it is in the normal mode. In this case, the Online Safety Monitor issues a *green light* to represent a safe system state;
- If *Confidence difference* dramatically increases, and *confidence* (both upper bound and lower bound) dramatically decreases, then it is in failure mode. In this case, the Online Safety Monitor issues a *red light* to represent the unsafe system state.

6.1.4 Discussion

We applied the proposed methodology to the adaptive intelligent flight control system. We constructed an Online Safety Monitor by combining four Online Stability Monitors. In nine simulations for seven failure modes and two normal modes, the constructed Online Safety Monitor is able to issue a warning when the system is *actually* in a failure mode.

This study is the first attempt to apply the Murphy's rule of combination in adaptive intelligent control flight systems. We proposed and proved that, when all sources of evidence are treated equally, there is an upper bound and lower bound of the combination results obtained by the Murphy's rule of combination. This aspect is applied in the Online Safety Monitor to differentiate between failure modes and non-failure modes of the Online Stability Monitors.

6.2 Fusion of Software Reliability Prediction Systems

Software reliability is a quantitative measure of software quality. It is defined as a probability of failure free execution given a specific environment and a fixed time interval. Currently there are many software reliability estimation models aimed at predicting the reliability of software products. It remains a problem to integrate the results of various software reliability estimation methods, for a more precise prediction. There are two important statistical theories for evidence combination: the Bayesian theory and Dempster-Shafer (D-S) theory. The Bayesian theory needs a complete knowledge of the probability laws to combine evidence and perform prediction. However, we do not have enough data to come up with a solid prior or conditional probability as required by the Bayesian theory in many cases. On the other hand, the D-S theory is a general form of the Bayesian theory, as discussed in Chapter 2, and does not require any subjective prior or conditional probability.

Some software reliability estimation models are built from software quality measures [140] [89]. If two software reliability estimation models share some software quality measures, these two software reliability models are correlated. In this case, the results of the software reliability models are also correlated, and the combination of such results amounts to correlated evidence combination. As discussed in the previous chapter, the proposed methodology is able to combine evidence in general, either correlated or uncorrelated. In this section, we will introduce a case study where we apply the proposed D-S fusion methodology to integrating software reliability prediction systems, some of which are correlated with each other.

6.2.1 Project Introduction

Software reliability models built from process and product measures are very useful in prediction of software quality in early software life cycle. However, reliability prediction models based on process and product measures alone may not be sufficiently accurate [45] [141]. These predictions need *corroboration*. If predictions by several software reliability models can be combined for a more accurate prediction, we can assume a limited belief in the accuracy of this refined prediction. This refined prediction can be employed in a theoretical framework of software reliability corroboration for the certification testing, such that the main drawback of input domain reliability assessment models, the impractically large number of statistical tests, disappears [139]. The project overview is depicted in Figure 6.8. This project is aimed at making software certification of high assurance systems practical.

Currently, the approach to building Reliability prediction Systems (RPS) has been developed by the research group from the University of Maryland at College Park [140] [89]. In a study carried out for the U.S. Nuclear Regulatory Commission, 40 software engineering measures were ranked with respect to their ability to predict reliability. The theoretical framework of software reliability corroboration have been established by the research group from west Virginia University [34]. The unresolved problem in the project shown in Figure 6.8 is the middle part, which is to combine Reliability Prediction Systems (RPS) for a more accurate prediction. The main issue is that, if two RPSs share same Software Quality Measures (SQM) or SQMs (for building the RPSs) are correlated, these two RPSs are correlated with each other. In this case, combining correlated RPSs amounts to combining correlated evidence.

The application studied in the reliability prediction experiment is a simplified version

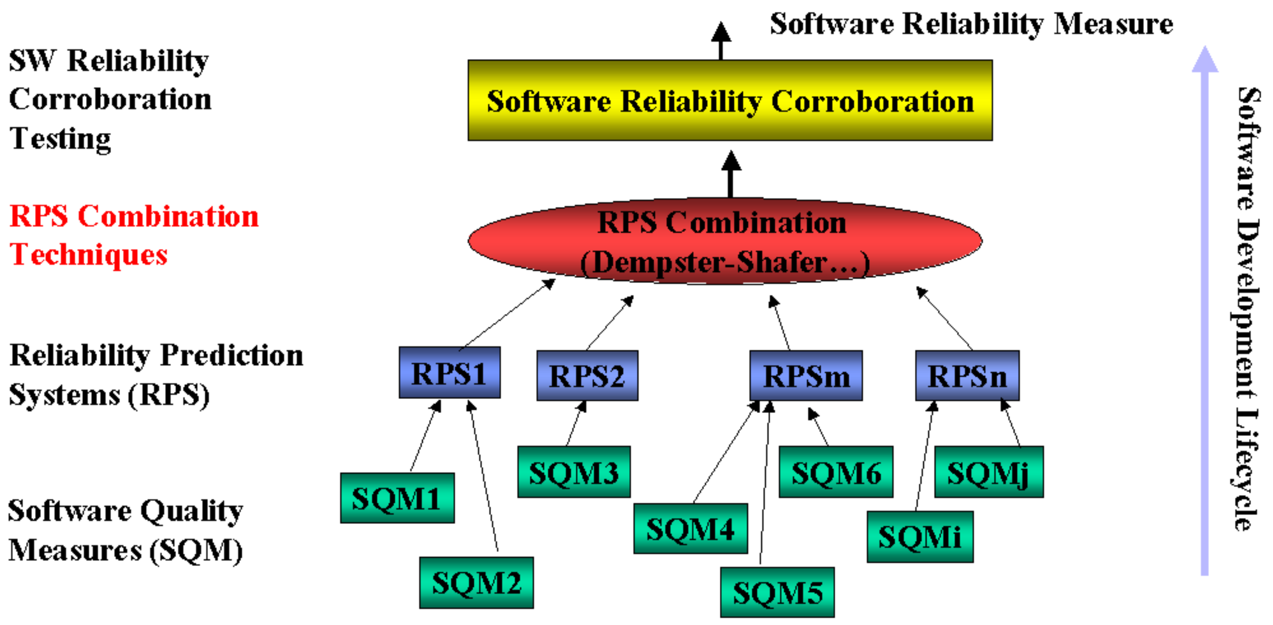

Software Reliability Measure
SW Reliability Corroboration Testing
Software Reliability Corroboration
RPS Combination Techniques
RPS Combination (Dempster-Shafer…)
Reliability Prediction Systems (RPS)
RPS1
RPS2
RPSm
RPSn
Software Quality Measures (SQM)
SQM1
SQM2
SQM3
SQM4
SQM5
SQM6
SQMi
SQMj
Software Development Lifecycle

Figure 6.8: Project Overview of Software Reliability Corroboration

of an automated personnel entry access system (gate) used to provide privileged access to rooms/buildings. The study involved the following Software Quality Measures (SQM): requirement traceability, function points, bugs per line of code (Gaffney estimate), fault density, and test coverage. Estimates of reliability were built based on these measures. The validation study was limited to the testing phase, and the estimates were used to predict reliability in operation. The ranking of each RPS and the corresponding estimate are listed in Table 6.9. The actual failure rate obtained from testing is 0.09.

Table 6.9: The Value of Relevance to Reliability and Estimated Reliability

Meaure	Relevance to Reliability r	Predicted Failure Probability P_f
Code Defect Density	0.85	0.078
Test Coverage	0.83	0.092
Requirements Traceability	0.45	0.078
Function Point Analysis	0.00	0.0020
Bugs Per Lines of Code	0.00	0.000028

Weighted sum based on Relevance to Reliability r was first used to combine PRSs. It is designed as follows:

$$P_f = \frac{\sum_{i=1}^{n} r_i (P_f)_i}{\sum_{i=1}^{n} r_i} = \frac{0.85 \times 0.078 + 0.83 \times 0.092 + 0.45 \times 0.078}{0.85 + 0.83 + 0.45} = 0.084$$

This method takes into consideration of relevance to reliability of each RPS. However, it cannot handle the correlation among the RPSs. We apply the proposed methodology in §5.3 in this application to combine several RPSs based on their Relevance to reliability and possible correlation with each other. The experimental procedure is explained in the next section.

6.2.2 Experimental Steps

We apply the proposed methodology to combining Reliability Prediction Systems (RPSs). According to the methodology, we need to first decide which RPSs are mutually correlated. Secondly, we calculate the correlation coefficient between the correlated RPSs. Thirdly, we need to develop a fuzzy rule set for the belief revision function of the Murphy's rule of combination. Finally, the Murphy's rule of combination is applied to combining RPSs. The detailed experiments are illustrated as follows.

Deciding Correlated RPSs

According to our general methodology for evidence combination, we need to first decide which sources of evidence are correlated. In this case, the sources of evidence are Reliability Prediction Systems (RPSs) build on code defect density, test coverage, and requirements traceability. RPSs based on function point and bugs per lines of code are ignored, because their relevance to reliability is estimated to be close to zero.

Defect density is defined as the number of defects unresolved at the testing stage divided by the number of lines of code in the software. Defects detected from requirements inspection, design inspection, and source code inspection will be used to predict software reliability. RPS based on defect density consists of three steps: (1) construction of a finite state machine representing the user's requirements and embedding user's profile information. (2) mapping of the defects to this model and actual tagging of the states and transitions. (3) execution of the model to evaluate the impact of the defects [140].

Requirements traceability identifies requirements implemented in source code that are either missing from, or in addition to, the original requirements. Each missing function or additional function in the requirements is a defect. As such, a finite machine model representing the requirements can be used to predict the failure probability. The approach is the same as that in defect density. However, the defects mapped into the finite state machine are not identical in these two RPSs. Therefore, the RPSs based on requirements traceability and defect density are not correlated.

Test coverage in [140] is defined as statement coverage. The RPS based on test coverage uses the information from the RPS based on defect density. Therefore, the RPSs based on test coverage and defect density are correlated. Detailed explanation is given in the following section.

Calculating Correlation Coefficient

The second step of the methodology is to calculate the correlation coefficient between possibly correlated evidence. In this case study, we need to obtain the correlation coefficient between two RPSs constructed from Test Coverage and Defect Density.

First, predicted software reliability $P_s(n)$ of RPS constructed from Test Coverage is calculated as (as described in $2.4.2):

$$P_s(n) = e^{-\frac{K}{T_L}N\tau n} \tag{6.1}$$

where K is fault exposure ratio during the n^{th} demand; T_L is the linear execution time; N is the number of defects remaining in the software; τ is the average execution time per demand.

The number of remaining defects N can be obtained by:

$$N = \frac{N^0}{C^0} \tag{6.2}$$

where N^0 is the number of defects found by the test cases; C^0 is the defect coverage.

The relationship between the defect coverage C^0 and code coverage is [99]:

$$C^0 = a_0 \ln[1 + a_1(e^{a_2 C_1} - 1)] \tag{6.3}$$

where C_1 is the code coverage achieved by the test cases; a_0, a_1, a_2 are coefficients. The coefficients can be estimated from data, or from previous projects. In [140], two sets of coefficients a_0, a_1, a_2 (calculated from the Data Set 3 and Data Set 4 in [99]) are used to calculate defect coverage. Statement coverage during the testing of PACS is 94.6%. The defect coverage and the total number of defects remaining in PACS given the parameters in [99] are shown in Table 6.10.

Suppose the fault exposure ratio K is constant for all defects. Then the additional, unknown defects determined by Equation 6.2 have the same fault exposure ratio K as the known defects. The fault exposure ratio K is calculated from the RPS based on defect density in [140].

Table 6.10: Defects Remaining vs. Test Coverage [140]

	Data Set 3 [99]	Data Set 4 [99]
Statement Coverage C_1	94.6%	94.6%
Defect Coverage C^0	84.9%	81.1%
Number of Defects Found through Test	N^0	N^0
Number of Remaining Defects N	$\frac{N^0}{84.9\%}$	$\frac{N^0}{81.1\%}$

From the RPS based on defect density, when $n = 1$, N^0 is the number of discovered defects, and $P'_s = 1 - P'_f$, from Equation 6.1 we can obtain K. P'_f is the probability of failure predicted by the RPS based on defect density when N^0 defects are detected [140]:

$$K = \frac{-\ln(1-P'_f)}{\tau * N^0} T_L \tag{6.4}$$

The fault exposure ratio K is used by the RPS based on test coverage to predict the reliability. By substituting Equation 6.4 into Equation 6.1, the predicted software reliability by the RPS based on test coverage is:

$$P_s = e^{\frac{N}{N^0}\ln(1-P'_f)} = P_s'^{\frac{N}{N^0}} = P_s'^{\frac{N^0}{N^0 C^0}} = P_s'^{\frac{1}{C^0}} \tag{6.5}$$

P'_s is the predicted reliability by the RPS based on defect density, when N^0 defects are detected during the inspection process.

Strictly speaking, the number of remaining defects is an integer. Thus, Equation 6.2 should be:

$$N = ceiling(\frac{N^0}{C^0})$$

As a result, Equation 6.5 should be:

$$P_s = (P'_s)^{ceiling(\frac{N^0}{C^0})\frac{1}{N^0}} \tag{6.6}$$

There are two scenarios for the defects detected in Defect Density and Test Coverage:

- If the same inspection process is used, the defects detected for Test Coverage and Defect Density are the same. In this case, P'_s is the predicted reliability by the RPS based on Defect Density. If we ignore the *ceiling* effect of the remaining defects, based on Equation 6.5, we have

$$P_{s(Test\ Coverage)} = (P_{s(Defect\ Density)})^{\frac{1}{C^0}}$$

It indicates that if the same defects are used for the defect density measure and the test coverage measure, the RPSs based on these two measures are highly correlated. In fact, we can derive one from the other. Based on the testing information collected for PACS, the correlation coefficient between these two RPSs can be calculated from the simulation of possible $P_{s(Defect\ Density)}$ values. Two different defect coverage values from Table 6.10 are used in the computation. The correlation coefficient between the RPSs based on Test Coverage and Defect Density is computed for each defect coverage value. The results listed in Table 6.11 are based on 1,000 simulations.

Table 6.11: Correlation between RPSs Based on Test Coverage and Defect Density (1)

Defect Coverage	$C^0 = 84.9\%$	$C^0 = 81.1\%$
Number of Simulations	1000	1000
Correlation coefficient	0.9985	0.9975

If we want to be more accurate by taking into account the ceiling effect of the remaining defects, based on Equation 6.6, we have

$$P_{s(Test\ Coverage)} = (P_{s(Defect\ Density)})^{ceiling(\frac{N^0}{C^0})\frac{1}{N^0}}$$

In this case, we need to simulate defects detected during the inspection process. During the simulation, defects are randomly injected to the finite state machine used in the Defect Density approach. The reliability of PACS is predicted correspondingly by the Defect Density approach. The information is used to predict the reliability by the RPS based on Test Coverage. Based on 1,000 simulations, we obtain the correlation coefficient between the RPSs constructed from Test Coverage and Defect Density as in Table 6.12.

Table 6.12: Correlation between RPSs Based on Test Coverage and Defect Density (2)

Defect Coverage	$C^0 = 84.9\%$	$C^0 = 81.1\%$
Simulations	1000	1000
Correlation coefficient	0.9997	0.9995

- If different inspection processes are used for Test Coverage and Defect Density, we can simulate the defects detected and the corresponding predicted reliability for this scenario. Normally, a more rigorous inspection is employed for Defect Density than Test Coverage. Therefore, it is reasonable to assume that there are more defects detected in Defect Density than Test Coverage [88].

  Two different inspection processes are simulated, one for Defect Density, the other for Test Coverage. N' and N^0 represent the number of detected defects in Defect Density and Test Coverage, respectively. By assuming that the number of defects in Defect Density is more than that in Test Coverage, we have $N' > N^0$. During the simulation, N' and N^0 number of defects are randomly injected to the finite state machine used in the Defect Density approach, separately. The reliability of PACS is obtained correspondingly by using the Defect Density approach for each inspection process. The predicted reliability corresponding to N' detected defects is the prediction result by the RPS based on Defect Density, represented by $P_{s(Defect\ Density)}$; while the predicted reliability corresponding to N^0 detected defects, represented by P'_s, is used to predict the reliability by the RPS based on Test Coverage, $P_{s(Test\ Coverage)}$, by using Equation 6.6:

$$P_{s(Test\ Coverage)} = (P'_s)^{ceiling(\frac{N^0}{C^0})\frac{1}{N^0}}$$

  The correlation coefficient between the RPSs based on Defect Density and Test Coverage, namely $P_{s(Defect\ Density)}$ and $P_{s(Test\ Coverage)}$, is calculated based on the simulations. The results are listed in Table 6.13.

  Table 6.13: Correlation between RPSs Based on Test Coverage and Defect Density (3)

Defect Coverage	$C^0 = 84.9\%$	$C^0 = 81.1\%$
$\min(N' - N^0)$	1	1
$\max(N' - N^0)$	8	8
average$(N' - N^0)$	4 (3.60)	4 (3.60)
Number of simulations	989	989
Correlation coefficient	0.9953	0.9951

Based on above results, we can conclude that the RPSs based on Defect Density and Test Coverage are highly correlated with each other; while RPS based on Requirement Traceability is uncorrelated with these two RPSs.

Developing a Fuzzy Rule Set

The next step in the methodology is to develop a fuzzy rule set for the belief revision function of the Murphy's rule of combination (Equation 2.9). More specifically, we need to define the value for the contextual weighting parameter, n, in Equation 2.10. In this case study, the value of n depends on each RPS's relevance to reliability r and whether it is correlated with other RPSs to be combined. The fuzzy rules are as follows:

- If relevance to reliability r is high, and uncorrelated, then $n > 0.5$
- If relevance to reliability r is high and correlated, then $n < 0.5$
- If relevance to reliability r is low, then $n < 0.5$

A more refined fuzzy rule set is developed. The applicable rules to this case study are listed in Table 6.14.

Table 6.14: Fuzzy Rules for Combining PRSs

Rule No.	If	Then $n =$
1	$r \in [0.8, 1.0]$ and uncorrelated	0.55
2	$r \in [0.8, 1.0]$ and correlated	0.50
3	$r \in [0.6, 0.8)$ and uncorrelated	0.49
4	$r \in [0.6, 0.8)$ and correlated	0.45
5	$r \in [0.4, 0.6)$ and uncorrelated	0.48
6	$r \in [0.4, 0.6)$ and correlated	0.43
7	$r \in [0.0, 0.4)$	Not combining

Applying the Murphy's Rule of Combination

Based on Table 6.14, we can apply the Murphy's rule of combination to fusing Reliability Prediction Systems (RPS). In this case, the fame of discernment Θ is $\{$*reliability (R), probability of failure (F)*$\}$. The RPSs based on Defect Density and Test Coverage are correlated and have close value of relevance to reliability. The Rule No. 1 from Table 6.14 is applicable to combining these two RPSs. The integration mapping of these two RPSs is shown in Table 6.15.

According to the Murphy's rule of combination (Equation 2.9 and 2.10), These two RPSs can be combined as:

Table 6.15: Integration mapping of RPSs (1)

m_2/m_1	$\{R\}$ (0.912)	$\{F\}$ (0.078)
$\{R\}$ (0.908)	$\{R\}$ (0.828)	$\{\emptyset\}$ (0.0708)
$\{F\}$ (0.092)	$\{\emptyset\}$ (0.0839)	$\{F\}$ (0.00718)

$$m(R)' = \frac{0.828^{0.5}}{0.828^{0.5} + 0.00718^{0.5}} = 0.915$$
$$m(F)' = \frac{0.00718^{0.5}}{0.828^{0.5} + 0.00718^{0.5}} = 0.085$$

The RPS based on Requirements Traceability is taken into the combination following the Rule No. 5. The integration mapping of this combination is shown in Table 6.16.

Table 6.16: Integration mapping of RPSs (2)

m_3/m'	$\{R\}$ (0.915)	$\{F\}$ (0.085)
$\{R\}$ (0.912)	$\{R\}$ (0.834)	$\{\emptyset\}$ (0.0775)
$\{F\}$ (0.078)	$\{\emptyset\}$ (0.0714)	$\{F\}$ (0.00663)

The RPS based on Requirements Traceability is combined according to the Murphy's rule as:

$$m(R)' = \frac{0.834^{0.48}}{0.834^{0.48} + 0.00663^{0.48}} = 0.911$$
$$m(F)' = \frac{0.00663^{0.48}}{0.834^{0.48} + 0.00663^{0.48}} = 0.089$$

Hence, the predicted reliability of PACS is 0.911, and the probability of failure is 0.089, after we combine the RPSs by the proposed methodology.

6.2.3 Evaluation

We applied the proposed methodology to combining Reliability Prediction Systems (RPS). As shown in Table 6.17, our prediction is very precise compared with the actual reliability of PACS. It is more accurate than the result given by the weighted sum method. What's more, as a combination technique, the weighted sum method cannot take into account the correlation among RPSs; while the proposed methodology can evaluate the credit given to each RPS in the combination based on its relevance to reliability prediction and its correlation with other RPSs.

Table 6.17: Evaluation of Our Result

	Reliability	Failure Probability
Actual Result	0.910	0.090
Weighted Sum	0.916	0.084
Our Result	0.911	0.089

6.2.4 Discussion

Currently there are many indirect software reliability prediction methods available. Some are highly relevant to reliability prediction, while others are not. In addition, the results of some reliability prediction methods are correlated with each other. It remains a problem to integrate available reliability prediction methods for a more precise prediction, by taking into account the predictive power of each method and the correlation effect during the combination. Current combination techniques, for instance weighted sum, only considers the relevance to reliability of each method, while the correlation effect is merely ignored. To the best of our knowledge, this study is the first attempt to integrate reliability prediction methods by evaluating the credit given to each method based on its predictive power and its correlation with other methods.

The proposed methodology is based on the Murphy's rule of combination and fuzzy rule sets. When applied to software reliability prediction, this general methodology proves to be feasible and yields accurate results. We believe that the proposed methodology provides a flexible and robust framework for combining available software reliability prediction methods for a more precise prediction. It fills the gap in the software reliability corroboration projet, aiming at making software certification of high assurance systems practical.

6.3 Summary

In this chapter, we applied the proposed information fusion methodology in two areas: a realtime intelligent flight control system and software reliability prediction. In the first case study, the constructed Online Safety Monitor can accurately detect failure modes in the flight control system. In the second case study, the predicted software reliability by the proposed methodology is higher than that by the weighted sum method.

The proposed methodology is proved applicable to both time-series applications and point estimate applications. This framework is flexible and is able to cope with situations where different sources of information carry the same weight or different weight. This general methodology is an extension to the D-S framework and has great potential to be applied in many other application domains.

Chapter 7

Contributions and Future Work

7.1 Contributions

Good statistical models are vital to provide accurate predictions in various application domains. The Bayesian theory and Dempster-Shafer (D-S) theory are two important statistical theories for probability reasoning. However, both the Bayesian theory and the D-S theory cope with independent evidence. This assumption of "evidence independence" is restrictive and unrealistic in many applications, for instance when sources of evidence are correlated with each other. In addition, the Bayesian theory is not robust enough to cope with incomplete information. Rooted from the Bayesian theory, the Bayesian Belief Networks also suffer from such limitations.

Our research has solved two open problems in this research area. First of all, we developed a novel induction algorithm for the Dempster-Shafer networks, and a probability reasoning methodology based on the D-S networks. The proposed methodology is automatic, objective, flexible and dynamic, such that it overcomes the limitations of the Bayesian Belief Networks which are subjective and difficult to build. In addition, we developed a general methodology that can combine evidence in general, including both correlated and uncorrelated evidence. To the best of our knowledge, this is the first such framework to combine evidence and take correlation into account. This general framework is an extension to Dempster-Shafer theory. We also provided an algorithm and a proof for the upper and lower bound of the combination belief function, which has not been presented before. Both proposed methodologies prove to be applicable to complex real-life applications. We applied these two methodologies to software quality and reliability prediction.

The first proposed methodology, based on the Dempster-Shafer (D-S) belief networks, is a novel contribution to the theoretical framework of software quality prediction. The proposed methodology is meaningful for real-world applications in software quality prediction. Firstly, it can be tuned to meet different real-wold optimization criteria. Secondly, we illustrated a set of experiments of case-based reasoning performed on existing data sets. The practical meaning is that we can reuse the information from a similar project for early prediction in the project currently under development. Thirdly, the prediction accuracy of our proposed methodology is higher than that achieved by logistic regression, discriminant analysis, random forests, as well as the algorithms in two machine learning software packages, See5 and WEKA. The difference in the performance of the proposed methodology over other methods is statistically significant. This general framework can be applied to other research areas.

The second proposed methodology focuses on information fusion, which can combine both correlated and uncorrelated evidence. This framework is based on the Murphy's rule of combination and fuzzy logic. We also provided an algorithm and a proof for the upper and lower bound of the belief function of the combination results for the Murphy's rule, when each source of information carries the same weight during the combination. The proposed methodology was applied in a realtime intelligent flight control system. The Online Safety Monitor constructed based on this methodology can accurately detect unsafe behavior in the flight pattern data. The proposed methodology was also applied to the fusion of various software reliability prediction systems for a more precise prediction of software reliability. The prediction result is more accurate than the previously used weighted sum method. This general framework can be applied to combining evidence for prediction in many other research areas.

7.2 Future Work

It is a common practice to use categorized data sets by belief networks such as the Bayesian networks [45]. In our study, the D-S networks deal with discrete data sets. The case studies in Chapter 4 discretize continuous data sets into binary ones. A future research direction for improving Dempster-Shafer belief networks is to generalize important implication rules relating two multichotomous variables. Therefore, we can discretize data sets into multinomial, rather than binary, data sets. It is worth mentioning that our induction algorithm based on prediction logic makes this potential improvement possible, while the previous induction algorithm by Liu *et al.* [94], based on binomial distribution, can only deal with binary data sets.

Our methodology based on Dempster-Shafer networks is distinguished by its ability of tuning to meet different requirements. We observed that the sequence of the predictors taken into the D-S networks has effect on optimal network inference, which is consistent with the observation presented in [96]. We found that the first four or five predictors have the greatest impact on the prediction accuracy in our study. Currently, we do not have an algorithm to identify the "magic" sequence. Therefore, another future research direction is to develop such an algorithm for optimal inference on the D-S networks. This problem is defined as NP-hard [96]. One possible solution is to explore the relationship between the sequence of the predictors and the entropy (the measure of uncertainty) of the inducted D-S network. If each time the predictor taken into the D-S network is the one that is most likely to reduce the entropy of the entire network, the algorithm to decide the sequence of the predictors is then an optimization algorithm discussed in [96].

In the study of software quality prediction, the software metrics collected in KC2 and JM1 projects are mainly the McCabe Metrics and the Halstead Metrics. Many other software metrics such as process metrics and design metrics were proved important in software quality prediction [135], so we hope to improve the performance of our methodology if information of such metrics is available in a future study.

We applied two proposed methodologies to software quality and reliability prediction, as well as in a realtime intelligent flight control system for unsafe event detection. We would like to apply these two general frameworks to other research areas in future studies.

Bibliography

[1] http://www.agena.co.uk/bbn_article/bbns.html, 2002

[2] http://www.dcs.qmul.ac.uk/norman/all_publications.htm, 2002.

[3] http://www.r-project.org, 2004.

[4] http://www.stat.berkeley.edu/users/breiman/RandomForests, 2004.

[5] http://www.sas.com, 2004.

[6] http://www.rulequest.com/see5-info.html, 2004.

[7] A. A. Abdalla-Ghaly and a. B. L. P. Y. Chan, "Evaluation of competing reliability predictions," *IEEE Transactions on Software Engineering*, pp. 950-967, 1986.

[8] P. E. Ammann, S. S. Brilliant, and J. C. Knight, "The Effect of Imperfect Error Detection on Reliability Assessment via Life Testing", *IEEE Trans. Software Eng.*, Vol. 20, No. 2, Feb. 1994.

[9] D. Azar, S. Bouktif, B. K*é*gl, H. Sahraoui, D. Precup, "Combining And Adapting Software Quality Predictive Models By Genetic Algorithms", *Proc. 17th IEEE International Conference on Automated Software Engineering (ASE2002)*, p285, 2002.

[10] V. R. Basili, L. C. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators", *IEEE Trans. Software Eng.* 22(10):751-761, Oct. 1996.

[11] S. Bhattacharya, A. Onoma, and F. B. Bastani,"High-Assurance System", *Communications of ACM*, Vol. 40, No. 1,pp.67, Jan. 1997.

[12] J. M. Bieman and B. K. Kang, "Measuring Design-Level Cohesion", *IEEE Trans. Software Eng.*, Vol. 24, No. 2, Feb. 1998.

[13] P. G. Bishop and R. E. Bloomfield, "A Conservative Theory for Long-Term Reliability Growth Predictions", *IEEE Trans. Reliability*, vol. 45, no. 4, pp 550-560, Dec. 1996.

[14] P. G. Bishop and R. E. Bloomfield, "Worst Case Reliability Predication Based on a Prior Estimate of Residual Defects", *Proc.* 13*th* *Int'l Symposium on Software Reliability Engineering*, 2002.

[15] M. M. Blattner and E. P. Glinert, "Multimodal Integration", *IEEE Multimedia*, pp14-24, 1996.

[16] L. Breiman, "Random Forests", *Machine Learning*, Vol. 45, pp5–32, 2001.

[17] L. C. Briand, V. R. Basili, and C. J. Hetmanski, "Developing Interpretable Models with Optimaized Set Reduction for Identifying High-Risk Software Components", *IEEE Trans. Software Eng.* 19(11):1028-1044, Nov. 1993.

[18] L. C. Briand, S. Morasca, and V. R. Basili, "Defining and Validating Measures for Objected-Based High-Level Design", *IEEE Trans. Software Eng.*, Vol. 25, No. 5, Sep./Oct. 1999.

[19] L. C. Briand and D. Pfahl, "Using Simulation for Assessing the Real Impact of Test Coverage on Defect Coverage", *IEEE Trans. Reliability*, Vol 49, No. 1, Mar. 2000.

[20] S. Brocklehurst, P. Y. Chan, B. Littlewood and J. Snell, "Recalibrating Software Reliability Models", *IEEE Trans. Software Eng.*, Vol. 16, No. 4, Apr. 1990.

[21] R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software", *IEEE Trans. Software Eng.*, Vol. 19, No. 1, pp. 3-12, Jan. 1993.

[22] H. Buxton and S. Gong, "Advanced Visual Surveillance using Bayesian Networks", *International Conference on Computer Vision*, Cambridge, Massachusetts, June, 1995. http://citeseer.ist.psu.edu/buxton95advanced.html

[23] S.K. Chang and T. Znati, "Adlet: An Active Document Abstraction for Multimedia Information Fusion", *IEEE Trans. Knowledge and Data Engineering*, Vol. 13, No. 1, pp112-123, Jan./Feb. 2001.

[24] T. Chávez, "A Decision-Analytic Stopping Rule for Validation of Commercial Software Systems", *IEEE Trans. Software Eng.*, Vol. 26, No. 9, Sep. 2000.

[25] S. Chen and S. Mills, "A Binary Markov Process Model for Random Testing", *IEEE Trans. Software Eng.*, Vol. 22, No. 3, Mar. 1996.

[26] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", *IEEE Trans. Software Eng.*, Vol. 24, No. 8, Aug. 1998.

[27] C.C. Chu and J. K. Aggarwal, "The Integration of Image Segmentation Maps Using Region and Edge Information", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 15, No. 12, pp1241-1252, Dec. 1993.

[28] S. Chulani, "Bayesian Analysis of Software Costs and Quality Models", *Dissertation*, University of Southern California, 1999. http://sunset.usc.edu/publications/dissertations/SChulani.pdf

[29] V. Cortellessa, H. Singh, B. Cukic, "Early reliability assessment of UML based software models", *Proc. the Third International wWorkshop on Software and Performance*, 2002.

[30] V. Coupe, L. van der Gaag and J. Habbema, "Sensitivity analysis: an aid for belief-network quantification", *Knowledge Engineering Review*, vol. 15, pp. 1–18, 2000. http://citeseer.ist.psu.edu/article/coupe00sensitivity.html

[31] S.B. Cousins and M.E. Frisse, "Query networks for medical information retrieval—assigning probabilistic relationships", In: Miller RA, ed. *Proceedings, Symposium on Computer Applications in Medical Care.* New York, NY: IEEE Computer Society, 1990:800–4. 1990. http://citeseer.ist.psu.edu/cousins90query.html

[32] F. Crimmins, A. F. Smeaton, T. Dkaki, and J. Mothe, "TétraFusion: Information Discovery on the Internet", *IEEE Intelligent Systems*, pp55-62, July/August 1999.

[33] B. Cukic and D. Chakravarthy, "Bayesian Framework for Reliability Assurance of a Deployed Safety Critical System", *Proceedings of 5th Int'l Symposium on High Assurance Systems Engineering (HASE 2000)*, Albuquerque, NM, Nov. 2000.

[34] B. Cukic, E. Gunel, H. Singh, and L. Guo, "The Theory of Software Reliability Corroboration", *IEICE Transactions on Information & Systems*, Vol. E86-D, No. 10, October, 2003.

[35] A. P. Dempster, D. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm", *Journal of the Royal Statistical Society*, Ser. B, 39, 1-38, 1977.

[36] S. Donohue and Y. Yu, “Developing a Probabilistic SRE Self-Assessment Tool Using BBNs”, *ISSRE 2001 Student Paper*, 2001.

[37] M. J. Druzdzel, “Some Useful Properties of Probabilistic Knowledge Representations From the Point of View of Intelligent Systems”, *Fundamenta Informaticae*, 30(3/4):241–254, 1996. http://citeseer.ist.psu.edu/article/druzdzel94some.html

[38] A. H. Dutoit and B. Bruegge, “Communication Metrics for Software Development”, *IEEE Trans. Software Eng.*, Vol. 24, No. 8, Aug. 1998.

[39] W. Dzida and R. Freitag, “Making Use of Scenarios for Validating Analysis and Design”, *IEEE Trans. Software Eng.*, Vol. 24, No. 12, Dec. 1998.

[40] C. Ebert, “Classification techniques for metric-based software development”, *Software Quality Journal*, 5(4):255-272, Dec. 1996.

[41] C. Ebert and E. Baisch, “Industrial Application of Criticality Prediction in Software Development”, *Proc. the Nineth Iternational Symposium on Software Reliability Eng.* pp80, Nov. 1998.

[42] K. EI Emam and A. Birk, “Validating the ISO/IEC 15504 Measure of Software Requirements Analysis Process Capability”, *IEEE Trans. Software Eng.*, Vol. 26, No. 6, Jun. 2000.

[43] M. Factor, D. H. Gelernter C. E. Kolb, P. L. Miller, and D. F. Sittig, “Real-Time Data Fusion in the Intensive Care Unit”, *IEEE Computer*, pp45-54, Nov. 1991.

[44] N. E. Fenton and M. Neil, “A Strategy for Improving Safety Related Software Engineering Standards”, *IEEE Trans. Software Eng.*, Vol. 24, No. 11, Nov. 1998.

[45] N. E. Fenton and M. Neil, “A Critique of Software Defect Prediction Models”, *IEEE Trans. Software Eng.*, Vol. 25, No. 5, Sep. 1999.

[46] N. Fenton and M. Neil, “Software Metrics and Risk”, *Proc. 2nd European Software Measurement Conference*, TI-KVIV, Amsterdam, Oct. 1999. http://www.agena.co.uk/resources.html

[47] N. Fenton and M. Neil, “Software Metrics: Roadmap”, invited paper for *ICSE2000*, Dec. 1999.

[48] N. Fenton and M. Neil, "Making Decisions: Using Bayesian Nets and MCDA", to appear *Knowledge Based Systems*, 2001. http://www.agena.co.uk/resources.html

[49] A. Filippidis, L.C. Jain, and N. Martin, "Fusion of Intelligent Agents for the Detection of Aircraft in SAR Images", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 22, No. 4, pp378-384, April 2000.

[50] A. Garg, S. Agarwal, and T. S. Huang, "Fusion of Global and Local Information for Object Detection", *Proc. 16th International Conference on Pattern Recognition (ICPR'02)*, Vol. 3, pp30723, August 2002.

[51] T.D. Garvey, J.D. Lowrance and M.A. Fischler, "An inference technique for integrating knowledge from disparate sources", *Proc. 7th International Joint Conference on Artificial Intelligence*, pp319-325, Vancover, British Columbia, 1981.

[52] M. Grottke, "A Vector Markov Model for Structural Coverage Growth and the Number of Failure Occurances", *Proc. 13th International Symposium on Software Reliability Engineering*, pp. 304-315, 2002.

[53] S. S. Gokhale and M. R. Lyu, "Regression tree modeling for the prediction of software quality", *Proc. the Third ISSAT International Conference on Reliability and Quality in Design*, pp31-36, Anaheim, CA, Mar. 1997.

[54] S. Gokhale, M. Lyu, K. Trivedi, "Reliability Simulation of Component-based Software Systems ", *Proc. The Ninth International Symposium on Software Reliability Engineering*, Nov. 1998.

[55] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History", *IEEE Trans. Software Eng.*, Vol. 26, No. 7, Jul. 2000.

[56] A. H. Gunatilaka and B. A. Baertlein, "Feature-Level and Decision-Level Fusion of Non-coincidently Sampled Sensors for Lan Mine Detection", *IEEE Tans. Pattern Analysis and Machine Intelligence*, Vol. 23, No. 6, pp577-589, June 2001.

[57] L. Guo, "Estimating Component Availability by Dempster-Shafer Networks", *Suppl. Proc. Thirteenth International Symposium on Software Reliability Engineering*, pp203, Nov. 2002.

[58] M. A. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining", *IEEE Trans. Knowledge and Data Eng.*, Vol. 15(3), May/June, 2003.

[59] M. Halstead, *Elements of Software Science*, Elsevier, 1977.

[60] D. Hecherman, "A Tutorial on Learning With Bayesian Networks", Technical Report, MSR-TR-95-06, Nov. 1996.

[61] Y. Hel-Or and M. Werman, "Pose Estimation by Fusing Noisy Data of Different Dimensions", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 17, No. 2, pp195-201, Feb. 1995.

[62] M. E. Helander, M. Zhao, and N. Ohlsson, "Planning Models for Software Reliability and Cost", *IEEE Trans. Software Eng.*, Vol. 24, No. 6, Jun. 1998.

[63] S. Henry and D. Kafura, "The Evaluation of Software System's Structure Using Quantitative Software Metrics", *Software Practice and Experience*, Vol. 14, no. 6, pp. 561-573. Jun. 1984.

[64] D. K. Hildebrand, J. D. Laing and H. Rosenthal, Prediction Analysis of Cross Classifications, New York: John Wiley & Sons, 1977.

[65] T. Horiuchi, "Decision Rule for Pattern Classification by Integrating Interval Feature Values", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 20, No. 4, pp. 440-448, April 1998.

[66] Hosmer, David and Stanley Lemeshow, *Applied Logistic Regression*, NY: Wiley & Sons, 1989.

[67] J. C. Huang, "An Approach to Program Testing", *ACM Computing Surveys*, Vol. 8, No. 3, pp. 113-128, Sept. 1975.

[68] C. J. Huberty, *Applied discriminant analysis*, Wiley, 1994.

[69] J. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, J. Maryland, "Emerald: Software Metrics and Models on the Desktop", *IEEE Software*, pp. 56-60, Sep. 1996.

[70] HUGIN Expert A/S. P. O. Box 8201 DK-9220 Aalborg, Denmark. http://www.hugin.com, 2004

[71] S. Jabri, Z. Duric, H. Wechsler, and A. Rosenfeld, "Detection and Location of People in Video Images Using Adaptive Fusion of Color and Edge Information", *Proc. International Conference on Pattern Recognition (ICPR'00)*, Vol. 4, pp627, Sep. 2000.

[72] J. J. Dolado, "A Validation of the Component-Based Method for Software Size Estimation", *IEEE Trans. Software Eng.*, Vol. 26, No. 10, Oct. 2000.

[73] C. Jones, *Applied Software Measurement.* McGraw-Hill, 1991.

[74] K. Kanoun, M. Kaâniche, and J-C Laprie, "Qualitative and Quantitative Reliability Assessment", *IEEE Software*, Mar./Apr. 1997.

[75] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Predition of Software Reliability Using Connectionist Models", *IEEE Trans. Software Eng.*, Vol. 18, No. 7, Jul. 1992.

[76] P. A. Keiller and D. R. Miller, "On the use and the performance of software reliability growth models," *Reliability Engineering and System Safety*, pp. 95-117, 1991.

[77] J. Kittler and F. M. Alkoot, "Sum versus Vote Fusion in Multiple Clssifier Systems", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 25, No. 1, pp110-115, Jan. 2003.

[78] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel, "Early quality prediction: A case study in telecommunications", *IEEE Software*, 13(1):65-71, Jan. 1996.

[79] T. M. Khoshgoftaar, E. B. Allen, F. D. Ross, R. Munik oti, N. Goel, and A. Nandi, "Predicting Fault-Prone Modules with Case-Based Reasoning", *Proc. the Eighth International Symposium on Software Engineering (ISSRE'97)*, pp27, Nov. 1997.

[80] T. M. Khoshgoftaar, E. B. Allen, and Z. Xu, "Prediction Testability of Program Modules Using a Neural Network", *3th IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00)*, Mar. 2000.

[81] T. M. Khoshgoftaar, and D. L. Lanning, "A neural network approach for early detection of program modules having high risk in the maintenance phase", *Journal of Systems and Software*, 29(1):85-91, Apr. 1995.

[82] T. M. Khoshgoftaar and N. Seliya, "Tree-Based Software Quality Estimation Models For Fault Prediction", *Proc. the Eighth IIIE Symposium on Software Metrics (METRICS'02)*, pp203, Jun. 2002.

[83] R. Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection”, *Internation Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[84] L. I. Kuncheva, “A Theoretical Study on Six Classifier Fusion Strategies”, *IEEE Trans. PAttern Analysis and Machine Intelligence*, Vol. 24, No. 2, pp281-286, Feb. 2002.

[85] J. Laplace, M. Brun, “Critical Software for Nuclear Reactors: 11 Years of Field Experience Analysis”, *Proc. Of the 9th International Symposium on Software Reliability Engineering (ISSRE99)*, Paderborn, Germany, pp. 364-368, Nov. 1998.

[86] S. L. Lauritzen and D. J. Spiegelhalter, “Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems (with discussion)”, *J. R. Statis. Soc. B*, 50, No. 2, pp 157-224, 1988.

[87] J. D. Lawrence, W. L. Persons, G. G. Preckshot, J. Gallagher, “Evaluating Software for Safty Systems in Nuclear Power Plants”, *Proceedings of the 9th Annual Conference on Computer Assurance (COMPASS 94)*, Vol. 35, No. 2, pp. 13-21, Feb, 1992.

[88] M. Li, *private communication*, Jan. 20, 2004.

[89] M. Li and C. S. Smidts, “A Ranking o Software Engineering Measures Based on Expert Opinions”, *IEEE Trans. Software Engineering*, Vol. 29, No. 9, pp811-824, Sep. 2003.

[90] JC Lin, SW Lin and I. Ho, “An Estimated Method for Software Testability Measurement”, *Proc. 8th International Workshop on Software Technology and Engineering Practice (STEP’97)*, 1997.

[91] B. Littlewood, “Predicting software reliability,” *Philosophical Transactions of the Royal Society* (London), pp. 513-526, 1989.

[92] B. Littlewood and L. Strigini, “Validation of Ultra-High Dependability for Software-based Systems”, *Communications of the ACM*, 36(11), pp.69-80, 1993.

[93] B. Littlewood and D. Wright, “Some Conservative Stopping Rules for the Operational Testing of Safety-Critical Software”, *IEEE Trans. Software Eng.*, Vol. 23, No. 11, Nov. 1997.

[94] J. Liu and M. C. Desmarais, “A Method of Learning Implication Networks from Empirical Data: Algorithm and Monte-Carlo Simulation-Based Validation”, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 6, pp. 990-1004, Nov./Dec. 1997.

[95] C. Liu, L. Kong, W. Zhoun, J. Zhu, and D. Xia, "Multi-Information Fusion Based Tumor Cell of Bone Marrow Involvement", *Proc. International Workshop on Medical Imaging and Augmented Reality (MIAR'01)*, pp211, June, 2001.

[96] J. Liu, D. Maluf and M. Desmarais, "A New Uncertainty Measure for Belief Networks with Applications to Optimal Evidential Inferencing", *IEEE Trans. Knowledge and Data Eng.* Vol. 13, No. 3, pp. 416-425, May/June 2001.

[97] R. M. Losee and L. Church Jr., "Information Retrieval with Distributed Databases: Analytic Models of Performance", *IEEE Trans. Parallel and Distributed Systems*, Vol. 15, No. 1, pp18-27, Jan. 2004.

[98] M. R. Lyu (editor in chief), *Handbook of software reliability engineering*, IEEE Computer Society Press , New York : McGraw Hill, 1996.

[99] Y. K. Malaiya, N. Li, M. Bieman, R. Karcich, B. Skibbe, and S. Tek, "Software Test Coverage and Reliability", *Technical Report CS-96-128*, Colorado State University, Fort Collins, Colorado, 1996.

[100] A. J. Maren, R. M. Pap, and C. T. Harston, "A hierarchical data structure representation for fusion of multisensor information", *SPIE*, Vol. 1100, Sensor Fusion II, pp.162-178, Orlando, FL, 1989.

[101] T. Matsuyama, "Belief Formation From Observation and Belief Integration Using Virtual Belief Space in Dempster-Shafer Probability Model", *Proc. 1994 Multisensor Fusion and Integration for Intelligent Systems (MFI'94)*, pp. 379-386, 1994.

[102] J. H. R. May and A. D. Lunn, "A Model of Code Sharing for Estimating Software Failure on Demand Probabilities", *IEEE Trans. Software Eng.*, Vol. 21, No. 9, Sep. 1995.

[103] McCabe and Associates, "Software metrics: McCabe metrics", http://www.mccabe.com/metrics.php, 2003.

[104] T. McCabe, "A Complexity Measure", *IEEE Trans. Software Eng.* 2(4):308-320, Dec. 1976.

[105] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing", *Communications of the ACM*, 32(12), pp. 1415-1425, Dec. 1989.

[106] , M. C. McCabe, A. Chowdhury, D. Grossman, and O. Frieder, "System Fusion for Improving Performance in Information Retrieval Systems", *Proc. International Conference on Information Technology: Coding and Computing (ITCC'01)*, pp639, April 2001.

[107] McKelvey, Richard D. and William Zavoina, "A Statistical Model for the Analysis of Ordinal Level Dependent Variables", *Journal of Mathematical Sociology*, 4:103-120, 1975.

[108] S. McRoy, S. Haller and S. Ali, "Uniform Knowledge Representation for Language Processing in the B2 System", *Journal of Natural Language Engineering 1997*; 3(2). 1997. http://citeseer.ist.psu.edu/mcroy97uniform.html

[109] MDP program at NASA IV&V, http://mdp.ivv.nasa.gov, 2004.

[110] T. Menzies, J. D. Stefano, K. Ammar, R. M. Chapman, K. McGill, P. Callis, and J. Davis, "When Can We Test Less?", *IEEE Metrics'03*, 2003. http://menzies.us/pdf/03metrics.pdf

[111] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas, "Estimating the Probability of Failure When Testing Reveals No Failures", *IEEE Trans. Software Eng.*, Vol. 18, No. 1, Jan. 1992.

[112] J. C. Munson and T. M. Khoshgoftaar, "The detection of fault-prone programs", *IEEE Trans. Software Eng.* 18(5):423-433, May 1992.

[113] R. R. Murphy, "Adaptive Rule of Combination for Observations Over Time", *Proc. 1996 Multisensor Fusion and Integration for Intelligent Systems (MFI'96)*, pp125-131, 1996.

[114] R. R. Murphy, "Sensor and Information Fusion for Improved Vision-Based Vehicle Guidance", *IEEE Inteligent Systems*, pp49-56, Nov./Dec. 1998.

[115] J. D. Musa, "Operational Profiles in Software-Reliability Engineering", *IEEE Software*, Mar. 1993.

[116] I. Myrtveit and E. Stensrud, "A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models", *IEEE Trans. Software Eng.*, Vol. 25, No. 4, Jul./Aug. 1999.

[117] M. Neil and N. Fenton, "Predicting Software Quality using Bayesian Belief Networks", *Proc. 21st Annual Software Eng. Workshop*, Dec. 1996. http://www.agena.co.uk/resources.html

[118] M. Neil, N. Fenton, and L. Nielsen, "Building Large-Scale Bayesian Networks", *The Knowledge Engineering Review*, Vol. 15, No. 3, pp257-284, 2000. http://www.agena.co.uk/resources.html

[119] M. Neil, B.Littlewood, and N. Fenton, "Applying Bayesian Belief Networks to System Dependability Assessment", *Proc. 4th Safety-Critical Systems Symposium*, Leeds, UK, Feb. 1996.

[120] B. N. Nelson, "Region of Interest Identification, Feature Extraction, and Information Fusion in A Forward Looking Infrared Sensor Used in Landmine Detection", *IEEE Workshop on Computer Vision Beyond the Visible Spectrum: Methods and Applications (CVBVS 2000)*, pp94, June 2000.

[121] N. E. Fenton and N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System", *IEEE Trans. Software Eng.*, Vol. 26, No. 8, Aug. 2000.

[122] N. Ohlsson and H. Alberg, "Predicting Error-Prone Software Modules in Telephone Switches", *IEEE Trans. Software Eng.*, Vol. 22, no. 12, pp. 886-894, 1996.

[123] H. Pan, Z.P. Liang, and T. S. Huang, "Exploiting the Dependencies in Infromation Fusion", *Computer Vision and Pattern Recognition*, Vol. 2, pp2407, June 1999.

[124] P. Piwowarski, M. Ohba, and J. Caruso, "Coverage Measurement Experience During Function Test", *Proc. 15th International Conference on Software Engineering*, pp. 287-301, 1993.

[125] M. Prantl, H. Ganster, and A. Pinz, "Active Fusion Using Bayesian Networks Applied to Multi-Temporal Remote Sensing Imagery", *Proc. 1996 International Conference on Pattern Recognition (ICPR'96)*, pp890-894, 1996.

[126] M. Ramoni and P. Sebastiani, "Efficient Parameter Learning in Bayesian Networks from Incomplete Data", KMi Technical Report KMi-TR-41, Jan. 1997.

[127] M. Ramoni and P. Sebastiani, "Learning Bayesian Networks from Incomplete Databases", KMi Technical Report KMi-TR-43, Feb. 1997.

[128] N.S.V.Rao, “On Fusuers that Perform Better than Best Sensor”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 23, No. 8, pp904-909, August 2001.

[129] S. Raudys, “Experts’ Boasting in Trainable Fusion Rules”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 25, No. 9, pp1178-1182, Sep. 2003.

[130] A. T. Rivers and M. A. Vouk, “Resource-Constrained Non-Operational Testing of Software”, *Proc. 9th International Syposium on Software Reliability Engineering*, pp. 287-301, 1993.

[131] J. Ruiz-del-Solar and A. Soria-Frisch, “Bio-inspired Framework for the Fusion of Chromatic, Infrared and Textual Information”, *Proc. International Conference on Pattern Recognition (ICPR’00)*, Vol. 3, pp3576, Sep. 2000.

[132] N. F. Schneidewind, “Methodology For Validating Software Metrics”, *IEEE Trans. Software Eng.* 18(5):410-422, May 1992.

[133] N. F. Schneidewind, “Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics”, *IEEE Trans. Software Eng.*, Vol. 25, No. 8, Nov./Dec. 1999.

[134] P. Sebastiani and M. Ramoni, “Bayesian Inference Eith Missing Data Using Bound and Collapse”, *Journal of Computational and Graphical Statistics*, 9(4):779-800, 2000.

[135] R. W. Selby and A. A. Porter, “Learning from examples: Generation and evaluation of decision trees for software resource analysis”, *IEEE Trans. Software Eng.* 14(12):1743-1756, Dec. 1988.

[136] G. Shafer, A Mathematical Theory of Evidence, Princeton University Press, 1976.

[137] M. Shin and A. L. Goel, “Empirical Data Modeling in Software Engineering Using Radial Basis Functions”, *IEEE Trans. Software Eng.*, Vol. 26, No. 6, Jun. 2000.

[138] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj, “A Bayesian Approach to Reliability Prediction and Assessment of Component Based Systems”, *Proc. 12th International Symposium on Software Reliability Engineering (ISSRE’01)*, Nov. 2001.

[139] C. Smidts, B. Cukic, E. Gunel, M. Li, and H. Singh, “Software Reliability Corroboration”, *Proc. of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop (SEW-27’02)*, 2002.

[140] , C. Smidts and M. Li, "Validation of a Methodology for Assessing Software Quality", *Technicle Report*, the University of Maryland at College Park, 2002.

[141] C. Smidts, M. Stutzke, and R. W. Stoddard, "Software Reliability Modeling: An Approach to Early Reliability Prediction", *IEEE Transactions on Reliability*, vol. 47, no.3 pp. 268-78, 1998.

[142] P.C. Smits and S. Dellepiane, "Information Fusion in a Markov Random Field-Based Image Segmentation Approach Using Adaptive Neighbourhoods", *Proc. 1996 International Conference on Pattern Recognition (ICPR'96)*, pp570-575, 1996.

[143] D. J. Spiegelhalter, A. Thomas, and N. G. Best, "Computation on Bayesian Graphical Models", *Bayesian Statistics 5*, Oxford, U. K.: Oxiford University Press, pp. 407-425, 1996.

[144] W. Stevens, G. Myers, and L. Constanine, "Structured Design", *IBM Systems J.*, Vol. 13, No. 2, pp. 115-139, 1974.

[145] H. Sugimoto and A. Ohnishi, "A Detecting and Interpreting Method of the Inconsistency of Software Requirements Specifications", *Sixth Asia Pacific Software Engineering Conference*, pp208, Takamatsu, Japan, Dec 7-10, 1999.

[146] A. G. Sutcliffe, N. A. M. Maiden, S. Minocha, and D. Manuel, "Supporting Scenario-Based Requirements Engineering", *IEEE Trans. Software Eng.*, Vol. 24, No. 12, Dec. 1998.

[147] M. R. Tremblay and M. R. Cutkosky, "Using Sensor Fusion and Contextual Information to Perform Event Detection during a Phase-Based Manipulation Task", *Proc. International Conference on Intelligent Robots and Systems (IROS'95)*, Vol. 3, pp262-267, August 1995.

[148] J. Troster and J. Tian, "Measurement and defect modeling for a legacy software system", *Annuals of Software Eng.* 1:95-118, 1995.

[149] J. M. Voas, C. C. Michael, and K. W. Miller, "Confidently Assessing a Zero Probability of Software Failure", *High Integrity Systems*, Vol. 1, No. 3, pp269-275, 1995.

[150] M. A. Vouk, "Using Reliability Models During Testing with Non-Operational Profiles", *2nd Bell-Core/Purdue Workshop Issues in Software Reliability Estimation*, pp. 103-111, Oct. 1992. http://renoir.csc.ncsu.edu/Faculty/Vouk/Papers/Vouk/Vouk_WISRE92.pdf

[151] G. Wiederhold, “Mediators in the Architecture of Future Information Systems”, *IEEE Computer*, pp38-49, March 1992.

[152] N. Wilson, “The Assumptions Behind Dempster’s Rule”, *Proc. the Ninth Conference on Uncertainty in Artificial Intelligence*, pp.527–534, Washhington, DC, July 9-11, 1993.

[153] Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, October 1999. http://www.cs.waikato.ac.nz/ml/weka/

[154] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham, “Weka: Practical Machine Learning Tools and Techniques with Java Implementations”, http://www.cs.waikato.ac.nz/∼ihw/papers/99IHW-ETF-LT-MJ-GH-SJC-Weka.pdf, 1999.

[155] S. T. Wong, R. C. Knowlton, R. A. Hawkins, and K. D. Laxer, “Multimodal Image Fusion for Noninvasive Epilepsy Surgery Planning”, *IEEE Computer Graphics and Applications*, pp30-38, Jan. 1996.

[156] Q. Wu, N. S. V. Rao, J. Berhen, S. Sitharama lyengar, V. K. Vaishnavi, H. Qi, and K. Chakrabarty, “On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks”, *IEEE Trans. Knowledge and Data Engineering*, Vol. 16, No. 6, pp740-753, June 2004.

[157] Y. Xiang, M. P. Beddoes, and D. Poole. “Sequential Updating Conditional Probability in Bayesian Networks by Posterior Probability”, *Proceedings of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 21–27, 1990.

[158] R. R. Yager, “On the Dempster-Shafer Framework and New Combination Rules”, *Information Science*, Vol. 41, pp. 93-137, 1987.

[159] S. Yerramalla, E. Fuller, M. Mladenovski, and B. Cukic, “Lyapunov Analysis of Neural Network Stability in an Adaptive Flight Control System”, *Sixth Symposium of Self-stabilization (SSS)*, June, 2003.

[160] L. Zadeh, “A Simple View of the Dempster-Shafer Theory of Evidence and its Implication for the Rule of Combination”, *AI Magazine*, Vol. 7, No. 2, pp85-90, 1986.

[161] Z. Zhang and C. Zhang, "Result Fusion in Multi-Agent Systems Based on OWA Operator", *Proc. Australasian Computer Science Conference*, pp234, Jan. 2000.

[162] Y. Zhou and H. Leung, "Minimum Entropy Approach for Multisensor Data Fusion", *Proc. 1997 IEEE Signal Processing Workshop on High-Order Statistics (SPW-HOS'97)*, pp336-339, 1997.