Enhancing System Reliability using Abstraction and Efficient Logical Computation

Takuro Kutsuna

Abstract

Embedded systems are working almost everywhere in our life from consumer electronics to process control systems in a factory. Automotive control systems are one of embedded systems that are getting increasingly large-scaled and complex. Another important aspect of automotive control systems is that it must be "safety critical", because an error happened in the system may have a strong impact on society. Therefore, it is required to guarantee high reliability in automotive control systems.

The goal of our research is to develop efficient methods for enhancing system reliability. In this thesis, we deal with three domains for enhancing system reliability: software testing, anomaly detection, and diagnosis of the system. We propose novel and efficient methods for applications in these domains. In Part I, we discuss an approach for automatically generating test cases that can be used to test software in the system. In Part II, a one-class classification method and an outlier detection method are proposed, both of which are data-driven methods for detecting anomalous behavior that may happen in the system. In Part III, we develop a diagnostic method for locating the root cause of errors that are redundantly detected in the system. All of the methods proposed in this thesis are built upon the notion of abstraction. The complexity of solving a problem can be reduced by properly abstracting a certain aspect of the problem. We employ logical representation for tackling our problems to benefit from recent progress in solving satisfiability problems and to utilize efficient data structure that is specialized to deal with Boolean functions.

Acknowledgments

I would first like to express my sincere gratitude to my supervisor, Professor Akihiro Yamamoto for providing me the opportunity to become a member of his laboratory as a PhD student. He gave me valuable, stimulating, and warm suggestions for this thesis. I wish to thank Professor Atsushi Igarashi and Professor Hisashi Kashima for being committee members of this thesis and for their insightful comments.

I am also grateful to Professor Masao Fukushima, the supervisor in my master course in Kyoto University. I have learned a lot from his sincere and earnest attitude toward scientific research.

I would also like to thank current and former members of the software research laboratory in Toyota Central R&D Labs. Inc.: Mr. Noriyoshi Sano, Mr. Yutaka Inamori, Dr. Shuichi Sato, Dr. Naoya Chujo, and Mr. Yoshinao Ishii for their useful comments and supports regarding my research and for encouraging me to take a doctoral degree.

Last but not the least, I would like to express my heartfelt gratitude to my family and my parents for the support they provided me throughout my life.

Takuro Kutsuna
August 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Embedded systems are getting large-scaled and complex in many fields of information and communication technology. Automotive control systems are one of such complex systems that are composed of a lot of electronic control units (ECUs). For example, a premium-class automobile consists of about 100 million lines of software code and has dozens of ECUs which are networked throughout the body of the car [20, 36]. Increasing demands on vehicle safety, driving assistance, environmental protection, and automated cruise will continue to drive this trend.

Automotive control systems have an important aspect that they must be "safety critical", because an error happened in the system may have a strong impact on society. Therefore, it is required to guarantee high reliability in the system. The goal of our research is to develop efficient methods for enhancing system reliability. In this thesis, we propose various methods to achieve the goal, which have a common feature that abstraction and logical representation are used as the keys for constructing efficient algorithms. We employ logical representation to tackle our problems to benefit from recent progress in solving satisfiability problems and to utilize efficient data structure that is specialized to deal with Boolean functions. In this chapter, we first explain the domains to which the methods discussed in this thesis are applied. Then, technical features that are common in the proposed methods are described.

1.1 Domains for Enhancing System Reliability

In this thesis, we consider three domains for enhancing system reliability: 1) testing a system under development, 2) detecting anomalous behavior of the system, and 3) diagnosing the system. Methods for testing the system are used during the development of the system, while methods for anomaly detection and diagnosis are expected to be used after the system is launched into the market, as shown in Fig. 1.1. The reliability of the system is secured by testing the system thoroughly before it is put on the market. Furthermore, it is also very important to find unexpected behavior of the system that may happen in the market and to investigate the root cause for providing an essential countermeasure as soon as possible. In the following, we give more details of each domain that is dealt with in this thesis.

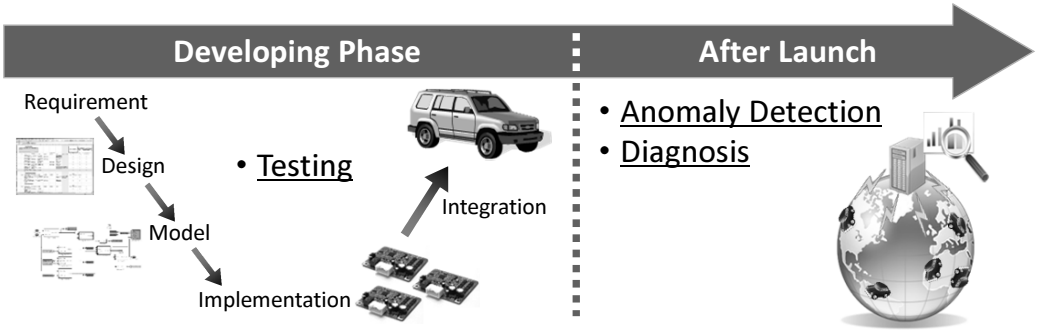


Figure 1.1: Domains discussed in this thesis (underlined).

Software Testing

We deal with the problem of testing software in embedded control systems. In particular, we develop a method for automatically generating test cases of embedded control software. In automotive industry, test cases are often used to show the conformance between a high-level model, such as Simulink® [70] model, and the code derived from the model. Practically, the automotive safety standard ISO 26262 recommends that manufacturers utilize test cases

that satisfy the modified condition/decision coverage [44] to show the conformance between the model and the code. Test-case generation methods can be used to produce such test cases automatically [54]. Moreover, mutation testing [39] and falsification [1] can be regarded as applications of test-case generation methods.

Anomaly Detection

Automotive control systems include many modules to detect pre-defined faults, such as those defined in diagnostic trouble codes in the on-board diagnostic standards [50]. However, it is essentially impossible to list all potential faults in advance. To provide methods for complementarily detecting anomalous behavior of the system, we employ a data-driven approach. Two types of data-driven methods for detecting anomalies in the system are proposed in this thesis: one-class classification and outlier detection.

One-Class Classification One-class classification is the problem of learning a classifier from a training data set that mostly contains objects of a target class. The learned classifier is used to distinguish objects of the target class from those of other classes. Figure 1.2 illustrates an application of one-class classification. In Figure 1.2, a classifier is learned from a data set that is collected from a car that works normally. Then, the learned classifier is used to monitor the system whether it is working normally or not.

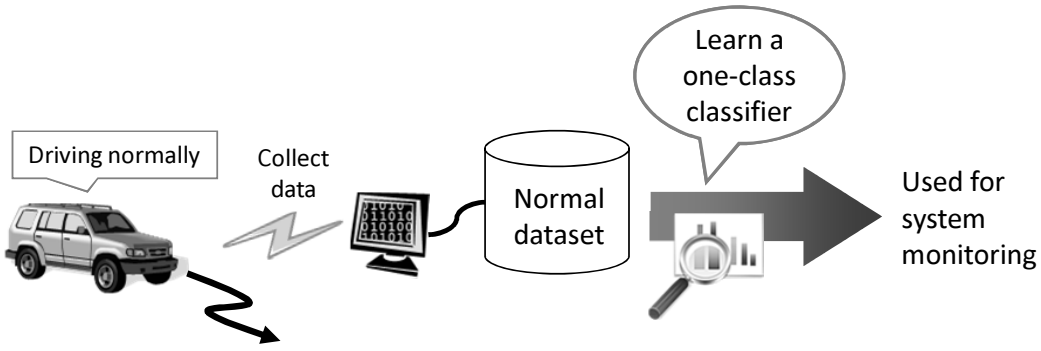


Figure 1.2: One-class classification for system monitoring.

Outlier Detection The goal of outlier detection is to find an unusual datum (*outlier*) within a given data set. The main difference between outlier detection and one-class classification is that no external criterion is available in outlier detection, whereas a training data set is given as an external criterion in one-class classification. An application image of outlier detection is illustrated in Figure 1.3. In Figure 1.3, a data set is first collected from automobiles in the market. Then, anomalous behavior of the system is discovered by using outlier detection methods.

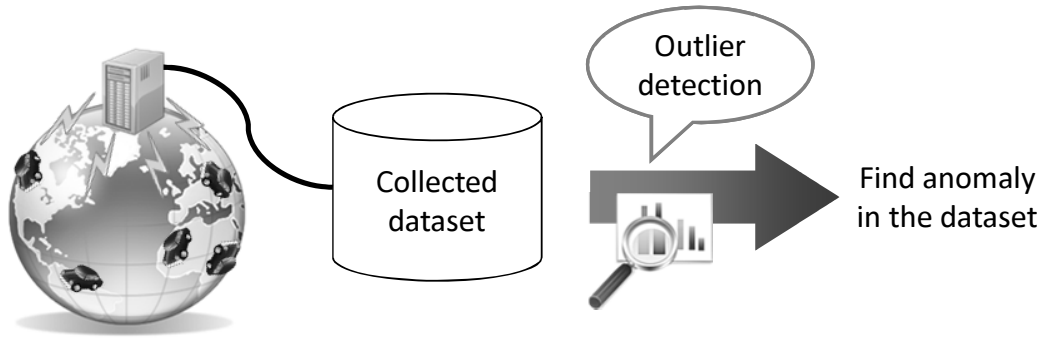


Figure 1.3: Analyzing a collected data set with outlier detection.

Diagnosis

As mentioned previously, automotive control systems comprise a lot of ECUs. These ECUs communicate with one another via automotive networks such as CAN, LIN, and FlexRay [52]. In such a situation, it tends to be difficult to locate faults during system failures, because ECUs that receive abnormal input data from other ECUs may also output abnormal data, even if they are not themselves in an abnormal state. Consequently, many redundant errors are detected in the system and the root cause analysis becomes an increasing challenge, as shown in Figure 1.4. We call this problem the error-propagation problem, which will be more likely in systems in which many components collaborate closely.

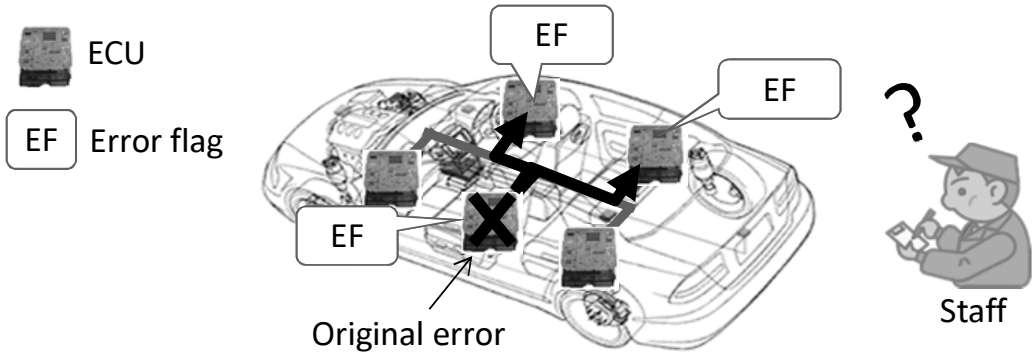


Figure 1.4: Error-propagation problem.

1.2 Contributions

All of the methods proposed in this thesis are built upon the notion of abstraction and logical representation. The complexity of solving a problem can be reduced by properly abstracting a certain aspect of the problem. The reason why we use logical representation for tackling our problems is to take advantage of recent progress in solving satisfiability problems and to utilize efficient data structure that is specialized to deal with Boolean functions. Table 1.1 summarizes the characteristics of the methods proposed in this thesis in two points: what is abstracted, and what kind of logical representation is used.

Chapter 2: Abstraction and Refinement of Mathematical Functions for Generating Test Cases

In Chapter 2, we propose a novel technique for test-case generation. The key feature of the proposed method is that it can generate test cases efficiently for software that contains mathematical functions, such as trigonometric functions, logarithmic functions, and functions implemented as look-up tables with non-linear interpolation. Such mathematical functions typically exist in embedded control software in automotive and avionic systems. Existing methods for test-case generation have difficulty in dealing with software with such mathematical functions. A satisfiability modulo theories (SMT) solver is iteratively used to generate test cases in the scheme of bounded model

Table 1.1: Characteristics of the methods proposed in this thesis.

Domain	What is abstracted?	Logical representation
Test-case generation (Chapter 2)	In/out relationship of mathematical functions	SMT formula
One-class classification (Chapter 3)	Numeric data	BDD
Outlier detection (Chapter 4)		
Diagnosis (Chapter 5)	Behavior of system components	MaxSAT formula

checking. In the proposed method, the relationship between the input and the output of a mathematical function in a program is abstracted so that the derived formula can be easily treated using an SMT solver. The abstraction is refined adaptively based on the previous counterexamples. We also propose a general method to estimate an abstraction of a mathematical function by means of sampling and machine learning. Although the method proposed addresses mainly the topic of test-case generation, it is also applicable to ordinary bounded model checking under the assumption that the abstraction should be a correct over-approximation.
(All results presented in this chapter have been published in [J1].)

Chapter 3: One-Class Classification using Binary Decision Diagrams

In Chapter 3, we propose a novel method for learning a one-class classifier. A distinguishing characteristic of the proposed classifier is that the parameter of the classifier can be tuned automatically using the training data set. Although cross validation is often used to tune parameters of a classifier, it is generally difficult to apply cross validation in the setting of one-class classification, because the training data set is assumed to contain no labeled data. Another key feature of the proposed method is that it enables us to learn a classifier efficiently from a large data set. The computational complexity of

learning the proposed classifier is almost linear with respect to the number of data in the training data set. Such a feature is very important in one-class classification, because the training data set tends to be large since collecting unlabeled data is relatively easy. In the proposed method, numeric data is abstracted by discretizing them into intervals. The region of a training data set is expressed as a Boolean formula that is constructed by using a binary decision diagram (BDD). Then, the region is efficiently over-approximated through the direct manipulation of the binary decision diagram.
(All results presented in this chapter have been published in [J2, P2].)

Chapter 4: Outlier Detection using Binary Decision Diagrams

In Chapter 4, we present a new approach for outlier detection, which is an extension of the method described in Chapter 3. Leave-one-out density is proposed as a new measure for detecting outliers, which is defined as a ratio of the number of data elements inside a region to the volume of the region after a focused datum is removed. We show that the leave-one-out density can be evaluated very efficiently on a set of regions around each datum in a given data set by using BDDs. The time complexity of the proposed method is near linear with respect to the size of the data set, while the outlier detection accuracy is still comparable to that of other methods.
(All results presented in this chapter have been published in [P1].)

Chapter 5: Abstract Model-Based Diagnosis for Detecting Root Causes

In Chapter 5, we propose a diagnostic method for automatically locating the origin of errors in a system in which the error-propagation problem may occur. The proposed method can be applied to a system that includes complex components like ECUs. The behavior of such components is abstracted so that they can be dealt with in the scheme of model-based diagnosis. We propose a new method to handle systems that have data-flow loops. The problem of locating the origin of errors is efficiently solved based on its formulation into the maximum satisfiability problem (MaxSAT).
(All results presented in this chapter have been published in [J3, P3].)

Part I

Software Testing

Chapter 2

Abstraction and Refinement of Mathematical Functions for Generating Test Cases

In this chapter, we propose a novel technique for automatically generating a test case of software that contains mathematical functions, such as trigonometric functions, logarithmic functions, and functions implemented as look-up tables with non-linear interpolation. The proposed method is based on the notion of *bounded model checking* (BMC) [21] and we utilize *satisfiability modulo theories* (SMT) solvers to generate test cases. The BMC approach for test-case generation is very powerful and large software can handled practically by virtue of the recent progress in SMT solving techniques. However, to the best of our knowledge, a problem related to handling software that contains complex mathematical functions still remains to be solved. Some mathematical functions, e.g., trigonometric functions, are usually not supported by the SMT language. Even if we succeeded in modeling mathematical functions into an SMT formula, the computation time of checking the satisfiability of this formula would increase drastically according to the size of the software. The method we propose in this study enables us to efficiently generate test cases for software that includes such mathematical functions. It is also applicable to the ordinary BMC under some assumptions. In other words, the proposed method allows a property of software

with complex mathematical functions to be verified.

In the proposed method, mathematical functions are abstracted so that the derived SMT formula can be expressed in difference logic [5]. A formula in difference logic is in general easier to solve than one in another logic, such as non-linear arithmetic. If the assignment obtained by an SMT solver is spurious because of an error caused by the abstraction, the part of the formula that states the relationship between the input and the output of a mathematical function is refined to express the relationship more precisely around the previous assignment. We also propose a general method to estimate an abstraction of a function based on sampling and machine learning. We can abstract any function by using the proposed method, provided that the function is executable.

We provide experimental results showing that by using the proposed method it is possible to generate test cases in a sufficiently short time to allow its practical application for software that is intractable with an ordinary SMT-based method. The software used in the experiment consists of an element of embedded control software taken from the automotive industry, which has three look-up tables with non-linear interpolation.

Test-case generation based on an SMT solver is reviewed using an example in Section 2.1. An abstraction of a mathematical function and its refinement is defined and the proposed method is explained in Section 2.2. A machine learning-based method to estimate an abstraction of a function is proposed in Section 2.3. In Section 2.4, the experimental results are presented. In Section 2.5, we review related work. In Section 2.6, we give summary of this chapter.

2.1 Background

In this section, we review SMT-based test-case generation using an example that is given as a Simulink® [70] model. Simulink® is a graphical programming tool that is often used in the development of embedded control software.

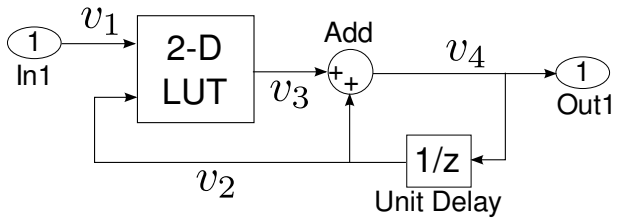


Figure 2.1: Example of a Simulink® model with a 2-D look-up table (LUT).

2.1.1 A Model with a Look-up Table

We consider a Simulink® model that includes a two-dimensional (2-D) look-up table (LUT), as shown in Figure 2.1. It is assumed that the model is executed K times. The model has input, which is denoted as v_1 in Figure 2.1. The value of the input may vary at each execution step. Therefore, a test case for this model consists of K values, i.e., time series. The unit delay block outputs the value that was inputted to the block in the previous execution. The block 2-D LUT is in general used to represent a non-linear function that has two inputs. In Figure 2.1, (v_1, v_2) are the inputs and v_3 is the output of the 2-D LUT; their relationship is defined in Table 2.1. The output value is calculated by interpolation if the input values are between breakpoints. Several interpolating methods exist. For example, *bilinear interpolation* calculates the output value based on the equations

$$\begin{aligned} v_3 = & \frac{1}{(\alpha_{i+1}-\alpha_i)(\beta_{j+1}-\beta_j)}\big(\gamma_{i,j}(\alpha_{j+1}-v_1)(\beta_{i+1}-v_2) \\ & + \gamma_{i+1,j}(\alpha_{j+1}-v_1)(v_2-\beta_i) \\ & + \gamma_{i,j+1}(v_1-\alpha_j)(\beta_{i+1}-v_2) \\ & + \gamma_{i+1,j+1}(v_1-\alpha_j)(v_2-\beta_i)\big) \\ \text{if} \quad & \alpha_i < v_1 < \alpha_{i+1} \quad \text{and} \quad \beta_j < v_2 < \beta_{j+1} \\ & (i = 1, \ldots, p-1; j = 1, \ldots, q-1). \end{aligned} \tag{2.1}$$

If the input values are outside breakpoints, the output value is calculated by extrapolation.

Table 2.1: 2-D LUT: $\alpha_1, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_q$ are breakpoints of v_1 and v_2, respectively. $\gamma_{1,1}, \ldots, \gamma_{p,q}$ are output values at grid points.

	β_1	$\cdots$	β_q
α_1	$\gamma_{1,1}$	$\cdots$	$\gamma_{1,q}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
α_p	$\gamma_{p,1}$	$\cdots$	$\gamma_{p,q}$

2.1.2 Test-case Generation with an SMT solver

In the BMC approach for test-case generation, three kinds of formula are considered: the initial condition, the transition relation, and the property. For example, the transition relation for the model shown in Figure 2.1 is defined as

$$T\left(v^{(k-1)}, v^{(k)}\right) := \left(v_3^{(k)} = f_{\mathrm{LUT}}\left(v_1^{(k)}, v_2^{(k)}\right)\right) \wedge \left(v_4^{(k)} = v_2^{(k)} + v_3^{(k)}\right) \wedge \left(v_2^{(k)} = v_4^{(k-1)}\right), \tag{2.2}$$

where a variable with superscript k represents one in the k-th step, $v^{(k)} = (v_1^{(k)}, \ldots, v_4^{(k)})$, and f_{LUT} is the function that represents the 2-D LUT in Figure 2.1. Assuming that the initial value of the unit delay block is set to zero, the initial condition is given as

$$I\left(v^{(0)}\right) := v_4^{(0)} = 0.$$

The property defines the requirement that the generated test case must satisfy[1]. For example,

$$P\left(v^{(1)}, \ldots, v^{(K)}\right) := 9 \leq v_4^{(K)} \leq 10. \tag{2.3}$$

Finally, we consider the formula Φ defined as

$$\Phi\left(v^{(0)}, \ldots, v^{(K)}\right) := I \wedge \bigwedge_{k=1}^{K} T\left(v^{(k-1)}, v^{(k)}\right) \wedge P, \tag{2.4}$$

[1]In the ordinary BMC, the property defines the condition that should not happen, while we define it as the condition that the generated test case should satisfy.

where the arguments of both I and P are omitted for simplicity. Then, the satisfiability of Φ is checked using a suitable SMT solver. If Φ is satisfiable, the assignments for $v_1^{(k)}$ $(k = 1, \ldots, K)$, which indicates the input to the system in Figure 2.1, become a test case that satisfies the property. Otherwise, it is proved that there is no test case that satisfies the property within K steps.

2.1.3 Motivation

Modern sophisticated SMT solvers can handle polynomial constraints, such as Equation (2.1). However, in our experience, it is very difficult to check the satisfiability of a practical model that contains LUTs using nonlinear interpolation, as shown in Section 2.4. Moreover, SMT-based test case generation is not applicable if a model includes mathematical functions, such as trigonometric or logarithmic functions, because they cannot be expressed in ordinary SMT language. In the following sections, we propose a novel abstraction and refinement approach that enables us to generate a test case effectively even if a model includes such mathematical functions.

2.2 Abstraction and Refinement of Mathematical Functions

For simplicity, we consider a model that includes only one mathematical function f, although it is possible to apply the proposed method to a model that includes multiple mathematical functions. By introducing an auxiliary variable $y^{(k)}$, the transition relation can be transformed as

$$T\left(v^{(k-1)}, v^{(k)}\right) := \tilde{T}\left(v^{(k-1)}, v^{(k)}\right) \wedge \left(y^{(k)} = f\left(x^{(k)}\right)\right), \tag{2.5}$$

where $x^{(k)}$ is a tuple of elements in $v^{(k)}$ representing the input variables of f, $\tilde{T}\left(v^{(k-1)}, v^{(k)}\right)$ denotes $T\left(v^{(k-1)}, v^{(k)}\right)$ in which $f\left(x^{(k)}\right)$ is replaced with $y^{(k)}$, and $y^{(k)}$ is appended to $v^{(k)}$. For example, $x^{(k)} = (v_1^{(k)}, v_2^{(k)})$, and $\tilde{T}$ is derived by replacing $f_{\mathrm{LUT}}\left(x^{(k)}\right)$ with $y^{(k)}$ in (2.2).

2.2.1 Abstraction of Mathematical Functions

We denote the input domain and the output domain of f by D_x and D_y, respectively. For a region $X \subseteq D_x$, we define an abstraction of f as follows:

Definition 1 Let $\mathcal{X} = \{X_1, \ldots, X_m\}$ be a set such that $X_i \subseteq D_x$ $(i = 1, \ldots, m)$ and

$$\bigcup_{i=1,\ldots,m} X_i = X,$$

where m is a parameter. Let $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$ be a set such that $Y_i \subseteq D_y$ $(i = 1, \ldots, m)$ and

$$x \in X_i \Rightarrow f(x) \in Y_i \tag{2.6}$$

for every X_i in $\mathcal{X}$. The formula

$$T_f(\mathcal{X}, \mathcal{Y}, k) := \bigwedge_{i=1}^{m} \left\{ x^{(k)} \in X_i \Rightarrow y^{(k)} \in Y_i \right\}. \tag{2.7}$$

is called an abstraction of f for $X \subseteq D_x$.

It is important to define X_i and Y_i such that the resulting T_f is easily treated by SMT solvers. In this study, we assume that X_i and Y_i are in the following simple box forms, Cartesian products of intervals:

$$X_i = \{x \mid x_l \diamond x \diamond x_u\}, \quad Y_i = \{y \mid y_l \diamond y \diamond y_u\}, \tag{2.8}$$

where $\diamond$ is either $\leq$ or $<$, and $\{x_l, x_u, y_l, y_u\}$ are constants. Figure 2.2 illustrates an example of an abstraction of f. We can solve the forumula Φ in (2.4) using an SMT solver that can handle difference logic [4] by replacing $y^{(k)} = f(x^{(k)})$ in (2.5) with T_f. It is worth mentioning that there is no need to prepare an SMT solver that can handle the mathematical function f directly.

It is important to estimate $(\mathcal{X}, \mathcal{Y})$ such that f is abstracted effectively for a given m. Figure 2.3 illustrates two different abstractions with $m = 10$ for the same function that increases exponentially. In Figure 2.3a, the input region is divided into equal intervals, while in Figure 2.3b it is divided

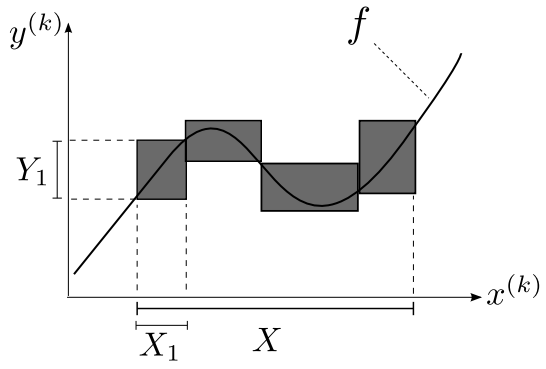
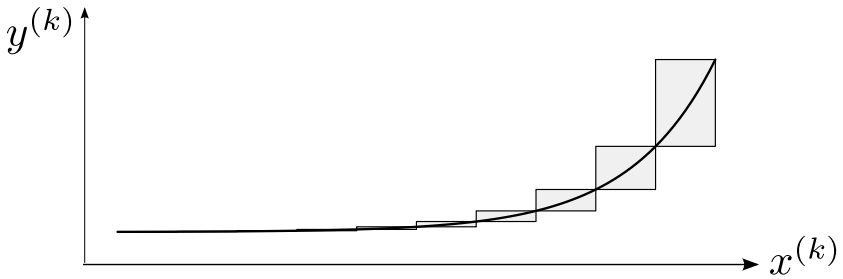
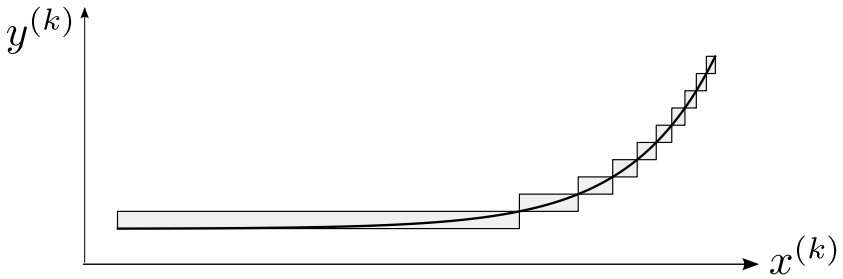


Figure 2.2: Example of an abstraction of f for $x^{(k)} \in X$, illustrated by the grey region.



(a) The input region is divided into equal intervals.



(b) The input region is divided considering the output.

Figure 2.3: Examples of abstraction with $m = 10$.

considering the output $y^{(k)}$. It is in general very difficult to determine which abstraction is better for test-case generation, because we do not know in advance in which part the actual solution (x^*, y^*) exists. However, it is reasonable to use the abstraction in Figure 2.3b rather than that in Figure 2.3a from the point of view that the abstraction error is averaged over the input. In Section 2.3, we propose a general method for estimating such an abstraction automatically for a given function, which can have multiple input, based on sampling and machine learning.

The proposed method is related to interval analysis [41, 51], in which Equations (2.6) and (2.8) are also considered. The biggest difference between the proposed method and interval analysis is as follows. The objective of interval analysis is to estimate the range of y accurately from $y = f(x)$ and $x \in X$, while the aim of the proposed method is to abstract the relationship between the input and the output of a given function f in the form of Equation (2.7). Although some methods were proposed for dividing the input region in interval analysis, such as sub-paving[41], the purpose of such methods is to overcome the dependency problem and to improve the accuracy of the output range estimation.

Piecewise-linear approximation is another possible approach for abstracting f. However, this approach has a fatal drawback in that it is impossible to conclude that the original formula is unsatisfiable when the abstracted formula becomes unsatisfiable, because piecewise-linear approximation is not an over-approximation of the original function in most cases. In contrast, the proposed method can prove the unsatisfiability of the original formula, because the abstraction becomes an over-approximation of f provided that (2.6) holds.

2.2.2 Focus and Refinement of Abstraction

We define a focus and a refinement of an abstraction as follows.

Definition 2 An abstraction $T_f(\mathcal{X}', \mathcal{Y}', k)$ is a focus of $T_f(\mathcal{X}, \mathcal{Y}, k)$ on $X_i \in \mathcal{X}$ iff the following holds:

$$\bigcup_{X' \in \mathcal{X}'} X' = X_i.$$

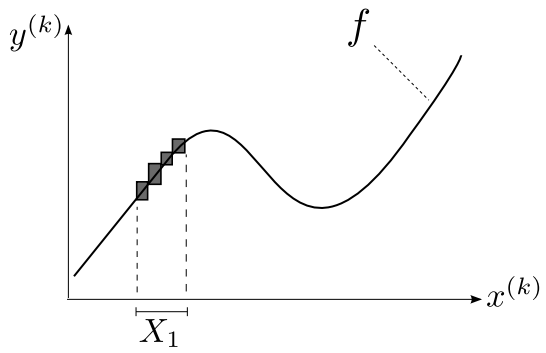


Figure 2.4: Example of a focus of the abstraction in Figure 2.2 on X_1.

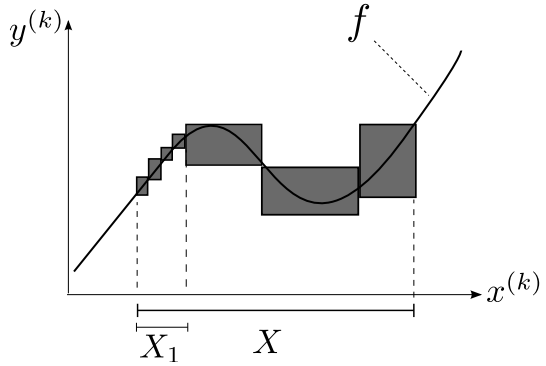


Figure 2.5: Refinement of the abstraction in Figure 2.2 with the focus in Figure 2.4.

Definition 3 A one-step refinement of $T_f(\mathcal{X}, \mathcal{Y}, k)$ is an abstraction that is obtained by replacing the X_i part of $T_f(\mathcal{X}, \mathcal{Y}, k)$ with a focus on X_i. A refinement of $T_f(\mathcal{X}, \mathcal{Y}, k)$ is an abstraction that is obtained by applying one-step refinements repeatedly to $T_f(\mathcal{X}, \mathcal{Y}, k)$.

An example of a focus of the abstraction in Figure 2.2 is shown in Figure 2.4. Figure 2.5 illustrates the refinement of the abstraction in Figure 2.2 with the focus in Figure 2.4.

2.2.3 CEGAR-style Strategy

We propose an algorithm in the style of CEGAR [22] that repeats the focusing and refinement of an abstraction based on the previous assignment. The

proposed algorithm is shown in Algorithm 1. A formula Φ_a is first composed by replacing T in (2.4) with $\tilde{T}$ in (2.5) as shown in Line 1 of Algorithm 1. $X^{(k)}$ $(k = 1, \ldots, K)$ are constraints on the input region of f in the k-th step. Note that the subscript k denotes the execution step of the model, as mentioned in Section 2.1. $X^{(k)}$ is initialized to $X_0 \subseteq D_x$ for all k (Line 2). X_0 may be given manually or estimated automatically by using a technique such as interval arithmetic [51]. Note that each of $X^{(k)}$ $(k = 1, \ldots, K)$ is focused or reset to X_0 in the main loop.

In the main loop, an abstraction of f inside $X^{(k)}$ is calculated by ABSTRACT, the details of which are explained in Section 2.3, and the abstraction is appended to Φ_a (Line 6). The constraints on the input region of f are added to Φ_a, resulting in Φ' (Line 8). Then, the satisfiability of Φ' is checked by an SMT solver. If Φ' is satisfiable, a test case is derived from the assignment, which is obtained by the SMT solver, to the input variables of the system. The original software, that is, the software without abstraction, is executed with the obtained test case. If the constraint P is satisfied in the execution, the test case is returned and Algorithm 1 terminates. If the test case does not satisfy the constraint P in the original software, which means that the obtained test case is *spurious*, the abstraction is focused on $X_i^{(k)}$ that includes $\tilde{x}^{(k)}$, where $\tilde{x}^{(k)}$ is the assignment to the input of f obtained by the SMT solver (Line 18). Note that the region on which the abstraction is focused may differ depending on k. The focusing is repeated until the true solution is obtained or Φ' becomes unsatisfiable. The advantage of using a focus is that the computation time of checking the satisfiability of Φ' tends to be short after each focusing, because the input region of f is specified tightly. Figure 2.6a illustrates the abstraction of f and assignments $\tilde{x}^{(k)}$ for $k = 1, 2, 3$ obtained by the SMT solver. Figure 2.6b shows the focused abstraction in the next loop iteration.

If Φ' becomes unsatisfiable and $X^{(k)} \neq X_0$ for some k, the abstraction of f is refined using the focuses obtained thus far, and the input region of f is reset to X_0. It is guaranteed that the same spurious test cases encountered before by the refinement will never be obtained. Figure 2.6c illustrates the abstraction after the refinement. If Φ' is unsatisfiable and $X^{(k)} = X_0$ for all k, it is proven that there is no test case that satisfies the property when

the input region of f is X_0.

In practice, it is better to store the result of ABSTRACT(f, $X^{(k_1)}$) and reuse it for $k = k_2$ if $X^{(k_1)} = X^{(k_2)}$. In addition, it is recommended that the redundant clauses that are related to the abstraction outside $X^{(k)}$ be removed from Φ' before checking the satisfiability of Φ'.

2.2.4 Limitations

The proposed method is effective for properties containing interval constraints such as Equation (2.3). However, it may not terminate when the given property consists of equality constraints, such as $v_4^{(K)} == 10$, and the variable that appears in the equality constraint includes an error caused by the abstraction, because the refinement may occur infinitely in this case. Note that the proposed method can handle properties with equality constraints provided that the variable in the constraint does not contain abstraction errors, even if the system includes a function to be abstracted. For example, a test case that satisfies "Out1 is equal to 25" for the model in Figure 2.7 can be generated by the proposed method, in which the block 2-D LUT is abstracted.

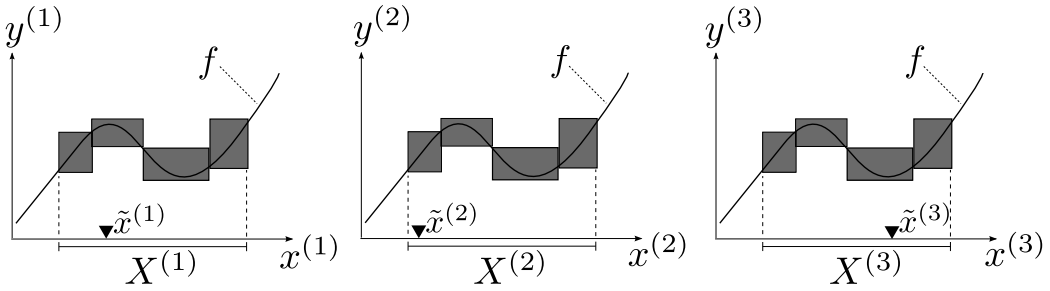
2.3 Machine Learning-based Abstraction

One purpose of machine learning methods is to generate a model $\tilde{f}$ that predicts the output of f as accurately as possible. On the other hand, a good abstraction for f in our context is $(\mathcal{X}, \mathcal{Y})$ in which Y_i are as tight as possible. From the analogy between these two points, it is expected that a good abstraction is estimated with the help of machine learning. In this section, we propose a general method to estimate an abstraction $(\mathcal{X}, \mathcal{Y})$ of f on $X \subseteq D_x$, which is used as ABSTRACT in Algorithm 1.

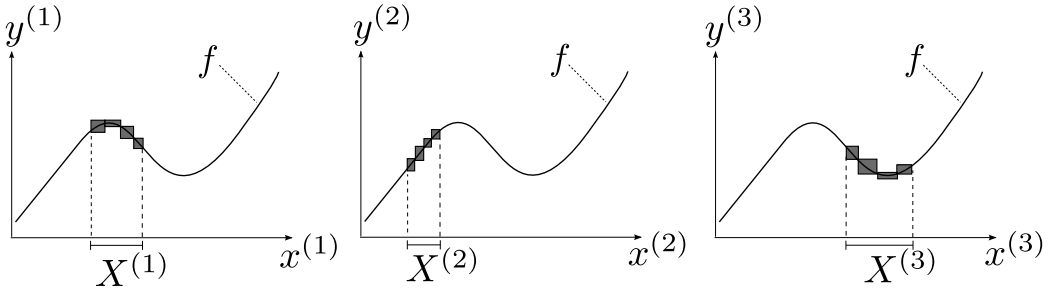
In the proposed method, an abstraction $(\mathcal{X}, \mathcal{Y})$ of f is estimated in two steps. In the first step, we estimate $\mathcal{X}$, which is a partition of X, by means of sampling and machine learning, in which X is divided into sub-regions X_i $(i = 1, \ldots, m)$ so that the value of f is distributed distinctly in each sub-region. In the second step, Y_i $(i = 1, \ldots, m)$ are calculated so

Algorithm 1 Abstraction and refinement of mathematical functions for SMT-based test-case generation.

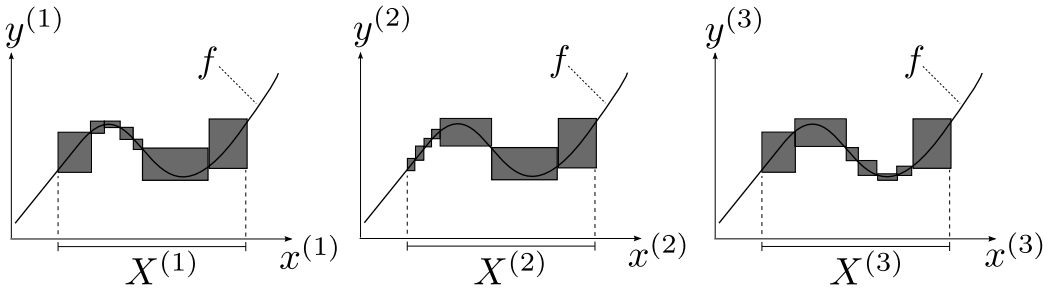
$\Phi_a \leftarrow I \wedge \bigwedge_{k=1}^{K} \tilde{T}(v^{(k-1)}, v^{(k)}) \wedge P$ ▷ Formula for BMC except for clauses with f
$X^{(k)} \leftarrow X_0$ for $k = 1, \ldots, K$ ▷ Initialize the input region of f
loop
  for $k = 1$ to K **do**
    $(\mathcal{X}^{(k)}, \mathcal{Y}^{(k)}) \leftarrow \textsc{abstract}(f,\, X^{(k)})$ ▷ Calculate an abstraction of f on $X^{(k)}$
    $\Phi_a \leftarrow \Phi_a \wedge T_f(\mathcal{X}^{(k)}, \mathcal{Y}^{(k)}, k)$ ▷ Append the obtained abstraction
  end for
  $\Phi' \leftarrow \Phi_a \wedge \bigwedge_{k=1}^{K} \{x^{(k)} \in X^{(k)}\}$ ▷ Add the constraints on the input region of f
  Check the satisfiability of Φ'.
  if Φ' is satisfiable **then**
    Derive a test case from the assignment to the input variables of the system.
    Execute the original software with the obtained test case.
    if the constraint P is satisfied in the execution **then**
      Return the test case and break the loop. ▷ Test-case generation is successful
    else
      for $k = 1$ to K **do**
        Let $\tilde{x}^{(k)}$ be the assignment to the input of f in the k-th step.
        $X^{(k)} \leftarrow X_i^{(k)} \in \mathcal{X}^{(k)}$ such that $\tilde{x}^{(k)} \in X_i^{(k)}$ ▷ The input sub-region to focus
      end for
    end if
  else
    if $X^{(k)} = X_0$ for all k **then**
      Return "unsatisfiable" and break the loop. ▷ Test-case generation is impossible
    else
      $X^{(k)} \leftarrow X_0$ for $k = 1, \ldots, K$ ▷ Reset the input region of f
    end if
  end if
end loop



(a) Assignments to the input of f in the k-th step ($k = 1, 2, 3$) are shown as $\tilde{x}^{(k)}$, which are obtained by solving Φ' with an SMT solver.



(b) The abstraction is focused on the sub-region that includes the previous assignment $\tilde{x}^{(k)}$ for each k. The focusing is repeated until the true solution is obtained or Φ' becomes unsatisfiable.



(c) If Φ' becomes unsatisfiable, the abstraction is refined for each k using the focuses obtained thus far.

Figure 2.6: Focusing and refinement of abstraction.

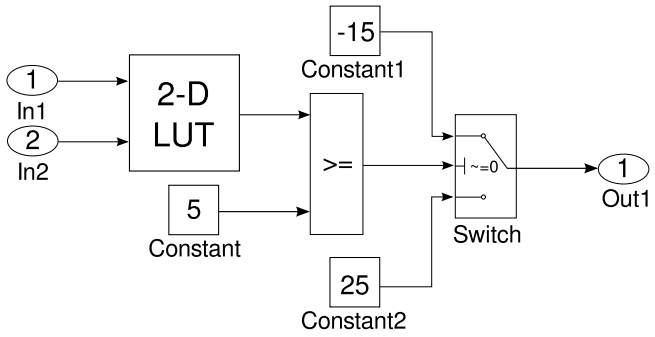


Figure 2.7: Example of a Simulink® model with a 2-D look-up table and a switch block.

that (2.6) holds for X_i obtained in the first step. In the following sections, we explain the details of the proposed method.

2.3.1 Estimating a Partition of an Input Region

Sampling

We assume that f is executable, that is, we can evaluate $f(x)$ for $x \in D_x$. In the proposed method, we first generate a data set $\mathcal{D}$ such that

$$\mathcal{D} = \left\{ (x^i, y^i) \;\middle|\; x^i \in X;\ y^i = f(x^i);\ i = 1, \ldots, N \right\}, \tag{2.9}$$

where N is the number of samples. On generating $\mathcal{D}$, x^i may be sampled randomly or taken as grid points in X. Figure 2.8 shows an example of $\mathcal{D}$ for

$$\begin{aligned} &f(x_1, x_2) = \sqrt{x_1}\log(x_2 + 1), \\ &X = \{40 \le x_1 \le 85,\ 0 \le x_2 \le 3000\}, \end{aligned} \tag{2.10}$$

where x^i are sampled randomly inside X.

Inducing a Partitioning Rule as a Decision Tree

We estimate $\mathcal{X}$ by applying the *decision tree induction* method [14] to $\mathcal{D}$, in which x and y are considered to be predictor variables and a response variable, respectively. From among many machine learning methods, we

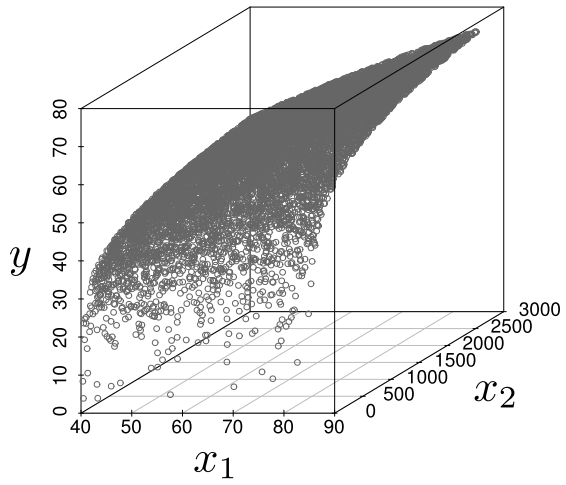


Figure 2.8: Example of $\mathcal{D}$.

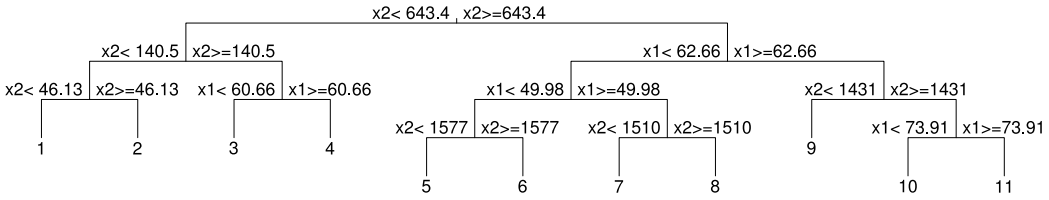


Figure 2.9: Decision tree induced from $\mathcal{D}$.

chose decision tree induction because it allows one to estimate $\mathcal{X}$ in which X_i are given in the form of (2.8). For example, Figure 2.9 shows a decision tree that is induced from $\mathcal{D}$ in Figure 2.8. The tree represents a partitioning rule of the input space of f. For example, Leaf 1 in Figure 2.9 represents the region

$$(x_2 < 643.4) \wedge (x_2 < 140.5) \wedge (x_2 < 46.13) \equiv x_2 < 46.13.$$

The partitioning rule is induced so that the dependent variable y is distributed distinctly in each partitioned region. We can derive $\mathcal{X}$ by dividing X into sub-regions using the obtained partitioning rule. Figure 2.10 shows the partition that is obtained by dividing X using the decision tree in Figure 2.9.

Note that the number of the leaf of an induced tree corresponds to m in (2.7). We can control the size of a tree to be induced by tuning a thresh-

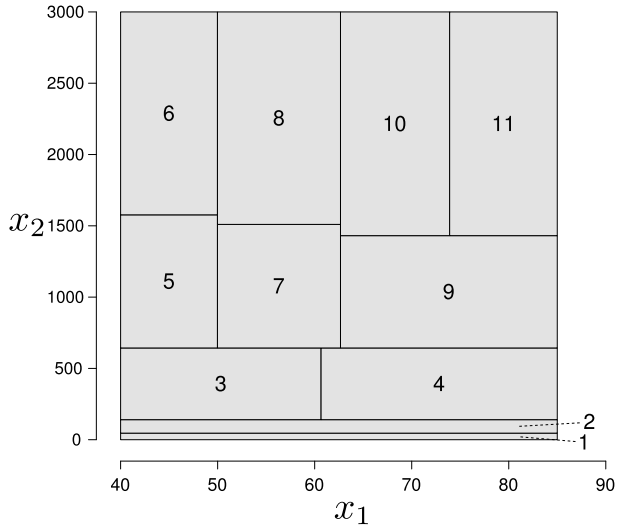


Figure 2.10: Partition of X by the decision tree in Figure 2.9.

old parameter of the decision tree induction. In addition, we can prune an induced tree to obtain a smaller one.

2.3.2 Output Range Calculation

We estimate $\mathcal{Y}$ such that (2.6) holds with respect to $\mathcal{X}$ obtained in the previous section. It is necessary to obtain as small a Y_i as possible for $X_i \in \mathcal{X}$ to achieve a tight abstraction. To compute Y_i strictly, we need the maximum and minimum of f under the assumption that the input region is restricted to X_i.

The estimation is easily executed for functions increasing or decreasing monotonically, such as logarithmic functions and exponential functions, or multiplications of them in which no variable is shared, like the function in (2.10). Then, we can compute the maximum and minimum of f based on the values of f at the vertices of X_i. Note that we can apply this procedure provided that a function is monotone in X_i. If f is a trigonometric function, the maximum and minimum of f can be computed in closed form. The maximum and minimum can also be computed simply when f is a function implemented as a look-up table using bilinear interpolation. Then, we need only the values of f at the grid points of the look-up table inside X_i and

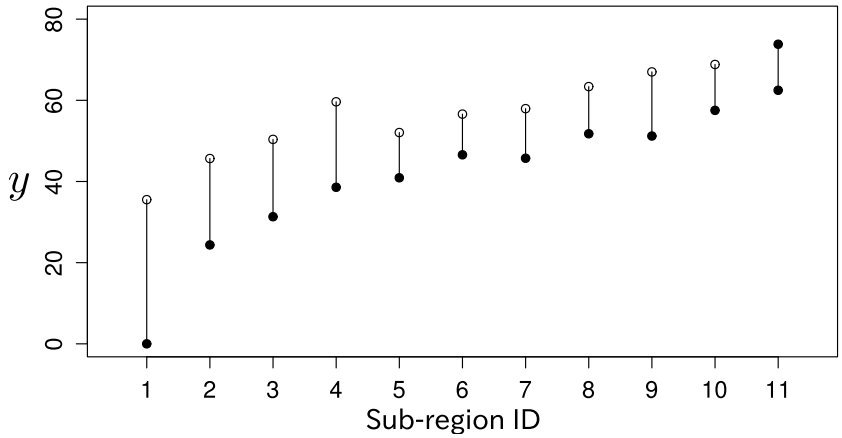


Figure 2.11: Calculated output range for each sub-region in Figure 2.10.

at the intersections of the grids and X_i, because f is monotone within each grid.

Another class of functions to which we can easily apply the estimation method consists of 1-D convex functions, the global minimum value of which can be computed analytically, because it is easy to compute the exact value of the minimum of f in X_i. If f is convex and the dimensionality of the input is more than one, we can utilize non-linear optimization methods, e.g., sequential quadratic programming [8], to estimate the minimum of f in X_i. However, we have to be aware that a calculation error may exist in the estimated minimum value, because non-linear optimization methods often calculate the minimum value numerically and the value might not be exact.

Figure 2.11 shows the output range that is calculated based on the values of f in (2.10) at the vertices of each sub-region in Figure 2.10. Whether the ends of the output range are included or not can be determined based on whether the vertices that achieve the maximum or minimum are included in the sub-region or not.

Limitations

In general, it is difficult to compute $\mathcal{Y}$ strictly if f is highly complicated or even black-box in the worst case. A possible solution to this problem is to estimate $\mathcal{Y}$ based on samples such as y^i in $\mathcal{D}$. However, it is important to notice that the abstraction thus obtained is no longer appropriate

in terms of the ordinary BMC, because there is no guarantee that $(\mathcal{X}, \mathcal{Y})$ over-approximate f. In other words, it is impossible to prove that there is no test case that satisfies the property even if the abstracted formula is proved to be unsatisfiable, unless the abstraction is an over-approximation of f.

The good news is that we can utilize the abstraction, which does not necessarily over-approximate f, for the purpose of test-case generation, because the main objective is not to prove that there is no test case, but rather to find a test case that satisfies the property.

2.4 Experimental Results

In this section, we present our experimental results. The experiment was conducted on an Intel® Core® i7/3.20 GHz machine with 64 GB RAM, running 64-bit Ubuntu 12.04.

2.4.1 Experimental Model

In the experiment, we used a Simulink® model taken from the automotive industry. Figure 2.12 illustrates the outline of the model. The model has four inputs and two outputs. The range of each input is specified as $[-1, 1]$. The initial value of each unit delay block is set to zero. The model has three 2-D LUTs with the bilinear interpolation (2.1), named LUT A, LUT B, and LUT C, the table sizes of which are 15×10, 10×8, and 10×8, respectively. The saturation block in Figure 2.12 rounds the first input into the range defined by the other inputs. Numeric variables in the model are defined as double-precision floating-point numbers.

The model includes two blocks related to hysteresis control. The relationship between the inputs and the output of "Hysteresis A" block in Figure 2.12 is given as

$$y_o = \begin{cases} x_i & \text{if } x_i < x_s, \\ x_s & \text{if } x_s \leq x_i < x_s + 0.25, \\ x_i - 0.25 & \text{if } x_s + 0.25 \leq x_i, \end{cases}$$

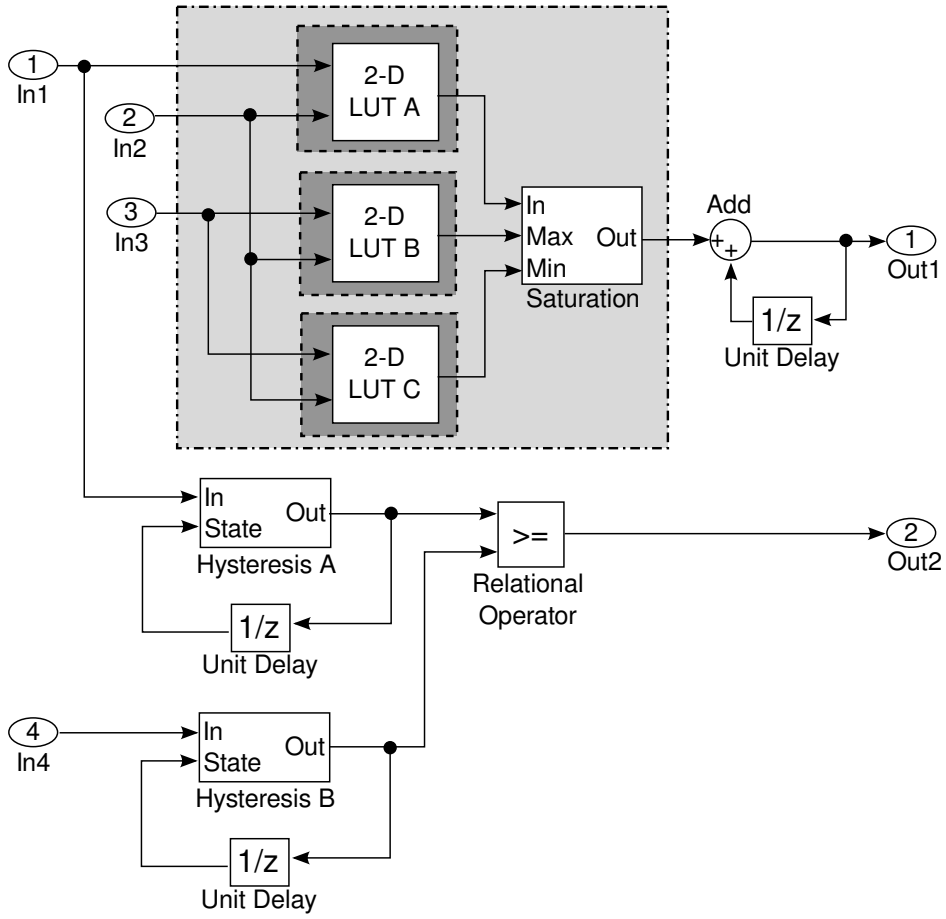


Figure 2.12: Experimental model.

where x_i, x_s, and y_o denote the variables related to In, State, and Out ports of "Hysteresis A" block, respectively. In addition, the relationship between the inputs and the output of "Hysteresis B" block is given as

$$y_o = \begin{cases} x_i + 0.25 & \text{if } x_i < x_s - 0.25, \\ x_s & \text{if } x_s - 0.25 \leq x_i < x_s, \\ x_i & \text{if } x_s \leq x_i. \end{cases}$$

In the situation where the model in Figure 2.12 is repeatedly executed K times, we attempted to generate a test case that satisfies two conditions simultaneously: 1) Out1 is within $[0.99, 1]$ in the K-th execution, 2) Out2 is always true. The value of K determines the difficulty of the test-case

generation: the larger K is, the more difficult is the test-case generation.

Although the model in Figure 2.12 may seem rather small and simple, it is very difficult to generate test cases based on the ordinary SMT-based method for $K \geq 5$, as shown in Section 2.4.4.

2.4.2 Abstraction Settings

In the experiment, the mathematical functions in Figure 2.12 were abstracted using one of the following two methods. The first is to abstract each LUT in the model separately, which is shown as the dark gray boxes with dash lines in Figure 2.12. In this case, we have three functions to abstract and each of them has two inputs. The second one is to abstract three LUTs and the saturation block simultaneously by viewing them as a function, which is shown as the light gray box with a dot-dash line in Figure 2.12. In this case, we have a function to abstract and the function has three inputs. Hereafter, we refer to the former and the latter as the *AbstEach* setting and the *AbstAll* setting, respectively.

To investigate the impact of parameter m, it was set to one of $\{25, 50, 100\}$ in both the AbstEach and the AbstAll setting. Every time ABSTRACT in Algorithm 1 is invoked in the proposed method, we generate $N = 50000$ samples randomly as $\mathcal{D}$ in (2.9). For estimating an abstraction, the output range of a function to be abstracted is estimated based on the output values on the grid points of each LUT in the AbstEach setting, as mentioned in Section 2.3.2. In the AbstAll setting, the output range is estimated based on samples, as mentioned in Section 2.3.2, because it is difficult to compute the output range analytically in this setting.

2.4.3 Implementation

We implemented the proposed algorithm using Matlab® [70] and R [55], in which `R.matlab` [7] is used for the communication between Matlab® and R, and `rpart` [71] is used for the decision tree induction. We employed Z3 (version 4.3.2) [27] as an SMT solver. The reason why we chose Z3 from among many SMT solvers is that it is known to be highly efficient [4] and can handle various background theories, including non-linear arithmetic, which

Table 2.2: Computation time of generating test cases without abstraction (in sec).

	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$
Without Abstraction	0.1	0.1	4.7	1653.3	TO
	$K=6$	$K=7$	$K=8$	$K=9$	$K=10$
	TO	TO	TO	TO	TO

TO: timeout (3600 seconds)

enables us to model the bilinear interpolation (2.1) as it is. Furthermore, Z3 can handle difference logic, which is required by the proposed method, as mentioned in Section 2.2.1. In the experiment, we invoked Z3 with its default options. The computation was aborted if the computation time exceeded the timeout limit, which we set to 3600 sec.

Although the model includes floating-point variables and floating-point arithmetic, we approximately regarded them as real variables and real arithmetic in the SMT formulation in this experiment. This is because handling floating point variables in the SMT formulation leads to high complexity. Note that such a strategy is allowed only for the purpose of test-case generation. It is necessary to confirm that the generated test case truly satisfies the required constraints in the original Simulink model.

2.4.4 Results

Without Abstraction

For comparison, we first generated test cases by solving the SMT formula (2.4) in which three LUTs are not abstracted and the bilinear interpolation is formulated directly using non-linear arithmetic. Table 2.2 shows the computation time of generating test cases by solving the formula with Z3 for $K = 1, \ldots, 10$. We observe that test cases are successfully generated if K is small. However, the computation time increases immediately as K increases, and it reaches the timeout limit for $K \geq 5$.

Results of the Proposed Method

The results of the proposed method vary depending on the random seed used to generate samples for the decision tree induction in ABSTRACT. Therefore, we repeated the experiment 10 times with different random seeds to evaluate each setting of the proposed method. Table 2.3 shows the average computation time of the proposed method over the 10 trials. In Table 2.3, the numbers in parentheses are the results of the average time over a subset of the 10 trials, because some computations were timed-out. The number of times the computation exceeded the time limit within the 10 trials is also shown in Table 2.3. In the AbstAll setting, test-case generation was always successful before timeout for $K = 1, \ldots, 10$. No obvious difference may be found between the results with different values of m in the AbstAll setting. In the AbstEach setting, test cases are most frequently generated before timeout, although the timeout rate is relatively high with $m = 100$ for $K \geq 6$.

A comparison of Table 2.3 and Table 2.2 shows that the proposed method can generate test cases much faster than the method without abstraction for $K \geq 4$. However, for $K = 1, 2, 3$ the method without abstraction is faster than the proposed method. This is because the proposed method consumes time calculating the abstraction in addition to solving SMT formulas. Therefore, on the one hand, when the size of the problem is sufficiently small it is better to formulate the problem directly without abstraction, while on the other the proposed method can generate test cases much faster than the direct method when the problem is rather large.

In Table 2.3, it can be seen that overall the AbstAll setting is superior to the AbstEach setting, the reason for which can be seen in Table 2.4. The table 2.4 shows the average of the number of times that Φ' becomes unsatisfiable in Algorithm 1 over the 10 trials. In the AbstEach setting, Φ' is more likely to be unsatisfiable, because the three LUTs in the model are independently abstracted, although they share some of their inputs. In such a case, an inconsistency tends to occur between the assignments of the inputs of LUTs after the input region of these LUTs are focused. However, Φ' becomes unsatisfiable much less frequently in the AbstAll setting, because the three LUTs are packed into a function and abstracted together.

Table 2.3: Average computation time of the proposed method over the 10 trials (in sec). The numbers in parentheses are the results calculated over a subset of the 10 trials, because in some cases the calculation time reached timeout. The number with TO is the number of times timeout occurred within the 10 trials.

	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	$K=7$	$K=8$	$K=9$	$K=10$
AbstEach ($m=25$)	25.6	80.7	252.2	258.1	932.5	684.0	604.4	1137.4	1378.0	(1877.0) TO:1
AbstEach ($m=50$)	30.6	114.6	159.9	285.3	742.6	882.7	(1017.2) TO:1	(493.9) TO:1	(1086.1) TO:3	(1039.5) TO:1
AbstEach ($m=100$)	15.1	117.2	195.7	357.1	342.7	(807.1) TO:2	(831.2) TO:2	(1237.1) TO:2	(1213.1) TO:1	(1191.3) TO:4
AbstAll ($m=25$)	19.2	52.7	80.2	116.1	214.2	402.5	323.0	271.5	563.1	302.0
AbstAll ($m=50$)	15.9	31.4	78.5	89.3	92.7	228.0	165.1	169.1	353.8	362.1
AbstAll ($m=100$)	12.3	36.0	51.8	155.3	153.2	143.5	323.9	320.2	161.4	339.7

Table 2.4: Average of the number of times that Φ' becomes unsatisfiable over the 10 trials. Results in parentheses are calculated over a subset of the 10 trials, omitting the timed-out results.

	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$	$K = 7$	$K = 8$	$K = 9$	$K = 10$
AbstEach ($m = 25$)	0.2	2.0	5.0	4.2	11.3	7.6	5.6	8.7	9.4	(11.6)
AbstEach ($m = 50$)	0.8	2.7	3.5	4.0	7.4	6.8	(7.2)	(3.2)	(6.7)	(5.9)
AbstEach ($m = 100$)	0.0	2.8	3.2	3.6	3.6	(5.9)	(4.5)	(5.5)	(4.3)	(4.2)
AbstAll ($m = 25$)	0.0	0.4	0.3	0.4	0.9	1.3	1.0	0.4	1.5	0.5
AbstAll ($m = 50$)	0.0	0.0	0.6	0.3	0.2	0.8	0.4	0.1	0.7	0.5
AbstAll ($m = 100$)	0.0	0.2	0.1	0.9	0.4	0.3	0.8	0.9	0.0	0.5

2.5 Related Work

Brady et al. [13] proposed an automatic approach for function abstraction to provide formal verification of hardware designs. In their method, a function in a design is replaced with an uninterpreted function on the condition that the replacement does not lead to spurious counterexamples. Then, the design is verified by using an SMT solver that is capable of handling uninterpreted functions. The abstraction of functions including uninterpreted functions is useful for verification purposes, but in most cases is not appropriate for the purpose of generating test cases, because the concrete relationship between the input and the output of a function is not expressed at all if the function is abstracted as an uninterpreted function. In contrast, the method we propose in this chapter abstracts a function by expressing the relationship between the input and the output of the function approximately, while the expression is partially refined as many times as necessary in the abstraction-refinement scheme.

Yeolekar et al. [75] proposed a method for test-case generation in which bounded model checking is used as the basis. In their method, a function is replaced with likely invariants of the function that are estimated by using the Daikon tool [31]. This tool estimates invariants based on the execution traces of the function by comparing them to a set of invariant templates, such as constants $x = k$, intervals $a \leq x \leq b$, linear relationships $y = ax + b$, and so on. In their approach, we noticed that it is expected that good invariants are estimated if the behavior of the function is relatively simple, but the obtained invariants may be coarse in the case where the function is highly non-linear and complicated. Our method overcomes this problem by abstracting a function in a form that can express the complicated behavior of the function.

Some methods reported in the literature utilize machine learning methods in the abstraction-refinement scheme for model checking. Clarke et al. [23] used a machine learning method to determine which part of an abstraction should be refined. Brady et al. [12] proposed a method that uses machine learning to find which part of an abstraction leads to spurious counterexamples. The abstraction of that part is then refined with priority. Both of these

methods use machine learning to improve the refinement strategy. However, we apply machine learning to obtain an abstraction of a mathematical function.

For generating test cases of embedded control software that includes complex mathematical functions, Satpathy et al. [61] proposed a method based on pattern-guided heuristics. In addition, Borges et al. [10] proposed a method that utilizes both meta-heuristics and interval solving. Such heuristic search-based methods are effective if the number of variables is small and the objective function is smooth. However, it is generally difficult to define smooth objective functions for embedded control software that includes both continuous and discontinuous operations. The proposed method overcomes this problem by expressing operations that are compatible with SMT solvers directly in the SMT formula.

In concolic testing [37, 64], the concrete and the symbolic execution [46] are carried out repeatedly. If the path constraint obtained by the symbolic execution includes a function that is intractable using SMT solvers, the function is under-approximated using the result of the previous concrete execution. Concolic testing generates test cases that explore distinct program paths serially. It is difficult to generate a test case that explores the targeted program path directly by using concolic testing. The proposed method can generate such a test case directly in the scheme of bounded model checking in which intractable functions are over-approximated.

2.6 Summary

In this chapter, we proposed a novel approach for test-case generation of software that includes mathematical functions, such as trigonometric functions, logarithmic functions, functions implemented as look-up tables with non-linear interpolation, and so on. In the proposed method, mathematical functions are abstracted automatically by means of machine learning, and the abstraction is refined adaptively. The proposed method enables us to handle software with such mathematical functions in the scheme of bounded model checking with SMT solvers. It is sufficient to prepare an SMT solver that can handle difference-logic. The experimental results showed that using

the proposed method it is possible to generate test cases for an example of embedded control software that is intractable using an ordinary SMT solver, in a practicable time.

Part II

Anomaly Detection

Chapter 3

One-Class Classification using Binary Decision Diagrams

A one-class classification problem is to estimate a classifier that distinguishes a target class data from the other class data from a given training data set. The problem is a type of unsupervised learning problems because the training data set is assumed to be one-sided, in other words, it only contains the target class data, but it may contain some noise data that do not belong to the target class. A lot of research have been done on one-class classification problems and various methods have been proposed to learn a one-class classifier [19, 69]. Generally, a classifier has parameters that need to be tuned depending on each problem. Some of these parameters have a strong influence on the final performance and have to be tuned carefully. We refer to such parameters as *magic parameters* in reference to Tax [69]. One of the serious issues in the one-class classification problem is that it is very difficult to tune magic parameters. Although the cross-validation technique is often used to tune parameters of a classifier, it is not straightforward to apply the technique when the training data set is one-sided. In this chapter, we propose a novel approach for the one-class classification problem. One of the key features of the proposed method is that the magic parameter of the proposed classifier can be tuned with a training data set that is one-sided. In this sense, the proposed method is *parameter-free* in reference to Keogh [45]. Another key feature of the proposed method is that a large training data set can be

dealt with very efficiently, because the computational complexity of learning the proposed classifier is approximately linear with regard to the number of training data.

In the proposed method, each datum in a training data set is converted into a logical formula after being truncated and is summed up to construct a Boolean function that corresponds to the region of the training data set. A binary decision diagram is used to represent and construct the Boolean function. Then the region is over-approximated based on its hierarchical local densities. The over-approximation can be done quite efficiently through direct manipulation of a binary decision diagram that represents the region. The proposed method has a threshold paremeter, which is the magic parameter of the proposed method, that adjusts the degree of the over-approximation. We propose a method to tune the parameter of the proposed classifier based on the minimum description length principle, in which the over-approximated region is considered to be a model that encodes the training data set.

The proposed method is evaluated with a synthetic data set and some realistic data sets. The experimental result indicates that a fine one-class classifier is learnt in a parameter-free manner by the proposed method, even though the training data set is non-linerly distributed and contains some noise data. One of the realistic data sets we used is the KDD Cup 99 data set [2] that is related to a network intrusion detection problem. The experimental results show that a classifier that detects the intrusion accurately is constructed parameter-freely by the proposed method. The detection accuracy achieved by the proposed method is equal to or better than that of the one-class support vector machine [63]. Moreover, the computational time for learning the proposed classifier is much shorter than that for the one-class support vector machine.

This chapter is organized as follows: Section 3.1 defines the problem considered in this chapter and intuitively describes the basic idea of the proposed method. Section 3.2 describes efficient implementation of the proposed method by using a binary decision diagram. Section 3.3 describes how to tune the parameter of the proposed method based on the minimum description length principle. In Section 3.4, the proposed method is compared with

other works. Section 3.5 shows some experimental results, and Section 3.6 summarizes this chapter.

3.1 Intuitive Explanation of the Proposed Method

In this section, we define the problem to be solved in this chapter and illustrate the basic idea of our solution. Note that the algorithms mentioned in this section are used only for explaining the idea and their efficient implementation are presented in Section 3.2.

3.1.1 Problem Setting

First, we difine terms required to explain the one-class classification problem. Let (x, y) be a data where $x = (x_1, \ldots, x_u)$ is a vector of continuous attributes and $y = (y_1, \ldots, y_v)$ is a vector of categorical attributes. The number of elements included in the domain of y_i is denoted by M_i $(i = 1, \ldots, v)$. Let D be a training data set that includes N data. The i-th data in D is denoted by $(x^{(i)}, y^{(i)})$ $(i = 1, \ldots, N)$. We assume that there are no missing values in D. Although all the data in D are unlabeled, it is assumed that a large part of the data in D belongs to a target class $\mathcal{A}$. The data that do not belong to the target class $\mathcal{A}$ in D are referred to as noise data. For example, let us consider the situation that we are treating a data set on network access analysis and x includes the duration time of a connection and y includes the protocol of a connection, such as *http* and *ftp*, for a network intrusion detection problem. Normal connection data are collected from the network as a training data set, where the normal connection is the target class. However, the collected data may contain some attacking connections as noise data.

The one-class classification problem is defined as follows:

Definition 4 Given a training data set D, the purpose of the one-class classification problem is to construct a one-class classifier *occ* such that

$$occ(x, y) = \begin{cases} +1 & \text{if } (x, y) \in \mathcal{A}, \\ -1 & \text{otherwise.} \end{cases}$$

For example, a classifier that distinguishes attacking connections from normal connections is learnt from the training data set mentioned above for a network intrusion detection problem.

3.1.2 Projection and Hashing of Data

Before explaining the proposed method, we define some notations. Let $\mathcal{H}$ be a u-dimensional hypercube whose side length is 2^m, where m is an arbitrary positive integer.

Definition 5 A function σ in $\mathbb{R}^u \to \mathbb{R}^u$ is called an *example normalizer* if $\sigma(x^{(i)}) \in \mathcal{H}$ for every $x^{(i)}$ $(i = 1, \ldots, N)$ in a training data set D. For all $x \in \mathbb{R}^u$, $\sigma(x)$ is denoted by z.

An example of σ is given as follows:

$$\sigma(x) = \left(\frac{scale(x_1) - x_{min}}{x_{max} - x_{min}} \left(2^m - \epsilon\right), \ldots, \frac{scale(x_u) - x_{min}}{x_{max} - x_{min}} \left(2^m - \epsilon\right) \right) \quad (3.1)$$

where ϵ is an arbitrary small positive number and

$$\begin{aligned} scale(x_i) &= \frac{x_i - \mu_i}{\delta_i} \quad (i = 1, \ldots, u), \\ x_{max} &= \max_{i=1,\ldots,N;\ j=1,\ldots,u} \left(scale(x_j^{(i)}) \right), \\ x_{min} &= \min_{i=1,\ldots,N;\ j=1,\ldots,u} \left(scale(x_j^{(i)}) \right), \end{aligned}$$

where μ_i and δ_i are the mean and the standard deviation of the i-th continuous attribute respectively, which are calculated from $\left\{ x^{(i)} \mid i = 1, \ldots, N \right\}$.

Definition 6 The *neighborhood function* ν is a function that returns a u-dimensional unit hypercube that subsumes $z \in \mathbb{R}^u$, which is defined as follows:

$$\nu(z) = [\lfloor z_1 \rfloor, \lfloor z_1 \rfloor + 1) \times \ldots \times [\lfloor z_u \rfloor, \lfloor z_u \rfloor + 1)$$

where $\lfloor \cdot \rfloor$ is the floor function.

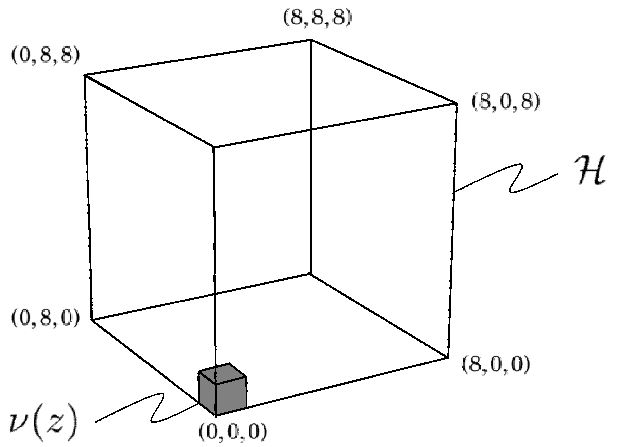


Figure 3.1: A hypercube $\mathcal{H} = [0, 2^m)^u$ and a unit hypercube $\nu(z)$, where $u = 3$, $m = 3$, and $z = (0, 0, 0)$.

Figure 3.1 shows an illustrative example of $\mathcal{H}$ and $\nu(z)$, where $u = 3, m = 3$, and $z = (0, 0, 0)$. Let ϕ be a perfect hash function that receives y as a key and returns a hash value. The range of ϕ may be assumed to be $\{1, \ldots, M\}$ without loss of generality because the range has $M = \prod_{i=1}^{v} M_i$ hash values from the definition of y. When a training data set D has no categorical attributes, it is assumed that $M = 1$ and $\phi(\cdot) = 1$.

3.1.3 Constructing the Initial Region Vector

In our approach, we first construct *the initial region vector* G from a given training data set D through Algorithm 2. In Algorithm 2, G_i is first initialized, and then, constructed as a u-dimensional region inside $\mathcal{H}$ that subsumes the projected data $\{z^{(j)} \mid \phi(y^{(j)}) = i,\ j = 1, \ldots, N\}$ for $i = 1, \ldots, M$. Then, G is returned as a set of G_i $(i = 1, \ldots, M)$.

Given the initial region vector G, a one-class classifier occ_G can be constructed as follows:

$$occ_G(x, y) = \begin{cases} +1 & \text{if } \nu\left(\sigma(x)\right) \subseteq G_{\phi(y)}, \\ -1 & \text{otherwise.} \end{cases} \tag{3.2}$$

However, occ_G works rarely if we use the initial region vector G obtained by Algorithm 2 directly, because G will be too coarse if we set m too small. On

Algorithm 2 Construction of the initial region vector G from a training data set D

Input: A training data set D and an example normalizer σ.

Output: The initial region vector G.

1: **for** $i = 1$ to M **do**
2: $\quad G_i \leftarrow \emptyset$
3: **end for**
4: **for** $j = 1$ to N **do**
5: $\quad z^{(j)} \leftarrow \sigma(x^{(j)})$
6: $\quad G_{\phi\left(y^{(j)}\right)} \leftarrow G_{\phi\left(y^{(j)}\right)} \cup \nu\left(z^{(j)}\right)$
7: **end for**
8: **return** $G = \{G_1, \ldots, G_M\}$

the other hand, G will be too fine if we set m too large. In our approach, we set m large enough, and over-approximate G obtained by Algorithm 2 according to its hierarchical local density, which is mentioned in the next section.

3.1.4 Over-approximating the Initial Region Vector

Let $G = \{G_1, \ldots, G_M\}$ be the initial region vector obtained by Algorithm 2. The over-approximation of G is defined as follows:

Definition 7 A region vector $\tilde{G} = \{\tilde{G}_1, \ldots, \tilde{G}_M\}$ is an over-approximation of the initial region vector $G = \{G_1, \ldots, G_M\}$ if the following condition is satisfied:

$$\forall y,\ \forall z, \quad z \in G_{\phi(y)} \ \Rightarrow\ z \in \tilde{G}_{\phi(y)}.$$

We propose Algorithm 3 that over-approximates G based on a threshold θ whose dimensionality is one and domain is $[0, 1]$. In Algorithm 3, for each i in $\{1, \ldots, M\}$, we first focus on inside $\mathcal{H}$, and if the volume ratio of $G_i \cap \mathcal{H}$ toward $\mathcal{H}$ is larger than or equal to θ, $\mathcal{H}$ is added to G_i, and the procedure is terminated. Otherwise, $\mathcal{H}$ is split into 2^u hypercubes $\mathcal{C}_k$ $(k = 1, \ldots, 2^u)$ of a size. Then the region inside $\mathcal{C}_k$ is focused recursively until $\mathcal{C}_k$ becomes a unit

Algorithm 3 Over-approximation of the initial region vector G with a threshold θ

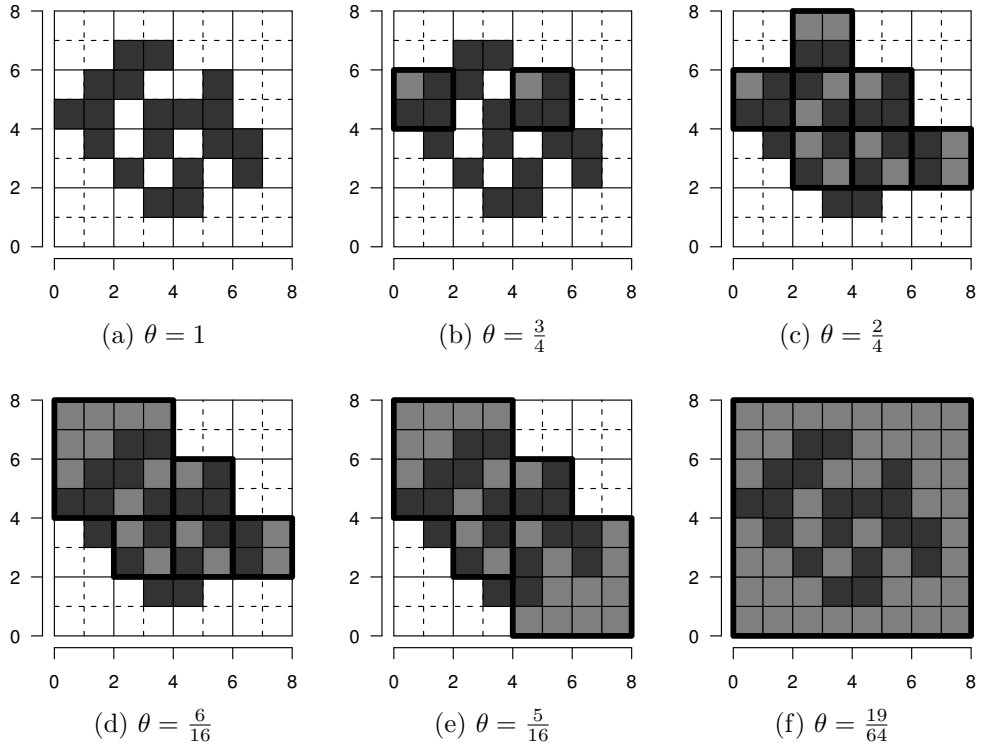
Input: The initial region vector G and a threshold θ.

Output: The over-approximated region vector G^θ.

for $i = 1$ to M **do**
  $G_i^\theta \leftarrow \textsc{OverApprox}(G_i, \mathcal{H}, \theta)$
end for
return $G^\theta = \left\{G_1^\theta, \ldots, G_M^\theta\right\}$

procedure $\textsc{OverApprox}(G_i, \mathcal{C}, \theta)$
  if $\mathcal{C}$ is a unit hypercube **then**
    return $G_i^\theta = G_i$
  end if
  if $\textsc{Density}(G_i, \mathcal{C}) \geq \theta$ **then**
    $G_i^\theta \leftarrow G_i \cup \mathcal{C}$
    return G_i^θ
  end if
  Split $\mathcal{C}$ into 2^u hypercubes, denoted by $\mathcal{C}_k$ $(k = 1, \ldots, 2^u)$, whose side length is half of that of $\mathcal{C}$.
  for $k = 1$ to 2^u **do**
    $G_i^\theta \leftarrow \textsc{OverApprox}(G_i, \mathcal{C}_k, \theta)$
  end for
  return G_i^θ
end procedure

procedure $\textsc{Density}(G_i, \mathcal{C})$
  return $\dfrac{\text{volume of } (G_i \cap \mathcal{C})}{\text{volume of } \mathcal{C}}$
end procedure



(a) $\theta = 1$ (b) $\theta = \frac{3}{4}$ (c) $\theta = \frac{2}{4}$

(d) $\theta = \frac{6}{16}$ (e) $\theta = \frac{5}{16}$ (f) $\theta = \frac{19}{64}$

Figure 3.2: An example of over-approximated results G^θ obtained by Algorithm 3 with the threshold $\theta = \frac{3}{4}, \frac{2}{4}, \frac{6}{16}, \frac{5}{16}, \frac{19}{64}$. The initial region corresponds to the region of (a), because no approximation occurs when $\theta = 1$.

hypercube or the volume ratio of $(G_i \cap \mathcal{C}_k)$ toward $\mathcal{C}_k$ is larger than or equal to θ. In the latter case, $\mathcal{C}_k$ is added to G_i. The over-approximated result G^θ is returned by Algorithm 3. Figure 3.2 shows an example of the initial region vector G, where $u = 2$, $m = 3$ and $M = 1$, and its over-approximated results obtained by Algorithm 3 with various θ. In Figure 3.2, the dark gray region means the initial region, the light gray region means the region added by Algorithm 3, and the bold square signifies a hypercube whose density is larger than or equal to a threshold θ. The smaller the threshold θ becomes, the larger the approximated region becomes monotonically, as can be seen in Figure 3.2. Note that $G^1 = G$, because no approximation occurs when $\theta = 1$. If θ is tuned properly, occ_G is expected to be a sound one-class classifier by using G^θ instead of G in Equation (3.2).

Although the above mentioned algorithms seem computationally complex, it is possible to implement them efficiently by using binary decision diagram-based techniques, which is mentioned in the next section. Furthermore, we propose a method for tuning parameter θ based on the minimum description length principle, which is mentioned in Section 3.3.

3.2 Implementation Based on Binary Dicision Diagrams

In this section, we propose efficient implementation of the proposed algorithms mentioned in Section 3.1 by using binary decision diagram-based techniques.

3.2.1 Binary Decision Diagram

A Binary decision diagram (BDD) is a compressed representation of a Boolean function. A reduced ordered BDD (ROBDD) is a class of BDD that has an important feature that it is canonical for a particular function and variable order. Hereafter, we refer to an ROBDD as a BDD, for short. A BDD also have an important feature that most logical operations between Boolean functions that are expressed as BDDs can be performed by polynomial-time algorithms [17]. For illustrative purposes, we consider Boolean function F, which is defined by a set of four Boolean variables $\{b_{11}, b_{12}, b_{21}, b_{22}\}$ as follows:

$$\begin{aligned} F = &\left\{b_{11} \wedge b_{21} \wedge b_{12}\right\} \vee \left\{b_{11} \wedge b_{21} \wedge \overline{b_{12}} \wedge \overline{b_{22}}\right\} \\ &\vee \left\{b_{11} \wedge \overline{b_{21}} \wedge b_{12} \wedge \overline{b_{22}}\right\} \vee \left\{\overline{b_{11}} \wedge b_{21} \wedge b_{12} \wedge \overline{b_{22}}\right\}. \end{aligned} \tag{3.3}$$

Figure 3.3 shows a BDD representation of F, where the variables are ordered as $b_{11} \prec b_{21} \prec b_{12} \prec b_{22}$. In Figure 3.3, square nodes, ellipsoidal nodes and double-squared nodes are referred to as terminal nodes, variable nodes and function nodes, respectively. Solid arrows, dashed arrows and double-lined arrows in the figure are referred to as true edges, false edges and root edges, respectively. Variable nodes are labeled with A, B, C, D, E and F and their corresponding variables are on the left side. For example, both Node B and

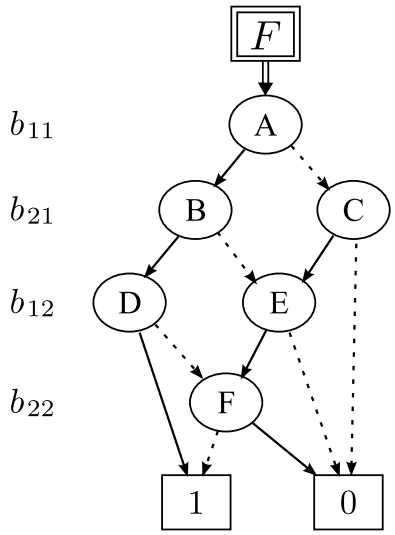


Figure 3.3: A BDD representation of F in Equation (3.3).

Node C correspond to b_{21}. A variable node that is directed by a root edge is called a *root node*. A path from the root node to the terminal 1 corresponds to a conjunction of literals. For example, the path that goes from the root node to the terminal 1 through Node A, Node B and Node D in Figure 3.3 corresponds to $b_{11} \wedge b_{21} \wedge b_{12}$. A Boolean function is given as a disjunction of all of such conjunctions.

In some implementation that are used to handle BDDs, such as CUDD [65], complement edges are used to express BDDs more efficiently. Although the BDD-based algorithm mentioned in the following sections cannot directly be applied when complement edges are used in BDDs, it is possible to adapt them to BDDs with complement edges by modifying the algorithm slightly. For the sake of simplicity, we explain our algorithms without complement edges in the following sections.

3.2.2 Constructing the Initial Boolean Function

In order to use a Boolean function for representing domains, we introduce a special function which codes every continuous data into a logical formula.

Definition 8 CodeZ is a function that codes data $z \in \mathcal{H}$ into a logical

Algorithm 4 Construction of the initial Boolean function F from a training data set D

Input: A training data set D and an example normalizer σ.

Output: The initial Boolean function F.

1: $F \leftarrow 0$
2: **for** $i = 1$ to N **do**
3: $\quad z^{(i)} \leftarrow \sigma\left(x^{(i)}\right)$
4: $\quad F \leftarrow F \vee \left\{\textsc{CodeY}(y^{(i)}) \wedge \textsc{CodeZ}(z^{(i)})\right\}$
5: **end for**
6: **return** F

formula. Each z_i is first truncated to an integer, and then, coded in the manner of an unsigned-integer-type coding by using m Boolean variables $\mathbb{B} := \{b_{ij} \mid i = 1, \ldots, u;\ j = 1, \ldots, m\}$, where b_{i1} is the most significant bit and b_{im} is the least significant bit.

For example, $\textsc{CodeZ}(z) = b_{11} \wedge \overline{b_{12}} \wedge b_{21} \wedge b_{22}$ if $z = (2.1, 3.7)$, where $m = 2$. In the proposed method, data z that is normalized by an example nomalizer σ, which is mentioned in Section 3.1.2, is coded by CodeZ. By using Boolean formulas, we can also treat categorical data. This means that we can combine a coding function for categorical data with CodeZ.

Definition 9 CodeY is a function that codes categorical data y into a logical formula, which is defined upon L Boolean variables $\mathbb{D} := \{d_1, \ldots, d_L\}$. For our convenience of explanation we define $\textsc{CodeY}(y) = 1$ if the given data have no categorical attribute.

For example, suppose that $y = (y_1, y_2)$ and the domain of y_1 and y_2 are {blue, red, yellow} and {ON, OFF}, respectively. By setting $L = 3$, the elements of the domains of y_1 and y_2 can be coded as $\{d_1 \wedge d_2,\ d_1 \wedge \overline{d_2},\ \overline{d_1} \wedge d_2\}$ and $\{d_3, \overline{d_3}\}$, respectively. If $y = (\text{red}, \text{ON})$, then $\textsc{CodeY}(y) = d_1 \wedge \overline{d_2} \wedge d_3$.

We propose Algorithm 4 that constructs *the initial Boolean function F* from a training data set D. In Algorithm 4, F is defined upon $L + mu$ Boolean variables. BDDs are used to represent Boolean functions and to perform logical operations in Algorithm 4. It is assumed that the variable

order satisfies the following condition:

$$[d_1, \ldots, d_L] \prec [b_{11}, \ldots, b_{u1}] \prec \ldots \prec [b_{1m}, \ldots, b_{um}], \tag{3.4}$$

where the variables inside the square brackets can be in arbitrary order.

Theorem 1 The initial Boolean function F that is constructed from a training data set D through Algorithm 4 is the informational equivalent of the initial region vector G that is constructed from D through Algorithm 2.

Proof. It is possible to derive G from F: 1) Calculate all the minterms of F. 2) Convert each minterm into a set of a hash value and a unit hypercube based on the relationship between $(\textsc{CodeY}, \textsc{CodeZ})$ and (ϕ, ν). 3) Construct G with the obtained sets of hash values and unit hypercubes in a similar manner to Algorithm 2. Note that the minterm is a logical product of Boolean variables in which each Boolean variable appears once. Also, F can be derived from G in an opposite manner. □

The initial region vector G can be expressed very efficiently by using BDD representations, because a common pattern in G is expressed as a node in a BDD. For example, the minterms of F defined by Equation (3.3) are

$$\begin{aligned} b_{11} \wedge b_{21} \wedge b_{12} \wedge b_{22}, \\ b_{11} \wedge b_{21} \wedge b_{12} \wedge \overline{b_{22}}, \\ b_{11} \wedge b_{21} \wedge \overline{b_{12}} \wedge \overline{b_{22}}, \\ b_{11} \wedge \overline{b_{21}} \wedge b_{12} \wedge \overline{b_{22}}, \\ \text{and } \overline{b_{11}} \wedge b_{21} \wedge b_{12} \wedge \overline{b_{22}}. \end{aligned}$$

Assuming that $\{b_{11}, b_{12}\}$ and $\{b_{21}, b_{22}\}$ represent the unsigned-integer-type coding of z_1 and z_2 respectively, the minterms mentioned above correspond to

$$(z_1, z_2) = (3,3),\ (3,2),\ (2,2),\ (3,0),\ \text{and}\ (1,2),$$

respectively. Figure 3.4a shows the region that F represents. In Figure 3.4a, there is a common pattern in $[0,2] \times [2,4]$ and $[2,4] \times [0,2]$, which is expressed as Node E in Figure 3.3.

Given the initial Boolean function F, a one-class classifier occ_F can be constructed as follows:

$$occ_F(x,y) = \begin{cases} +1 & \text{if } \sigma(x) \in \mathcal{H} \text{ and} \\ & \textsc{CodeY}(y) \wedge \textsc{CodeZ}(\sigma(x)) \wedge F = 1, \\ -1 & \text{otherwise,} \end{cases} \tag{3.5}$$

which corresponds to a classifier defined by Equation (3.2).

3.2.3 The Relationship between Over-approximations

The over-approximation of the initial Boolean function is defined as follows:

Definition 10 A Boolean function $\tilde{F}$ is an over-approximation of the initial Boolean function F if the following condition is satisfied:

$$\begin{aligned} \forall y,\ \forall z, \quad & \textsc{CodeY}(y) \wedge \textsc{CodeZ}(z) \wedge F = 1 \\ & \Rightarrow\ \textsc{CodeY}(y) \wedge \textsc{CodeZ}(z) \wedge \tilde{F} = 1. \end{aligned}$$

Theorem 2 A region vector $\tilde{G}$ that over-approximates the initial region vector G is derived from a Boolean formula $\tilde{F}$ that over-approximates the initial Boolean formula F, which is the informational equivalent of G.

Proof. Assume that $\tilde{G}$ is derived from $\tilde{F}$ in the same manner as mentioned in the proof of Theorem 1. We prove that $\tilde{G}$ over-approximates G. Let (y, z) be data that satisty the following:

$$z \in G_{\phi(y)}. \tag{3.6}$$

Since F is the informational equivalent of G, F includes the following minterm:

$$\textsc{CodeY}(y) \wedge \textsc{CodeZ}(z). \tag{3.7}$$

Since $\tilde{F}$ over-approximates F, the minterm expressed by Equation (3.7) is also included in $\tilde{F}$. Since $\tilde{G}$ is derived from $\tilde{F}$, the following satisfies:

$$\nu(z) \subseteq \tilde{G}_{\phi(y)} \tag{3.8}$$

From the fact that $z \in \nu(z)$ and Equation (3.8), it follows

$$z \in \tilde{G}_{\phi(y)}. \tag{3.9}$$

From Equation (3.6) and Equation (3.9), it is proved that $\tilde{G}$ over-approximates G. □

Note that it is not always possible to derive $\tilde{F}$ from $\tilde{G}$ because $\tilde{F}$ can only express a region vector that is a union of unit hypercubes. From Theorem 2, we can over-approximate the initial region vector G by over-approximating the initial Boolean function F. In the following sections, we propose an efficient algorithm to over-approximate the initial Boolean function.

3.2.4 Calculation of the Density and the Level of a BDD Node

Let F be the initial Boolean function obtained by Algorithm 4 and T be a BDD representation of F. Each node in T corresponds to a Boolean function and the number of minterms of each node can be calculated recursively by the following equation [66]:

$$\tau_{\mathcal{N}} = \begin{cases} 2^{L+mu} & \text{if } \mathcal{N} \text{ is the terminal 1,} \\ 0 & \text{if } \mathcal{N} \text{ is the terminal 0,} \\ \dfrac{\tau_{\mathcal{T}}}{2} + \dfrac{\tau_{\mathcal{E}}}{2} & \text{otherwise,} \end{cases} \tag{3.10}$$

where $\tau_{\mathcal{X}}$ is the number of minterms of node $\mathcal{X}$, and $\mathcal{T}$ and $\mathcal{E}$ are the destination nodes of the true and false edges of the node $\mathcal{N}$, respectively. The density and the level of a BDD node is defined as follows:

Definition 11 The density of node $\mathcal{N}$, denoted by $\rho_{\mathcal{N}}$, is defined as

$$\rho_{\mathcal{N}} = \frac{\tau_{\mathcal{N}}}{2^{L+mu}}. \tag{3.11}$$

Definition 12 The level of node $\mathcal{N}$, denoted by $\lambda_{\mathcal{N}}$, is defined as

$$\lambda_{\mathcal{N}} = \begin{cases} m+1 & \text{if } \mathcal{N} \text{ is a terminal node,} \\ j & \text{if } w_{\mathcal{N}} \in \{b_{1j}, \ldots, b_{uj}\},\ j = 1, \ldots, m \\ 0 & \text{if } w_{\mathcal{N}} \in \{d_1, \ldots, d_L\}, \end{cases} \tag{3.12}$$

where $w_{\mathcal{N}}$ is a Boolean variable to which node $\mathcal{N}$ corresponds.

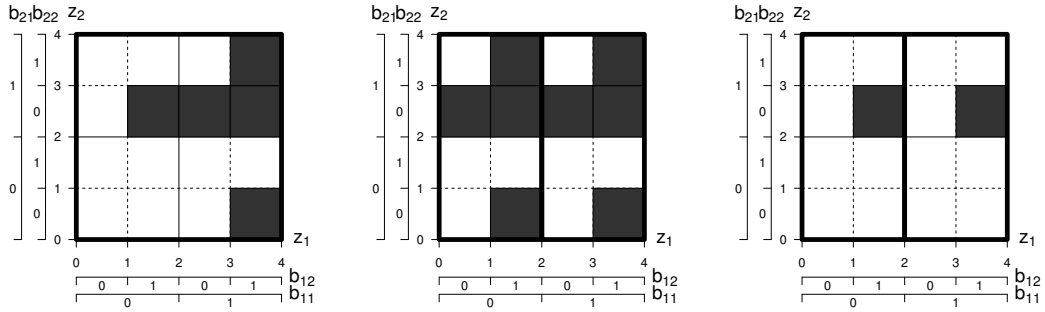
Table 3.1: The number of minterms $\tau_{\mathcal{N}}$, the density $\rho_{\mathcal{N}}$ and the level $\lambda_{\mathcal{N}}$ of each node $\mathcal{N}$ in Figure 3.3.

$\mathcal{N}$	$\tau_{\mathcal{N}}$	$\rho_{\mathcal{N}}$	$\lambda_{\mathcal{N}}$
Node A	5	0.3125	1
Node B	8	0.5	1
Node C	2	0.125	1
Node D	12	0.75	2
Node E	4	0.25	2
Node F	8	0.5	2
The terminal 1	16	1	3
The terminal 0	0	0	3

For example, Table 3.1 shows $\tau_{\mathcal{N}}$, $\rho_{\mathcal{N}}$ and $\lambda_{\mathcal{N}}$ of the nodes in Figure 3.3. Figure 3.4 shows the regions that the nodes in Figure 3.3 represents where bold squares mean common patterns caused by don't-care variables. For example, Node D in Figure 3.3 represents Boolean function $b_{12} \vee \left(\overline{b_{12}} \wedge \overline{b_{22}}\right)$, where b_{11} and b_{21} are don't-care variables. As a result, common patterns appear in four regions in Figure 3.4d that are defined by the combination of these don't care variables. Note that the pattern in the region $b_{11} \wedge b_{21}$ in Figure 3.4d also appears in the same region in Figure 3.4a, because it is possible to reach Node D from Node A through the path that corresponds to $b_{11} \wedge b_{21}$ in Figure 3.3.

3.2.5 The BDD based Over-approximation

We propose Algorithm 5 that over-approximates the initial Boolean function F through the direct manipulation of its BDD representation T based on the densities and the levels of BDD nodes, which are mentioned in the previous section. In Algorithm 5, DensAndLevs first calculates the density and the level of each node in T based on Equation (3.10), (3.11), and (3.12). After copying T to T^{θ}, all the edges in T^{θ} are searched in a depth-first manner. If the density of the destination node of a edge is greater than or equal to a threshold θ, the destination of the edge is modified to the terminal 1. Al-



(a) Node A (b) Node B (c) Node C

(d) Node D (e) Node E (f) Node F

Figure 3.4: The regions that the nodes of the BDD in Figure 3.3 represents.

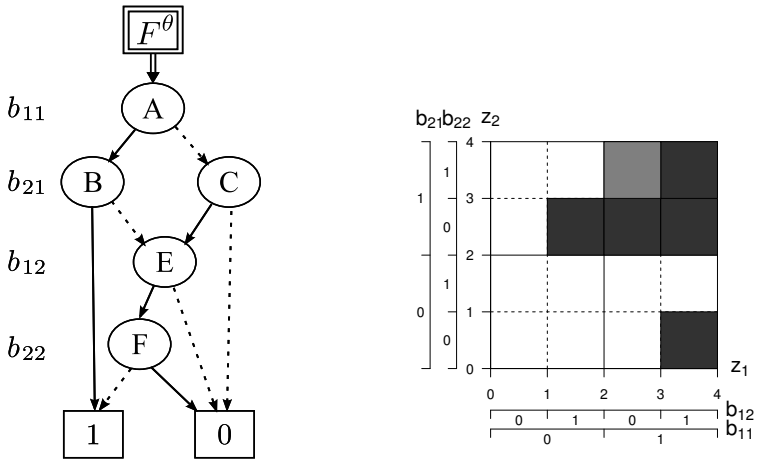
Algorithm 5 Over-approximation of the initial Boolean function F with a threshold θ

Input: A BDD T that represents the initial Boolean function F and a threshold θ.

Output: The BDD T^θ that represents the over-approximated Boolean Function F^θ.

```
1: DensAndLevs(T)
2: T^θ ← T
3: BddOverApprox(the root edge of T^θ, 0, θ)
4: return T^θ

5: procedure BddOverApprox(e, l, θ)
6:     N ← the destination node of e
7:     if N is a terminal node then
8:         return
9:     end if
10:    if ρ_N ≥ θ and λ_N > l then
11:        Modify the destination of e from N to the terminal 1.
12:        return
13:    else
14:        BddOverApprox(the true edge of N, λ_N, θ)
15:        BddOverApprox(the false edge of N, λ_N, θ)
16:        return
17:    end if
18: end procedure
```



(a) BDD (b) Region

Figure 3.5: (a) The result of the over-approximation by Algorithm 5, where F in Figure 3.3 is used as the initial Boolean function and the threshold θ is set to 0.75. (b) The region that F^θ represents.

gorithm 5 returns T^θ, which is a BDD that represents an over-approximated Boolean foumula F^θ. Algorithm 5 is a BDD-based implementation of Algorithm 3. The reconnection of edges works as the manipulation of line 10 in Algorithm 3. The levels of nodes are considered so that the reconnection happens only at the hypercube level. For example, Figure 3.5a shows the result obtained by Algorithm 5, where F in Figure 3.3 is used as the initial Boolean function and the threshold θ is set to 0.75. The edge from Node B to Node D in Figure 3.3 is modified by Algorithm 5 because the density of Node D is equal to θ and the level of Node B is smaller than that of Node D, as seen in Table 3.1. Figure 3.5b shows the region that F^θ represents. The light gray region means the region added by Algorithm 5. Because the density of the hypercube $[2, 4] \times [2, 4]$ in Figure 3.5b is equal to θ, the hypercube is added to the initial region.

A one-class classifier can be constructed by replacing F in Equation (3.5) with the over-approximated function obtained by Algorithm 5.

3.2.6 Computational Complexity

The computational complexity of constructing the initial Boolean function by Algorithm 4 is approximately $O(MN)$, where M is the maximum size of the created BDDs, because logical operations between BDDs are practically almost linear to the size of the BDDs [11]. On the other hand, the computational complexity of over-approximating the initial Boolean formula by Algorithm 5 is $O(E)$, where E is the number of edges of the BDD that represents the initial Boolean function. Algorithm 5 is efficient because the approximation is directly manipulated on a compressed form of a BDD. Consequently, the proposed method can deal with a large training data set practically, unless the created BDD become intractably huge.

3.3 Parameter Tuning with the MDL Principle

The proposed method described in the previous sections has parameter θ and it is necessary to tune θ to achieve high performance. Cross-validation-based techniques are often used to tune parameters of a model, but it is difficult to apply these techniques in the situation considered in this chapter as mentioned previously. One possible solution is to apply the likelihood-based cross-validation [73] if the model can be viewed as a probabilistic model. Cross-validation-based methods have another drawback that they are computationally expensive because a learning problem must be solved n times to evaluate each parameter setting if the n-fold cross-validation is employed. Here we propose a method for tuning the parameter of the proposed method based on the minimum description length (MDL) principle [59].

3.3.1 MDL for the Proposed Method

In the MDL principle, parameters are tuned so as to minimize the MDL that is the sum of the code length required to describe the model and the code length required to describe the data encoded by the model. The MDL is

defined as

$$\mathrm{MDL} = -\log L + \frac{\kappa}{2}\log \nu, \tag{3.13}$$

where L is the likelihood, κ is the number of parameters required to describe the model, and ν is the number of the training data. Unlike cross-validations, a learning problem must be solved once to evaluate the MDL of each parameter setting. Also, it is possible to evaluate the MDL even if the training data set is one-sided.

Let T be the BDD that represents the initial Boolean function F obtained by Algorithm 4, and let T^θ be the BDD that represents the over-approximated Boolean function F^θ obtained by Algorithm 5. Also, let G and G^θ be the initial region vector and its over-approximation corresponding to F and F^θ, respectively. Now, assume that each data in the training data set is allocated to a unit hypercube in hypercube $\mathcal{H}$ uniquely. Considering T^θ as a model that encodes the training data, the parameters in Equation (3.13) are calculated as follows:

$$L = \left(\frac{1}{V^\theta}\right)^\nu, \tag{3.14}$$

$$\kappa = 3\left|T^\theta\right|, \tag{3.15}$$

$$\nu = V, \tag{3.16}$$

where V and V^θ are the volumes of G and G^θ respectively, and $\left|T^\theta\right|$ is the number of variable nodes in T^θ. The volumes V and V^θ are calculated as the number of minterms of F and F^θ, respectively. The idea behind Equation (3.14) is that the probability density is assumed to equally distribute inside the region of G^θ, then the probability of each unit-hypercube in G^θ is $V^{-\theta}$. Consequently, the likelihood is given by Equation (3.14) because we have ν training data that are allocated to unit hypercubes. The number of the training data ν is given by Equation (3.16) under the assumption that each data is allocated to a unique unit hypercube. The number of parameters required to describe T^θ is given by Equation (3.15) because we need the information about the variable name, the destination node of the true edge, and that of the false edge of each node to store a BDD.

If we want to evaluate MDL with various θ, it is enough to execute Algorithm 4 and procedure DensAndLevs in Algorithm 5 just once, because these procedures do not depend on θ. Then, we execute procedure BddOverApprox in Algorithm 5 repeatedly with various θ and evaluate the MDL of each over-approximated result. Also, if the over-approximated Boolean function results in the constant 1 function with parameter $\hat{\theta}$, there is no need to calculate over-approximations with θ that is less than $\hat{\theta}$, because they also results in the constant 1 function from the monotonicity of the proposed algorithm, as mentioned in Section 3.1.

3.4 Related Work

3.4.1 One-Class Classifiers

Many methods have been proposed in relation to outlier detection, anomaly detection, novelty detection, concept learning, and so on [19, 69]. Most of these techniques can be applied to one-class classification problems, with or without modifications. The features of some of the existing methods are listed in Table 3.2, comparing with the proposed method. In Table 3.2, the column *model settings* contain the settings that must be determined by users in advance, but whose choice is less sensitive to the final performance and for which there exists a common choice, for example, the Gaussian kernel for kernel-based methods, and the Euclidian distance for distance-based methods. The column *magic parameters* contains the parameters that must be set by users in advance and have a strong influence on the final performance [69], such as variance parameters for the Gaussian kernel, and the number of hidden units for neural networks. The column *output score* contains the type of value returned by the model toward a particular input, for example, the Parzen density estimator returns the probability and the μLSIF returns the importance of the given data. Finally, the column *score threshold* indicates whether a threshold of the output score is required for building a one-class classifier, for example, the one-class SVM does not require any score threshold because it outputs either $+1$ or -1, while the LOF requires a score threshold for the output, because it returns a continuous score. From Table 3.2, we

can see that most existing methods, except for the Parzen density estimator, have no means for selecting their magic parameters when the training data set is one-sided. Note that the μLSIF requires both a training data set and a test data set instead of requiring a labeled training set to learn its magic-parameters. Although it is possible to select the magic parameters of the Parzen density estimator through likelihood cross-validation, a score threshold is required to construct a one-class classifier based on its output score, and no explicit criterion exists for choosing the score threshold. Consequently, no existing one-class classifier can be learnt fully automatically from a one-sided training data set, in other words, there has been no parameter-free [45] approach for the one-class classification problem. This is one of the motivating factors for our research. In the proposed method, the magic parameter is determined based on the MDL principle, and it does not require any score threshold to construct a one-class classifier, because it outputs either $+1$ or -1. Another motivation for out research comes from the fact that some methods in Table 3.2 become computationally intractable when the training data set is large, because the computational complexity is $O(N^2)$ or more, where N is the number of data in the training data set. In comparison, the proposed method can deal with a large training data set practically, unless the created BDD become intractably huge, as mentioned in Section 3.2.6.

Table 3.2: A list of methods related to the one-class classification problem and their features.

Name	Category	Model settings	Magic parameters (MP)	MP tuning methods	Output score	Score threshold
LOF [16]	kNN-like	Distance function	# of nearest neighbors	Not available	Local outlier factor	Required
One-class SVM [63]	SVM-based	Kernel function	Kernel parameters, Outlier upper bound	Not available	Either $+1$ or -1	Not required
Parzen density estimator [9]	Nonparametric estimation	Kernel function	Kernel parameters	Likelihood cross validation	Probability	Required
Replicator neural networks [38]	Neural network-based	Activation function	# of hidden units	Not available	Reconstruction error	Required
μLSIF [42]	Direct importance estimation	Kernel function, # of basis functions	Kernel parameters, Regularization parameters	Not available	Importance	Required
OCBDD (proposed)	BDD-based	Example normalizer σ, Coding length m	Density threshold θ	MDL principle	Either $+1$ or -1	Not required

3.4.2 BDD-based Learning Methods

Few applications use BDDs as their basis in the fields of data mining and machine learning. For example, Minato and Arimura [40] applied the zero-suppressed BDD (ZDD), which is a modification of a BDD, for frequent item set mining. Loekito and Bailey [48] applied ZDD for the analysis of contrast patterns of two item sets. As far as we know, BDDs have not been applied to classification problems in the past.

3.4.3 Over-Approximating Methods

Some algorithms that approximate a Boolean function have been proposed in the literature. Ravi et al. [57] proposed an algorithm that over-approximates a Boolean function that is expressed as a BDD by manipulating edges so as to minimize the ratio of minterms toward the number of nodes. Although we conducted some experiments that use the approximation algorithm proposed by Ravi et al. instead of the proposed algorithm, it ended in a poor result for the purpose of classification, as shown in Section 3.5.

3.5 Experimental Results

In this section, we present experimental results to demonstrate the proposed method. First, we show the result with a synthetic data set, and then we describe the results with real data sets, which include Shuttle data set and KDD Cup 99 data set from the UCI machine learning repository [2]. We implemented the proposed method as a C program with the help of CUDD [65]. Although the algorithms mentioned in the previous sections assume no use of complement edges, they are modified slightly so as to apply complement edges, because complement edges are used in CUDD. Hereafter, we refer to the proposed classifier as OCBDD. We use σ in Equation (3.1) as an example normalizer in the experiment. For comparison, real problems were also solved by the one-class support vector machine (OCSVM) [63], where e1071 [29] was used to solve the OCSVM. The Gaussian kernel is used in the OCSVM. The experiment was performed on a 32 bit Microsoft Windows XP machine with an Intel Core i7 CPU (2.80 GHz, 3.5 GB RAM).

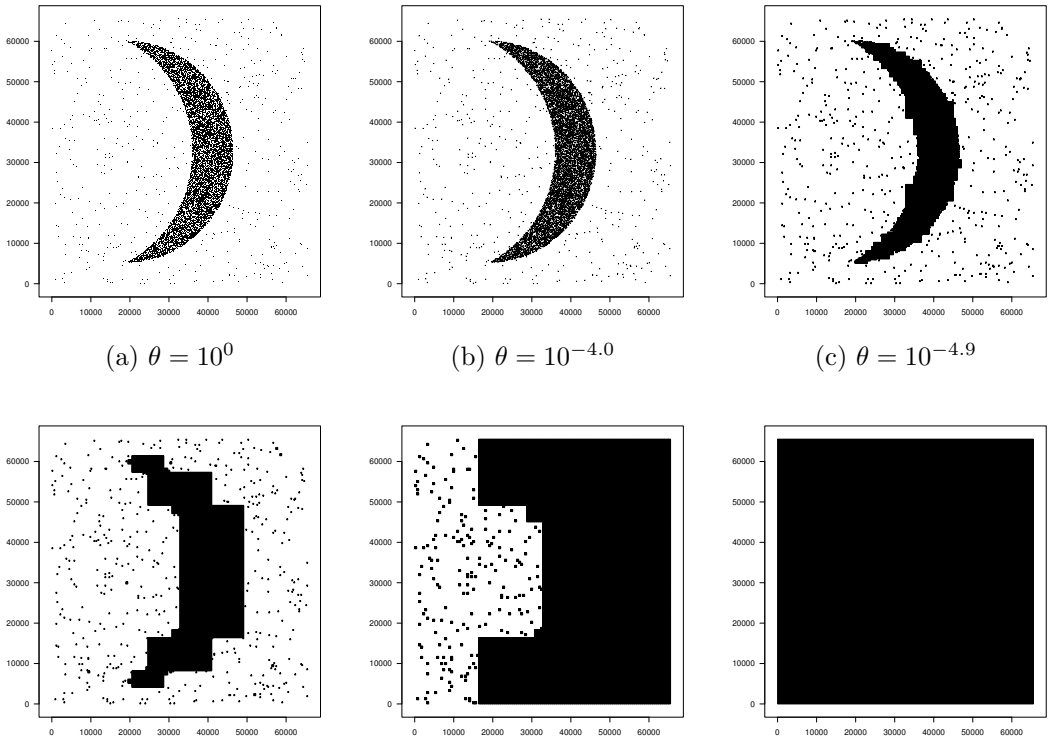
3.5.1 Synthetic Data Set

Data Overview

A synthetic data set that has no categorical attributes and two continuous attributes was generated and named MoonStar. The data that are inside the moon-shaped region are considered to belong to *the moon class*, and the data that are outside the moon-shaped region are considered to belong to *the star class*. The moon class is the target class and data from the star class are regarded as noise. The MoonStar data set consists of $N = 10000$ data: 95 % of which are randomly sampled from the moon class, and the remaining 5 % of which are randomly sampled from the star class.

Results

We first set $m = 16$, which means that sixteen Boolean variables are used to code each continuous attribute. The initial Boolean function F is constructed from the MoonStar data set through Algorithm 4. Then F is over-approximated by Algorithm 5 with $\theta = 10^0, 10^{-0.1}, \ldots, 10^{-10}$. Figure 3.6 shows the regions that the over-approximated Boolean functions represent, where the results with $\theta = 10^0, 10^{-4.0}, 10^{-4.9}, 10^{-5.2}, 10^{-5.5}, 10^{-5.7}$ are picked up so that the behavior of the proposed over-approximation can be seen distinctly. Note that the result of $\theta = 10^0$ corresponds to the region that the initial Boolean function represents, because no over-approximation is done when $\theta = 10^0$. In this experiment, the result with $\theta = 10^{-4.9}$ achieved the lowest MDL. Figure 3.6c shows that the moon-shaped region, which is the region of the target class, is estimated quite properly with the MDL optimal threshold $\theta = 10^{-4.9}$. Also, we can see that the region outside the moon is kept very small when $\theta = 10^{-4.9}$ even though the training data set contains 5 % of the star data, which means that the proposed method is robust toward the noise in a training data set. Table 3.3 shows the number of variable nodes of the BDD, the size of the region, and the MDL of each over-approximated result in Figure 3.6. From Table 3.3, the BDD nodes are reduced from 46137 to 1153, and the region is expanded from 10^4 to $10^{8.8}$ in size by Algorithm 5 with $\theta = 10^{-4.9}$. The computational time for constructing the initial Boolean function F by Algorithm 4 was about 0.08 seconds, and it took



(a) $\theta = 10^{0}$ (b) $\theta = 10^{-4.0}$ (c) $\theta = 10^{-4.9}$

(d) $\theta = 10^{-5.2}$ (e) $\theta = 10^{-5.5}$ (f) $\theta = 10^{-5.7}$

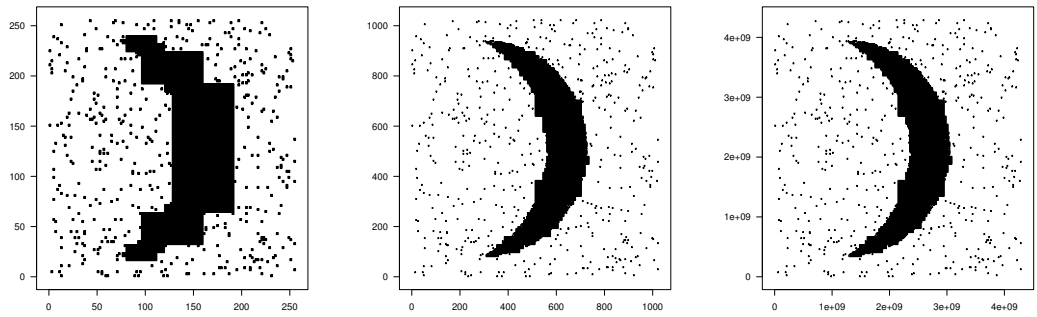
Figure 3.6: The over-approximated region G^{θ} for the MoonStar data set. The result with $\theta = 10^{-4.9}$ achieves the lowest MDL among the results with $\theta = 10^{0}, 10^{-0.1}, \ldots, 10^{-10}$.

about 0.03 seconds to over-approximate F by Algorithm 5 with $\theta = 10^{-4.9}$.

The MDL optimal over-approximations are also calculated for $m = 8, 10, 32$ respectively to assess the robustness of the proposed method with regard to m, which are shown in Figure 3.7. The threshold parameters selected by the MDL principle are $\theta = 10^{-0.8}, 10^{-1.3}, 10^{-14.5}$ for $m = 8, 10, 32$, respectively. From Figure 3.7, we can see that the choice of m has little impact on the result if m is set large enough, in this experiment $m \geq 10$.

Table 3.3: The number of variable nodes of the BDD, the size of the region that the BDD represents, and the MDL of each result in Figure 3.6

θ	# of nodes	region size	MDL
$10^{0.0}$	46137	$10^{4.0}$	729510
$10^{-4.0}$	8289	$10^{7.7}$	291801
$10^{-4.9}$	1153	$10^{8.8}$	217837
$10^{-5.2}$	899	$10^{9.0}$	218785
$10^{-5.5}$	318	$10^{9.4}$	221761
$10^{-5.7}$	0	$10^{9.6}$	221807



(a) $m = 8$ (b) $m = 10$ (c) $m = 32$

Figure 3.7: The MDL optimal results for the MoonStar data set with $m = 8, 10, 32$.

3.5.2 Shuttle Data Set

Data Overview

The Shuttle data set consists of eight continuous attributes and no categorical attributes, where the attribute named *time* is excluded in our experiment, and each of them is classified into seven classes. The number of data in the training data set and the test data set are 43500 and 14500, respectively, in which approximately 80 % of the both data sets belong to class 1. In the experiment, we consider class 1 as the target class and the data other than class 1 are extracted from the training data set. As a result, the size of the training data set is reduced to 34108. The estimated classifiers are evaluated with the test data set by checking if they can distinguish the class 1 data from the other class data.

Results of the Proposed Method

We set $m = 16$. The initial Boolean function is constructed from the training data set, and then, the initial Boolean function is over-approximated with $\theta = 10^{-0.1}, 10^{-0.2}, \ldots, 10^{-30}$, respectively, by the proposed method. The MDL is evaluated for each over-approximation. Also, the true positive rate (TPR) and the false positive rate (FPR) on the test data set are evaluated on each over-approximated result. Note that the TPR means the ratio of the correctly classified target class data toward the target class data in the test data set, and the FPR means the ratio of the incorrectly classified non-target class data toward the non-target class data in the test data set. Both the TPR and the FPR range from 0 to 1, and (TPR, FPR)=(1,0) means that the classification is done perfectly on the test data set. Results are shown in Figure 3.8. From Figure 3.8a, the MDL decreases as θ gets smaller until θ reaches around 10^{-15}, then, the MDL increases. The MDL is minimal at $\theta = 10^{-14.5}$. Although the MDL in Figure 3.8a is not unimodal strictly, there is a possiblity of finding a near-optimal θ effectively by using optimization methods, such as the golden section search. On the other hand, an accurate classification, which means that the TPR and the FPR are close to 1 and 0 respectively, is achieved if we choose θ around $[10^{-18}, 10^{-15}]$ from Figure 3.8b. Therefore, we can see that a proper threshold is selected based on the MDL principle for the Shuttle

Table 3.4: The true positive rate and the false positive rate for the Shuttle test data set. The threshold parameter of OCBDD is set to $\theta = 10^{-14.5}$, which is tuned based on the MDL principle.

		OCSVM ($\nu = 0.01$)		
	OCBDD	$\gamma = 0.1$	$\gamma = 1$	$\gamma = 10$
True positive rate	0.983	0.990	0.988	0.954
False positive rate	0.000	0.299	0.000	0.000

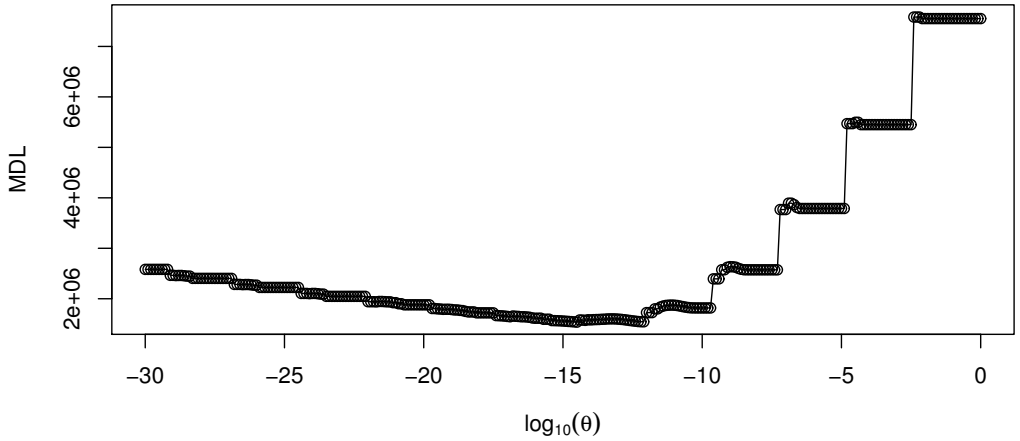
Table 3.5: Computational time for the Shuttle data set (in seconds).

			OCSVM ($\nu = 0.01$)		
		OCBDD	$\gamma = 0.1$	$\gamma = 1$	$\gamma = 10$
Training	Magic parameter tuning	37.6	–	–	–
	Learning a classifier	2.3	4.1	4.4	26.4
Classifying the test data set		0.3	0.4	0.4	1.4

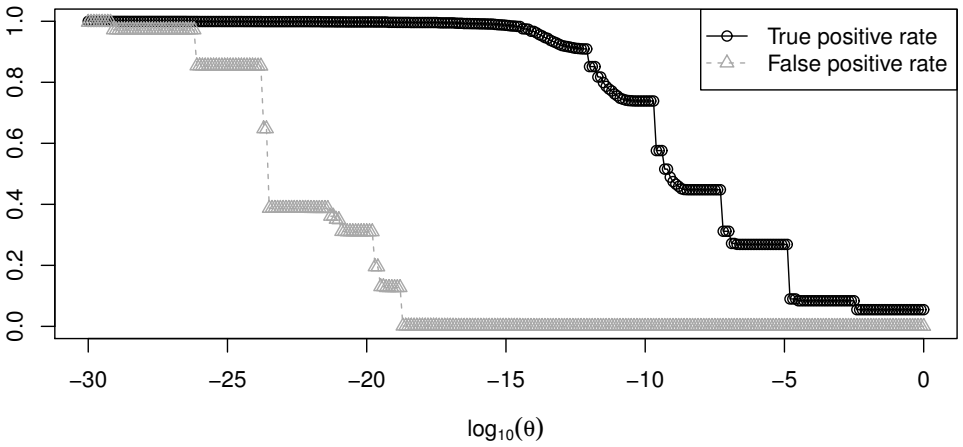
data set.

Comparison with Other Methods

The classification result of the proposed method with $\theta = 10^{-14.5}$, which is selected based on the MDL principle, is compared with that of the OCSVM. Although the OCSVM has two magic parameters, kernel parameter γ and outlier upper bound ν, it has no means to tune its magic parameters as mentioned in Section 3.4. Therefore, we evaluated the OCSVM with $\gamma = 0.1, 1, 10$, where ν is fixed to 0.01, respectively. The TPR and the FPR on the test data set are listed in Table 3.4. From Table 3.4, we can see that the classification ability of the OCSVM changes with respect to its magic parameter setting. On the other hand, the result of the proposed method is comparable to the best result of the OCSVM. The computational time are listed in Table 3.5. The 300 candidates of θ are evaluated based on the MDL principle in the proposed method as mentioned in the previous section, but it took only less than 40 seconds, showing the efficiency of the proposed method.



(a) The MDL of the over-approximated results



(b) The true positive rate and the false positive rate on the test data set

Figure 3.8: Experimental results of the proposed method with the Shuttle data set: The MDL of the over-approximated results (a), and the true positive rate and the false positive rate on the test data set (b).

Table 3.6: Classification results of the Shuttle test data set where the over-approximation algorithm proposed by Ravi et al. [57] is used instead of the proposed over-approximation algorithm in Algorithm 5

	$q = 1$	$q = 1.1$	$q = 1.2$	$q = 1.3$	$q = 1.4, 1.5, 1.6$	$q = 1.7$
True positive rate	1.000	1.000	0.999	0.984	0.820	0.055
False positive rate	1.000	0.999	0.995	0.988	0.852	0.000

We also evaluate the approximation algorithm proposed by Ravi et al. [57] on its apllicability to the one-class classification problem. Their algorithm is implemented as a function in CUDD[65], which is named `Cudd_RemapOverApprox`. In the experiment, `Cudd_RemapOverApprox` is used instead of BddOverApprox in Algorithm 5. `Cudd_RemapOverApprox` has a parameter q that controls the degree of the approximation. The TPR and the FPR on the Shuttle test data set is evaluated with various p, and the result is shown in Table 3.6. From Table 3.6, the approximation algorithm proposed by Ravi et al. [57] is not suitable for the purpose of the one-class classification.

3.5.3 KDD Cup 99 Data Set

Data Overview

The KDD Cup 99 data set is related to a network intrusion problem. It consists of seven categorical attributes and 34 continuous attributes, and each data is labeled either *normal* or one of the 21 types of attacks, *back*, *buffer_overflow*, and so on. The training data set and the test data set contain approximately five million data and 310 thousand data, respectively. In reference to Yamanishi [74], we employ the following problem settings: 1) The data that satisfies $logged\_in = 1$ is extracted from the original data set. The size of the extracted data are 703067 and 53645 in the training data set and the test data set, respectively. A one-class classifier that distinguishes the *normal* access from attacks is estimated from the training data set. Note that the training data set contains 3377 (0.5%) attacks, which are considered as noise data in the training data set. 2) Four of the original 41 attributes {*service*, *duration*, *src_bytes*, and *dst_bytes*} are used to construct a classi-

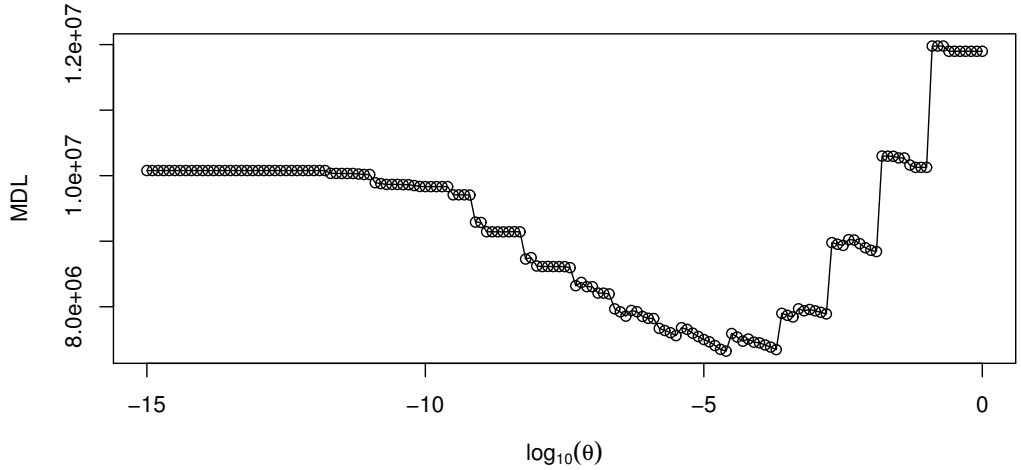
fier because they are thought to be the most basic attributes. *Service* is the only categorical attribute that has 41 original levels that are reassigned into five levels {*http*, *smtp*, *ftp*, *ftp_data*, and *others*} in the experiment. Because the continuous attributes are concentrated around 0, each of them are transformed by $\log(x + 0.1)$.

Results of the Proposed Method

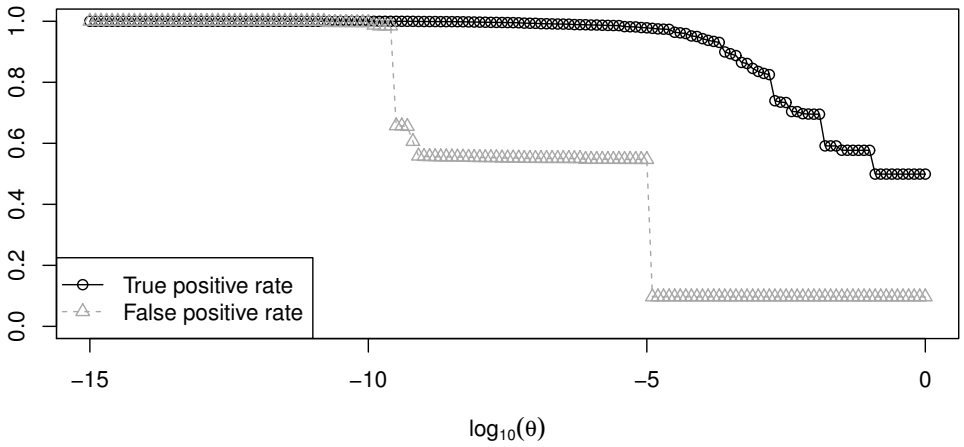
We set $m = 16$. The initial Boolean function is constructed from the training data set by Algorithm 4 and is over-approximated by Algorithm 5 with $\theta = 10^{-0.1}, 10^{-0.2}, \ldots, 10^{-15}$, respectively. Figure 3.9 shows the MDL, the TPR and the FPR of each over-approximated result. From Figure 3.9a, the MDL is roughly unimodal with respect to θ, which is similar to the result of the Shuttle data set, and the MDL is minimal at $\theta = 10^{-4.6}$. From Figure 3.9b, the classification is done accurately if we choose θ around $[10^{-5}, 10^{-4}]$. Therefore, the parameter θ is tuned properly based on the MDL principle for the KDD Cup 99 data set.

Comparison with the OCSVM

The classification result of the proposed method with the tuned parameter $\theta = 10^{-4.6}$ is compared with that of the OCSVM. The OCSVM are evaluated with $\gamma = 0.1, 1, 10$, where ν is fixed to 0.01, respectively. The TPR and the FPR on the test data set are listed in Table 3.7. From Table 3.7, the proposed method achieves the lowest FPR, and the TPR of the proposed method is high comparable to the results of the OCSVM. It goes without saying that there is a possibility that the OCSVM achieves better results if more combinations are tried on its magic parameters, but it is practically very hard to select good parameters for the OCSVM in advance. The computational time are listed in Table 3.8. It took only less than 35 seconds to evaluate the MDL of the 150 candidates of parameter θ in the proposed method as seen in Table 3.8. Although the size of the training data set of the KDD Cup 99 data set is twenty times larger than that of the Shuttle training data set, the computational time of tuning the magic parameter has little difference between the two data sets, because the computational time of the



(a) The MDL of the over-approximated results



(b) The true positive rate and the false positive rate on the test data set

Figure 3.9: Experimental results of the proposed method with the KDD Cup 99 data set: The MDL of the over-approximated results (a), and the true positive rate and the false positive rate on the test data set (b).

Table 3.7: The true positive rate and the false positive rate for the KDD Cup 99 test data set. The threshold parameter of OCBDD is set to $\theta = 10^{-4.6}$, which is tuned based on the MDL principle.

		OCSVM ($\nu = 0.01$)		
	OCBDD	$\gamma = 0.1$	$\gamma = 1$	$\gamma = 10$
True positive rate	0.974	0.985	0.985	0.947
False positive rate	0.097	0.463	0.453	0.547

Table 3.8: Computational time for the KDD Cup 99 data set (in seconds).

			OCSVM ($\nu = 0.01$)		
		OCBDD	$\gamma = 0.1$	$\gamma = 1$	$\gamma = 10$
Training	Magic parameter tuning	34.8	–	–	–
	Learning a classifier	18.9	7133.1	2070.6	15591.9
Classifying the test data set		0.5	22.7	22.1	29.7

proposed over-approximation algorithm does not depend on the size of the training data set, but the size of the BDD that represents the initial Boolean function. The computational time for constructing the initial Boolean function for the KDD Cup 99 training data set was 18.4 seconds, which is about nine times larger than that for the Shuttle training data set. On the other hand, the computational time of learning the OCSVM increases rapidly as the training data set becomes large, as seen in Table 3.8.

3.5.4 Memory Usage of the Proposed Method

Table 3.9 shows the memory usage peak of the proposed method for each data set mentioned above. The number of categorical attributes, that of continuous attributes and that of training data are also shown in Table 3.9. The memory usage peaked while constructing the initial Boolean function by Algorithm 4 for every data set. From Table 3.9, the memory usage peak seems to be larger as the number of continuous attributes gets larger. The reason is that the number of Boolean variables increases rapidly according to the number of continuous attributes, and then the size of the created BDD

Table 3.9: Memory usage peaks of the proposed method.

Data set	# of categorical attributes	# of continuous attributes	# of training data	Mem. usage peak (MB)
MoonStar	0	2	10000	30.1
Shuttle	0	8	34108	93.2
KDD Cup 99	1 (5 levels)	3	703067	97.5

tends to be large. Also, the memory usage peak for the KDD Cup 99 data set is slightly more than that for the Shuttle data set even though the size of the training data set of the former is much larger than that of the latter. This is because a BDD is a kind of compressed form of data and can treat the overlapped data very efficiently.

3.6 Summary

In this chapter, we proposed a novel approach for the one-class classification problem. The region of the training data set is expressed as a Boolean function and is over-approximated efficiently through the direct manipulation of the binary decision diagram that represents the Boolean function. One of the key features of the proposed method is that a one-class classifier can be learnt in a parameter-free manner. In other words, the magic parameter of the proposed method can be tuned with a one-sided training data set that only contains the target class data. This feature is essential in some applications, such as anomaly detection. Another feature is that the computational complexity is almost linear in terms of the number of training data, and large training data sets can be dealt with efficiently unless the created binary decision diagram become intractably huge.

Chapter 4

Outlier Detection using Binary Decision Diagrams

In this chapter, we propose a novel and efficient method for outlier detection. The goal of outlier detection is to find an unusual datum (*outlier*) from a given data set. Although many kinds of notion have been proposed to define an outlier, we consider a datum as an outlier if the *leave-one-out density* is lower than a given threshold for a set of regions around the datum. The leave-one-out density is a ratio of the number of data inside a region to the volume of the region, in which the focused datum is removed from the original data set. Generally, a leave-one-out like method is time consuming because a learning procedure is repeated N-times, where N is the cardinality of a data set. However, the proposed method enables us to evaluate the leave-one-out density efficiently without repeating a learning procedure N-times.

We utilize the initial region method described in Chapter 3. Although the one-class classification problem and the outlier detection problem are very similar, the one-class classifier proposed in Chapter 3 is not directly applicable to outlier detection, because the classifier is estimated as an over-approximation of a given data set and never classify a datum in the data set as an outlier. We extend the method proposed in Chapter 3 to outlier detection by introducing the notion of leave-one-out density and developing an efficient algorithm to evaluate it.

The proposed method is compared to other well-known outlier detection

methods, the one-class support vector machine [62] and the local outlier factor [15], with both synthetic data sets and realistic data sets. The experimental results indicate that the computation time of the proposed method is shorter than those of the other methods, keeping the outlier detection accuracy comparable to the other methods.

The outlier detection problem addressed in this chapter is formally defined in Section 4.1. We give an explanatory example and define some notations in Section 4.2. In Section 4.3, the outline of the proposed method is first stated, and then, its efficient implementation based on binary decision diagrams is proposed. Related works are reviewed in Section 4.4. Section 4.5 gives experimental results. In Section 4.6, we summarize this chapter.

4.1 Problem Setting

Let D be a data set that includes N data. The i-th datum in D is denoted by $x^{(i)} \in \mathbb{R}^u$ $(i = 1, \ldots, N)$. We assume that there is no missing value in D and all the data in D are unlabeled. In the proposed method, we regard a datum as an outlier if the *leave-one-out density* is lower than a threshold for a set of regions around the datum. The leave-one-out density ρ_{LOO} of the i-th datum is defined as:

$$\rho_{\mathrm{LOO}}\left(i, \mathcal{D}\right) = \frac{\#\left(\left\{x^{(j)} \;\middle|\; x^{(j)} \in \mathcal{D},\ j = 1, \ldots, N,\ j \neq i\right\}\right)}{vol\left(\mathcal{D}\right)}$$

where $\mathcal{D}$ denotes a u-dimensional region such that $x^{(i)} \in \mathcal{D}$, $\#(\cdot)$ represents the cardinality of a set and $vol\left(\cdot\right)$ indicates the volume of the region. The *outlier score* S of the i-th datum is defined as:

$$S\left(i\right) = \max_{\mathcal{D} \in \tilde{\mathcal{D}}(i)} \rho_{\mathrm{LOO}}\left(i, \mathcal{D}\right)$$

where $\tilde{\mathcal{D}}\left(i\right)$ represents a set of regions around $x^{(i)}$ defined as:

$$\tilde{\mathcal{D}}\left(i\right) = \left\{u\text{-dimensional region } \mathcal{D} \;\middle|\; x^{(i)} \in \mathcal{D}\right\}.$$

A datum is detected as an *outlier* if the outlier score of the datum is less than a given threshold. Note that $\tilde{\mathcal{D}}\left(i\right)$ is not the set of all possible regions,

but rather a fixed family, which is defined in Section 4.3.1. In the following sections, we propose an efficient algorithm that enables us to evaluate the outlier score in near linear time with respect to N.

4.2 Preliminaries

In this section, we give an example data set which is used to demonstrate the proposed method and define some notations used in this chapter. We consider a data set in $\mathbb{R}^2$. The parameter used in the example normalizer σ, which is given in Equation 3.1, is set to $m = 3$. Then, the data set is projected into $\mathcal{H} = [0, 2^3)^2$ by σ. The projected data $z^{(j)}$, where j indicates the index of each datum in the data set, are shown as x-marks in Figure 4.1a. The initial region G is a u-dimensional region inside $\mathcal{H}$ that subsumes all the projected data, which is given as

$$G = \bigcup_{j=1,\ldots,N} \nu\left(z^{(j)}\right)$$

by using Algorithm 2, where ν is the neighborhood function given in Definition 6. The initial region G is illustrated as the gray region in Figure 4.1a.

The initial region G is expressed as a Boolean function by using the coding function CodeZ, which is given in Definition 8. The set of Boolean variables $\mathbb{B} := \{b_{ij} \mid i = 1, \ldots, u;\ j = 1, \ldots, m\}$ is used to code z, where b_{i1} and b_{im} represent the most and the least significant bit of the i-th element of z, respectively. The initial Boolean function F is given as a disjunction of logical formulas such as

$$F = \bigvee_{i=1,\ldots,N} \textsc{CodeZ}(z^{(i)})$$

by using Algorithm 4. Figure 4.1b shows a BDD that represents the initial Boolean formula F that is obtained from the data set in Figure 4.1a. Note that complement edges are used in Figure 4.1b, which is illustrated as dotted lines. A path from a function node to the terminal 1 corresponds to a conjunction of literals. If the path contains an even number of complement edges, the conjunction is included in the function.

We introduce the decoding function R that is defined as follows.

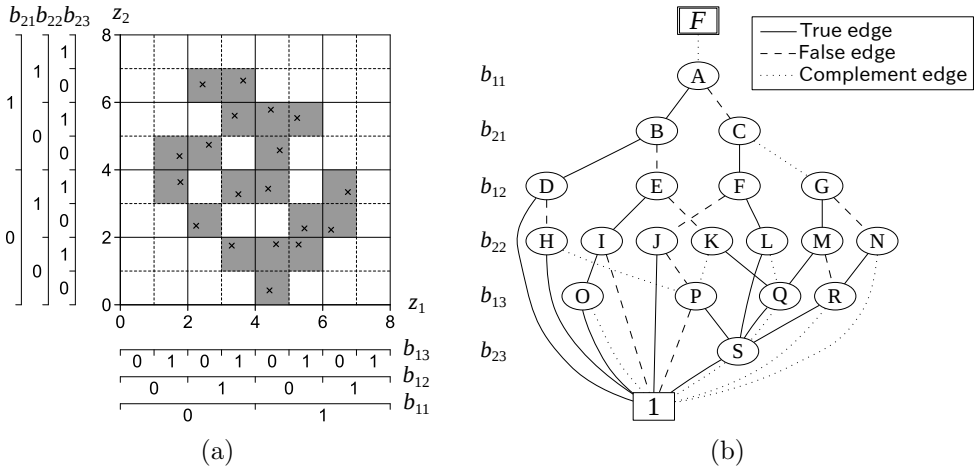


Figure 4.1: X-marks mean a normalized data set and the gray region is the initial region G (*left*). A BDD that represents the initial Boolean function F (*right*).

Definition 13 The decoding function R in $\mathbb{B} \to \mathbb{R}^u$ is a function that decodes a Boolean function defined on $\mathbb{B}$ into the corresponding u-dimensional region. In particular, $R(1) = \mathcal{H}$ and $R(F) = G$.

4.3 Proposed Method

4.3.1 Outline

In the proposed method, we calculate the leave-one-out density based on the normalized data $z^{(i)} = \sigma\left(x^{(i)}\right)$ as follows:

$$\rho_{\mathrm{LOO}}\left(i, \mathcal{C}\right) = \frac{\#\left(\left\{z^{(j)} \;\middle|\; z^{(j)} \in \mathcal{C},\ j = 1, \ldots, N,\ j \neq i\right\}\right)}{vol\left(\mathcal{C}\right)} \tag{4.1}$$

where $\mathcal{C}$ is a region such that $z^{(i)} \in \mathcal{C}$. The outlier score is given as:

$$S\left(i\right) = \max_{\mathcal{C} \in \tilde{\mathcal{C}}(i)} \rho_{\mathrm{LOO}}\left(i, \mathcal{C}\right) \tag{4.2}$$

where $\tilde{\mathcal{C}}(i)$ is a set of hypercubes defined as:

$$\begin{aligned} &\tilde{\mathcal{C}}\left(i\right) = \left\{R\left(\mathcal{L}_l(i)\right) \mid l = 0, \ldots, m\right\}, \\ &\mathcal{L}_0(i) = 1, \quad \mathcal{L}_l(i) = \bigwedge_{i'=1,\ldots,u} \bigwedge_{j'=1,\ldots,l} \tilde{b}_{i'j'} \quad (l = 1, \ldots, m) \end{aligned} \tag{4.3}$$

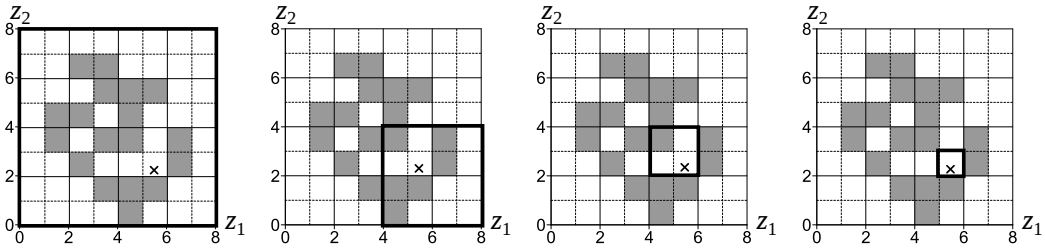


Figure 4.2: Bold squares show $\tilde{\mathcal{C}}(i)$ for the data shown as the x-mark. The leave-one-out density ρ_{LOO} of the datum are $\frac{18}{64}$, $\frac{6}{16}$, $\frac{1}{4}$ and $\frac{0}{1}$ for each hypercube. The outlier score of the datum is evaluated as $\max\left(\frac{18}{64}, \frac{6}{16}, \frac{1}{4}, \frac{0}{1}\right) \approx 0.38$.

where $\tilde{b}_{i'j'}$ are assignments of Boolean variables that is obtained by coding $z^{(i)}$ with CodeZ. For example, the datum in the region $[5,6) \times [2,3)$ in Figure 4.1a is coded as $(\tilde{b}_{11}, \tilde{b}_{21}, \tilde{b}_{12}, \tilde{b}_{22}, \tilde{b}_{13}, \tilde{b}_{23}) = (1,0,0,1,1,0)$. From (4.3), the set $\tilde{\mathcal{C}}(i)$ for this datum is derived as

$$\tilde{\mathcal{C}}(i) = \{[0,8) \times [0,8),\ [4,8) \times [0,4),\ [4,6) \times [2,4),\ [5,6) \times [2,3)\}$$

which are shown as bold squares in Figure 4.2.

We assume that there is no duplicate in D, that is, at least one of the attributes has a different value for $x^{(i)}$ and $x^{(j)}$ if $i \neq j$. Then, we can construct the initial region G so that each datum in D is allocated in a distinct unit hypercube by setting m large enough and using an appropriate example normalizer. In this case, the number of data inside a region can be calculated as the volume of the region unless the boundary of the region goes across a unit hypercube in which a datum exists. Therefore, the leave-one-out density defined as (4.1) can be calculated based on the initial region as follows for $\mathcal{C} \in \tilde{\mathcal{C}}(i)$:

$$\rho_{\mathrm{LOO}}(i, \mathcal{C}) = \frac{vol\left(G_{\mathrm{LOO}}(i) \cap \mathcal{C}\right)}{vol\left(\mathcal{C}\right)} \tag{4.4}$$

where G_{LOO} is the leave-one-out region defined as:

$$G_{\mathrm{LOO}}(i) = G \setminus \nu\left(z^{(i)}\right). \tag{4.5}$$

For example, Figure 4.2 illustrates the calculation of the leave-one-out density for the datum shown as the x-mark.

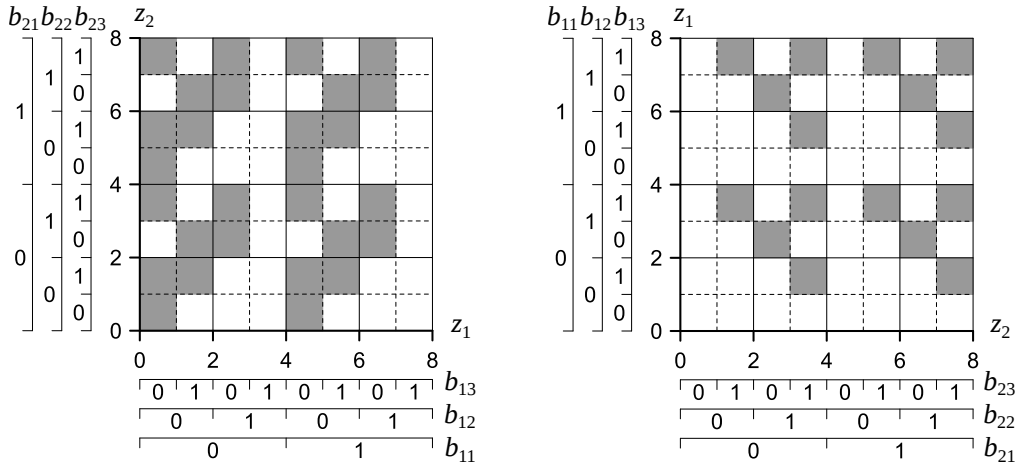


Figure 4.3: Examples of regions that BDD nodes represent: $R\left(\mathcal{N}_{\mathrm{E}}^{-}\right)$ (*left*) and $R\left(\mathcal{N}_{\mathrm{G}}^{+}\right)$ (*right*) where E and G are nodes in Figure 4.1b.

4.3.2 BDD based Implementation

The Number of Minterms Calculation

Let $\#_A$ be a function that returns the number of minterms of a Boolean function on the assumption that the domain of the function is A. For example, $\#_a(1) = 2$ and $\#_{\{a,b\}}(1) = 4$ where a and b are Boolean variables.

Lemma 1 For a Boolean formula F that is defined on $\mathbb{B}$, it holds that:

$$vol\left(R\left(F\right)\right) = \#_{\mathbb{B}}\left(F\right).$$

Proof. Since a minterm represents a unit hypercube in the initial region method, the number of minterms equals to the volume of the region that F represents. □

Assume that a BDD that represents the initial Boolean function F is given. For every node α in the BDD, we denote the Boolean function represented by α as $\mathcal{N}_{\alpha}^{+}$. Further, we denote the complement of $\mathcal{N}_{\alpha}^{+}$ as $\mathcal{N}_{\alpha}^{-}$. Both $\mathcal{N}_{\alpha}^{+}$ and $\mathcal{N}_{\alpha}^{-}$ represent regions inside $\mathcal{H}$. For example, Figure 4.3 shows $R\left(\mathcal{N}_{\mathrm{E}}^{-}\right)$ and $R\left(\mathcal{N}_{\mathrm{G}}^{+}\right)$ where E and G are nodes in Figure 4.1b. It is possible to efficiently calculate the number of minterms of $\mathcal{N}_{\alpha}^{+}$ and $\mathcal{N}_{\alpha}^{-}$ for each node α in a BDD in a depth-first manner [66]. For example, Table 4.1 shows the number of minterms of $\mathcal{N}_{\alpha}^{+}$ and $\mathcal{N}_{\alpha}^{-}$ for each node in Figure 4.1b. We can

Table 4.1: The number of minterms of $\mathcal{N}_\alpha^+$ and $\mathcal{N}_\alpha^-$ for each node α in Figure 4.1b.

α	A	B	C	D	E	F	G	H	I	J	K	L	M	N
$\#_{\mathbb{B}}\left(\mathcal{N}_\alpha^+\right)$	45	44	46	52	36	44	16	40	48	56	24	32	24	8
$\#_{\mathbb{B}}\left(\mathcal{N}_\alpha^-\right)$	19	20	18	12	28	20	48	24	16	8	40	32	40	56

O	P	Q	R	S	1
32	48	32	16	32	64
32	16	32	48	32	0

see that $\#_{\mathbb{B}}\left(\mathcal{N}_{\mathrm{E}}^-\right)$ and $\#_{\mathbb{B}}\left(\mathcal{N}_{\mathrm{G}}^+\right)$ are equal to the volumes of regions which are shown in Figure 4.3.

The Leave-one-out Density Calculation

Because of the fact that $\nu\left(z^{(i)}\right) \subseteq G$ and $\nu\left(z^{(i)}\right) \subseteq \mathcal{C} \in \tilde{\mathcal{C}}(i)$, it is derived from (4.4) and (4.5) that the following equation holds for $\mathcal{C} \in \tilde{\mathcal{C}}(i)$:

$$\rho_{\mathrm{LOO}}\left(i, \mathcal{C}\right) = \frac{vol\left(G \cap \mathcal{C}\right) - 1}{vol\left(\mathcal{C}\right)}. \tag{4.6}$$

By replacing $\mathcal{C}$ in (4.6) with $R\left(\mathcal{L}_l\left(i\right)\right)$, the following equation is derived.

$$\rho_{\mathrm{LOO}}\left(i, R\left(\mathcal{L}_l\left(i\right)\right)\right) = \frac{vol\left(G \cap R\left(\mathcal{L}_l\left(i\right)\right)\right) - 1}{vol\left(R\left(\mathcal{L}_l\left(i\right)\right)\right)} = \frac{vol\left(R\left(F \wedge \mathcal{L}_l\left(i\right)\right)\right) - 1}{vol\left(R\left(\mathcal{L}_l\left(i\right)\right)\right)} \tag{4.7}$$

From Lemma 1, (4.7) is transformed as follows:

$$\rho_{\mathrm{LOO}}\left(i, R\left(\mathcal{L}_l\left(i\right)\right)\right) = \frac{\#_{\mathbb{B}}\left(F \wedge \mathcal{L}_l\left(i\right)\right) - 1}{\#_{\mathbb{B}}\left(\mathcal{L}_l\left(i\right)\right)} = \frac{\#_{\mathbb{B}}\left(F \wedge \mathcal{L}_l\left(i\right)\right) - 1}{2^{(m-l)u}} \tag{4.8}$$

Lemma 2 In a BDD that represents F, let $\alpha_{i,l}$ be the node that can be reached from the function node through the path defined by $\mathcal{L}_l\left(i\right)$. Let c be the number of complement edges on the path. Then, it holds that:

$$\#_{\mathbb{B}}\left(F \wedge \mathcal{L}_l\left(i\right)\right) = \begin{cases} \#_{\mathbb{B}}\left(\mathcal{N}_{\alpha_{i,l}}^+\right)/2^{lu} & \text{if } c \text{ is even,} \\ \#_{\mathbb{B}}\left(\mathcal{N}_{\alpha_{i,l}}^-\right)/2^{lu} & \text{if } c \text{ is odd.} \end{cases}$$

Proof. $F \wedge \mathcal{L}_l(i)$ means that $l \times u$ Boolean variables that appear in $\mathcal{L}_l(i)$ are fixed to specific values in F. On the other hand, $\mathcal{N}^+_{\alpha_{i,l}}$ ($\mathcal{N}^-_{\alpha_{i,l}}$) means F with $l \times u$ Boolean variables in $\mathcal{L}_l(i)$ smoothed[1] if c is even (odd). Therefore, the number of minterms of $\mathcal{N}^+_{\alpha_{i,l}}$ ($\mathcal{N}^-_{\alpha_{i,l}}$) is 2^{lu} times larger than that of $F \wedge \mathcal{L}_l(i)$. □

Theorem 3 The leave-one-out density ρ_{LOO} is evaluated based on a BDD that represents the initial Boolean function F as follows:

$$\rho_{\mathrm{LOO}}\left(i, R\left(\mathcal{L}_l(i)\right)\right) = \begin{cases} \left(\#_{\mathbb{B}}\left(\mathcal{N}^+_{\alpha_{i,l}}\right) - 2^{lu}\right)/2^{mu} & \text{if } c \text{ is even,} \\ \left(\#_{\mathbb{B}}\left(\mathcal{N}^-_{\alpha_{i,l}}\right) - 2^{lu}\right)/2^{mu} & \text{if } c \text{ is odd.} \end{cases}$$

Proof. It follows from (4.8) and Lemma 2 immediately. It is worth mentioning that we can evaluate the leave-one-out density from the initial Boolean function F straightforwardly without any leave-one-out operation by using Theorem 3. □

For example, we consider a data set in Figure 4.1a and the datum in $[5,6) \times [2,3)$. The path of the datum is $F \bullet \mathrm{A} \circ \mathrm{B} \circ \mathrm{E} \circ \mathrm{K} \circ \mathrm{Q} \circ \mathrm{S} \bullet 1$ in Figure 4.1b, where $\circ$ and $\bullet$ mean non-complement and complement edges, respectively. The leave-one-out density of the datum is calculated from Theorem 3 and Table 4.1 as follows:

$$\left(\#_{\mathbb{B}}\left(\mathcal{N}^-_{\mathrm{A}}\right) - 2^0\right)/2^6 = 18/64, \quad \left(\#_{\mathbb{B}}\left(\mathcal{N}^-_{\mathrm{E}}\right) - 2^2\right)/2^6 = 6/16,$$
$$\left(\#_{\mathbb{B}}\left(\mathcal{N}^-_{\mathrm{Q}}\right) - 2^4\right)/2^6 = 1/4, \quad \left(\#_{\mathbb{B}}\left(\mathcal{N}^+_1\right) - 2^6\right)/2^6 = 0.$$

We can see that these values are equal to those in Figure 4.2.

4.3.3 The Proposed Algorithm and Computational Complexity

We propose Algorithm 6 that calculates the outlier score of each datum in D. In Algorithm 6, the time complexity of constructing the initial Boolean function F is approximately $O(MN)$, where M is the number of nodes of the

[1] Smoothing a Boolean function f with respect to x means $(f \wedge x) \vee (f \wedge \overline{x})$.

Algorithm 6 The outlier score calculation.

Input: A data set D.

Output: The outlier score S of each datum in D.

1: Construct the initial Boolean function F as a BDD.
2: Calculate the number of minterms of each node of the BDD.
3: **for** $i = 1$ to N **do**
4: Search the path that CODEZ($z^{(i)}$) represents in the BDD and evaluate $\rho_{\mathrm{LOO}}\left(i, R\left(\mathcal{L}_l\left(i\right)\right)\right)$ for $l = 0, \ldots, m$ by using Theorem 3.
5: $S\left(i\right) \leftarrow \max_{l=0,\ldots,m} \rho_{\mathrm{LOO}}\left(i, R\left(\mathcal{L}_l\left(i\right)\right)\right)$
6: **end for**

created BDD, because logical operations between BDDs are practically almost linear to the size of the BDDs [11]. The size of the created BDD depends on the characteristics of the data set and can be exponentially large in the worst case, but, it is compact for realistic data sets used in our experiments. The time complexity of calculating the number of minterms is $O(M)$ as mentioned in Section 4.3.2. The time complexity of calculating the outlier score is $O(muN)$ because the depth of the BDD is mu. Consequently, the time complexity of Algorithm 6 is $O((M + mu)N)$. Therefore, the proposed method can deal with a large data set efficiently unless the number of Boolean variables and the created BDD are intractably huge.

4.3.4 Dealing with Categorical Attributes

The proposed method can be extended in order to deal with a data set that consists of both continuous attributes and categorical attributes. Let $y^{(i)}$ be a vector of categorical attributes of the i-th datum. We extend the leave-one-out density defined as (4.1) as follows:

$$\rho_{\mathrm{LOO}}\left(i, \mathcal{C}\right) = \frac{\#\left(\left\{z^{(j)} \;\middle|\; z^{(j)} \in \mathcal{C},\ y^{(j)} = y^{(i)},\ j = 1, \ldots, N,\ j \neq i\right\}\right)}{vol\left(\mathcal{C}\right)} \tag{4.9}$$

Then, the outlier score defined as (4.2) can be evaluated very efficiently in the same manner as mentioned in the previous sections. The way of dealing with categorical attributes is described below.

Using CodeY defined in Definition 9, we can construct the initial Boolean formula F as

$$F = \bigvee_{i=1,\ldots,N} \left(\textsc{CodeY}(y^{(i)}) \wedge \textsc{CodeZ}(z^{(i)}) \right), \tag{4.10}$$

with Algorithm 4. On the construction of F as a BDD, the variable order is set such that the constraint in (3.4) holds. This leads to a corollary of Theorem 3 as follows:

Corollary 1 For a dataset that consists of both continuous attributes and categorical attributes, the leave-one-out density given as (4.9) is evaluated on the basis of the number of minterms of the nodes of a BDD that represents the initial Boolean function F, given as (4.10), as follows:

$$\rho_{\mathrm{LOO}}\left(i, R\left(\mathcal{L}_l\left(i\right)\right)\right) = \begin{cases} \left(\#_{\{\mathbb{B},\mathbb{D}\}} \left(\mathcal{N}^{+}_{\beta_{i,l}} \right) - 2^{lu+L} \right) / 2^{mu+L} & \text{if } d \text{ is even,} \\ \left(\#_{\{\mathbb{B},\mathbb{D}\}} \left(\mathcal{N}^{-}_{\beta_{i,l}} \right) - 2^{lu+L} \right) / 2^{mu+L} & \text{if } d \text{ is odd,} \end{cases}$$

where $\beta_{i,l}$ denotes the node that can be reached from the function node F through the path defined by $\textsc{CodeY}(y^{(i)}) \wedge \mathcal{L}_l(i)$ and d represents the number of complement edges on the path.

The outlier score can be evaluated by using Algorithm 6 in which Theorem 3 is replaced with Corollary 1.

4.4 Related Work

For outlier detection, the Mahalanobis distance [49] between the center of a data set and each datum is often used as a score of being an outlier. The Mahalanobis distance is a classical measure of distance between data points, in which the correlation of a data set is taken into account. The Mahalanobis distance works well if a data set fits well to a normal distribution, but, this condition seldom holds in practice.

Some of methods for one-class classification, such as those listed in Table 3.2, can also be applied to outlier detection. Schölkopf et al. [62] extended the support vector machine (SVM), which was originally invented for binary

classification, to the one-class setting. Their method estimates a hyperplane that separates the origin and a data set with maximum margin, in which the hyperplane can be nonlinear by introducing kernel functions. The data that are classified to the origin side are detected as outliers. The SVM has an advantage that various nonlinear hyperplanes are estimated by changing kernel parameters. Some heuristics are proposed to tune kernel parameters, such as [18]. Breunig et al. [15] proposed the local outlier factor (LOF) that is calculated based on the distance to the k-nearest neighbor of each datum and has an advantage that it can detect local outliers, that is, data that are outlying relative to their local neighborhoods. The LOF has been shown to perform very well in realistic problems [47]. An efficient calculation of the k-nearest neighbors is essential in the LOF. Some techniques are proposed to accelerate the k-nearest neighbors calculation, such as [6].

4.5 Experimental Results

We compare the proposed method with existing methods, the one-class support vector machine (OCSVM) and the local outlier factor (LOF). The proposed method is referred to as ODBDD. We implemented ODBDD as a C program with the help of CUDD [65]. The `ksvm` function in the `kernlab` package [43] is used for OCSVM and the `lofactor` function in the `DMwR` package [72] is used for LOF. The parameter m that defines the size of the hypercube $\mathcal{H}$ is fixed to $m = 16$ in ODBDD. In OCSVM, the Gaussian kernel is used and the kernel parameter γ is set to one of the 10%, 50% and 90% quantiles of the distance between samples [18], which are referred to as $\gamma_{0.1}$, $\gamma_{0.5}$ and $\gamma_{0.9}$, respectively. The parameter ν is fixed to $\nu = 0.1$ in OCSVM. In LOF, the number of neighbors is set to either $k = 10$ or $k = 50$. In OCSVM and LOF, continuous attributes are scaled and categorical attributes are coded by using dummy variables. The accuracy is evaluated in terms of the *area under an ROC curve* (*AUC*) [32]. The experiment was performed on a Microsoft Windows 7 machine with an Intel Core i7 CPU (3.20 GHz) and 64 GB RAM.

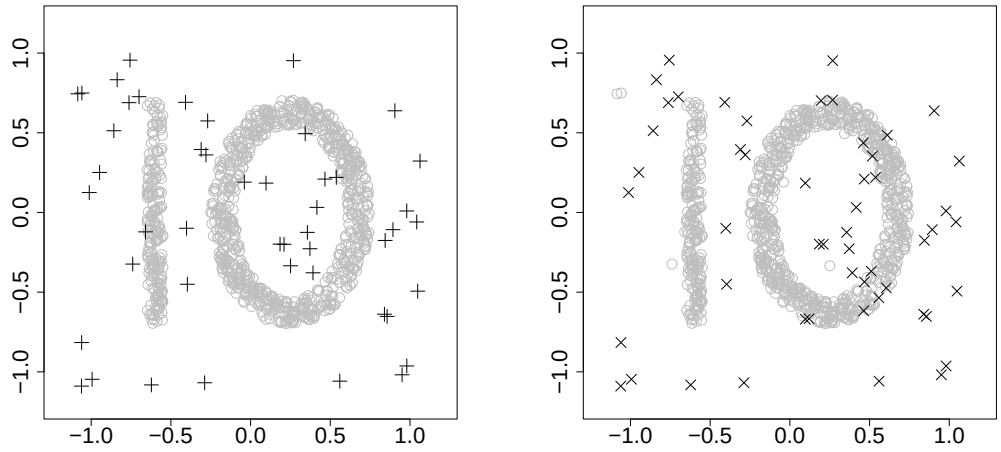


Figure 4.4: An example of the Ten-10^3 dataset in which true outliers are shown as "+"-marks (*left*). The "X"-marks mean outliers detected by the proposed method (*right*).

4.5.1 Evaluation with Synthetic Data Sets

Ten data set is a synthetic data set that consists of two continuous attributes and no categorical attribute. The 95 % data of Ten data set distributes inside the shape "10" randomly. The remaining 5 % data distributes outside randomly, which are regarded as outliers. The number of data is set to 10^3, 10^4, 10^5 or 10^6. An example of Ten-10^3 data set is shown in the left side of Figure 4.4, in which the "+"-marks denote generated outliers.

Table 4.2 shows the mean and the standard deviation of computation time over 10 random trials for Ten data set. We can see that the computation time of the proposed method increases moderately compared to the other methods.

Table 4.3 shows the mean and the standard deviation of AUC values over 10 random trials for Ten data set. From Table 4.3, the accuracy of the proposed method is comparable to those of the other methods. For example, the outliers detected by ODBDD is shown in the right side of Figure 4.4, in which the top 5 % of the data are detected as outliers based on the outlier score of ODBDD.

Table 4.2: The mean and standard deviation of the computation time for the Ten data set (in seconds).

		OCSVM ($\nu = 0.1$)			LOF	
Data set	ODBDD	$\gamma = \gamma_{0.1}$	$\gamma = \gamma_{0.5}$	$\gamma = \gamma_{0.9}$	$k = 10$	$k = 50$
Ten-10^3	0.1 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.4 (0.0)	0.5 (0.0)
Ten-10^4	0.3 (0.0)	3.0 (0.1)	2.7 (0.1)	2.6 (0.1)	22.6 (0.1)	24.8 (0.3)
Ten-10^5	2.3 (0.0)	294.5 (5.7)	267.3 (4.0)	250.6 (4.0)	2942.0 (38.4)	3052.1 (34.3)
Ten-10^6	30.1 (0.9)	timeout	timeout	timeout	timeout	timeout

The timeout limit is 3600 seconds.

Table 4.3: The mean and standard deviation of the AUC values for the Ten data set.

		OCSVM ($\nu = 0.1$)			LOF	
Data set	ODBDD	$\gamma = \gamma_{0.1}$	$\gamma = \gamma_{0.5}$	$\gamma = \gamma_{0.9}$	$k = 10$	$k = 50$
Ten-10^3	0.97 (0.02)	0.79 (0.02)	0.80 (0.03)	0.95 (0.01)	0.99 (0.01)	0.97 (0.02)
Ten-10^4	0.99 (0.00)	0.80 (0.02)	0.82 (0.02)	0.97 (0.00)	0.82 (0.02)	1.00 (0.00)
Ten-10^5	0.99 (0.00)	0.80 (0.00)	0.82 (0.00)	0.97 (0.00)	0.60 (0.01)	0.77 (0.01)
Ten-10^6	1.00 (0.00)	timeout	timeout	timeout	timeout	timeout

Table 4.4: An overview of the UCI data sets used in the experiment.

Data set	N_a	N_m	N_o	# of cate. attr.	# of cont. attr.
abalone	4177	689	7	1	7
adult	32561	24720	248	8	6
bank	45211	39922	400	9	7
ionosphere	351	225	3	0	34
magic	19020	12332	124	0	10
shuttle	43500	34108	342	0	8
yeast	1484	463	5	0	8

4.5.2 Evaluation with Realistic Data Sets

We use seven data sets from UCI machine learning repository [3] as shown in Table 4.4, where N_a is the size of the original data set. All of these data sets are originally arranged for the classification task. In order to apply these data sets to the evaluation of outlier detection algorithms, we randomly picked out data from each data set to generate a new data set as follows: 1) Pick out all the data whose class are C_m where C_m is the class of the maximum data size. Let N_m be the number of data that belong to class C_m. 2) Pick out $N_o = round(0.01N_m)$ data randomly from the remaining data set, which are regarded as outliers.

Table 4.5 shows the mean and the standard deviation of computation time over 10 random trials for UCI data sets. The computation time varies drastically depending on the size of the data set in both OCSVM and LOF. On the other hand, ODBDD works quite fast for all of the data sets.

Table 4.6 shows the mean and the standard deviation of AUC values over 10 random trials. Although some results of ODBDD are not as good as the best result of the other methods, ODBDD achieves similar accuracy to the others.

Table 4.5: The mean and standard deviation of the computation time for the UCI data sets (in seconds).

		OCSVM ($\nu = 0.1$)			LOF	
Data set	ODBDD	$\gamma = \gamma_{0.1}$	$\gamma = \gamma_{0.5}$	$\gamma = \gamma_{0.9}$	$k = 10$	$k = 50$
abalone	0.2 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.3 (0.0)	0.4 (0.0)
adult	3.1 (0.1)	26.5 (0.2)	24.5 (0.1)	24.2 (0.1)	2688.9 (320.0)	2935.3 (459.5)
bank	6.0 (0.0)	66.2 (0.4)	61.5 (0.5)	60.0 (0.3)	2784.8 (672.6)	2328.6 (155.7)
ionosphere	0.2 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.1 (0.0)	0.1 (0.0)
magic	2.4 (0.0)	4.6 (0.1)	4.1 (0.1)	4.0 (0.1)	52.3 (0.7)	56.6 (0.1)
shuttle	2.9 (0.1)	35.6 (0.4)	32.3 (0.3)	31.0 (0.3)	398.4 (10.9)	424.4 (12.5)
yeast	0.2 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.2 (0.0)	0.2 (0.0)

Table 4.6: The mean and standard deviation of the AUC values for the UCI data sets.

		OCSVM ($\nu = 0.1$)			LOF	
Data set	ODBDD	$\gamma = \gamma_{0.1}$	$\gamma = \gamma_{0.5}$	$\gamma = \gamma_{0.9}$	$k = 10$	$k = 50$
abalone	0.59 (0.13)	0.59 (0.08)	0.57 (0.08)	0.58 (0.08)	0.61 (0.11)	0.66 (0.15)
adult	0.59 (0.02)	0.62 (0.01)	0.62 (0.01)	0.61 (0.01)	0.46 (0.02)	0.52 (0.03)
bank	0.61 (0.02)	0.62 (0.01)	0.62 (0.01)	0.62 (0.01)	0.62 (0.01)	0.69 (0.01)
ionosphere	0.87 (0.12)	0.79 (0.08)	0.87 (0.09)	0.64 (0.13)	0.94 (0.07)	0.95 (0.08)
magic	0.79 (0.02)	0.64 (0.03)	0.67 (0.03)	0.71 (0.03)	0.85 (0.03)	0.83 (0.03)
shuttle	0.93 (0.01)	0.78 (0.01)	0.89 (0.01)	0.93 (0.00)	0.44 (0.02)	0.69 (0.03)
yeast	0.65 (0.15)	0.66 (0.11)	0.64 (0.10)	0.55 (0.06)	0.69 (0.15)	0.73 (0.12)

4.6 Summary

In this chapter, we proposed a new approach for outlier detection. A score of being an outlier is defined based on the leave-one-out density, which is evaluated very efficiently by processing a binary decision diagram that represents a data set in a logical formula. The proposed method can deal with a large data set efficiently, because the time complexity is near linear unless the created BDD gets intractably huge. The experimental results indicate that the computation time and the outlier detection accuracy of the proposed method are better or comparable to those of existing methods.

Part III

Diagnosis

Chapter 5

Abstract Model-Based Diagnosis for Detecting Root Causes

In this chapter, we aim to develop a diagnosis method to locate the origin of errors automatically in a system in which the error-propagation problem may occur. We use a model-based diagnosis (MBD) scheme developed in the field of artificial intelligence [26, 58]. It is necessary to model the behavior of each component in a system to apply the original MBD scheme, in other words, we need to describe the relationship between the input and the output of each component in a system precisely. Unfortunately, however, it is very difficult to model the behavior of complex software components, such as ECUs, because they have too many branches in the calculation of their output to describe the relationship between the input and the output. Moreover, it will be computationally intractable to diagnose a system that includes complex software components even though we managed to model their behavior. Therefore, we apply the abstract behavior modeling technique proposed in [35, 67], which enables us to handle complex software components within the MBD scheme without modeling their concrete behavior. Instead, unlike for the original MBD, we need some criteria to judge whether or not data flows in a system are normal. Hereafter, we refer to such MBD as "abstract MBD" (AMBD). Although the original AMBD can be applied to a

system that has no data flow loops, problems occur when it is applied to a system with them. The original AMBD detects no errors when all data flows on a loop are abnormal, because all components on the loop can claim that they output abnormal data due to the abnormal input data. Automotive systems that include ECU components may have data flow loops because most communications among ECUs are mutual and not one-way. In this paper, we propose a modified AMBD to overcome this problem and enable a diagnosis that at least one of the components on a loop is abnormal, even when all data flows on the loop are abnormal.

MBD traditionally employs a two-stage approach based on the notions of conflict and diagnosis. First, conflicts are calculated using a theorem prover [58, 67] or constraint propagation [25]. Second, diagnoses are calculated from conflicts using a hitting set algorithm [58, 67] or a prime implicant algorithm based on Boolean decision diagrams [25]. In this paper, we formulate AMBD into a partial maximum satisfiability problem (PMaxSAT). PMaxSAT is an optimization extension of the Boolean satisfiability problem that has been studied for a long time. Although PMaxSAT is an NP-hard problem, many of large practical problems can be solved by state-of-the-art PMaxSAT solvers due to technical advancements in recent years [30]. Compared with traditional approaches, ours can receive the benefit of modern sophisticated algorithms easily, because sound solvers can be used to diagnose the system. Moreover, we extend our method so as to get diagnoses in the order of their occurrence probabilities.

This chapter is organized as follows: Section 5.1 describes the existing AMBD. Section 5.2 discusses a drawback of the existing AMBD and presents our modified approach. Section 5.3 describes how we formulate AMBD into the partial maximum satisfiability problem and proposes some extensions. Section 5.4 shows some experimental results. Section 5.5 summarizes this chapter.

5.1 Abstract Model-Based Diagnosis

The abstract behavior modeling technique in MBD was first introduced for the purpose of VHDL debugging [35]. Then it was extended to diagnose

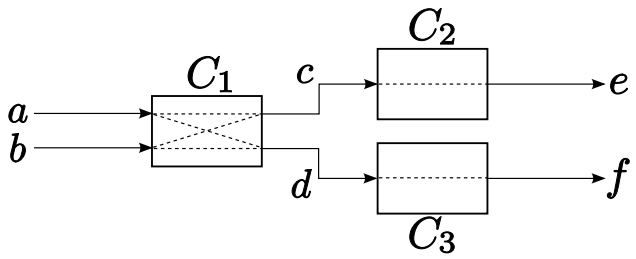


Figure 5.1: An example of the abstracted system.

robotic control systems that include complex software components [67]. We define terms and review AMBD below.

5.1.1 Terms

First, we define terms required to explain AMBD.

Definition 14 $COMP$ is a set of components in the system, and DF is a set of data flows that includes both communication signals among components and input/output signals of the system. $OUT(c)$ is a function that returns a set of output data flows of component c, and $IN(c, s)$ is a function that returns a set of input data flows of component c related to output s.

Example 1 Figure 5.1 is an example of an abstracted system where $COMP = \{C_1, C_2, C_3\}$ and $DF = \{a, b, c, d, e, f\}$. In the figure, $OUT(C_1)$ is $\{c, d\}$, and $IN(C_1, d)$ is $\{a, b\}$ for example.

For an automotive system, we regard ECUs and signals communicated via networks as components and data flows, respectively. Moreover, for a software system with an operating system, we regard tasks and messages among them as components and data flows, respectively.

5.1.2 System Description and Observations

We use the notions of system description (SD) and observations (OBS) in AMBD which is the same as in the original MBD.

Definition 15 SD is a set of first-order sentences that represents obvious relationships among components and data flows in a system. OBS is a set of first-order sentences that represents observations of a system.

In AMBD, SD is basically derived in the following manner:

$$SD = \bigwedge_{c \in COMP} \bigwedge_{s \in OUT(c)} \left[ok(c) \wedge \bigwedge_{s' \in IN(c,s)} ok(s') \rightarrow ok(s) \right], \tag{5.1}$$

where a predicate $ok(x)$ means that x is normal. The idea behind (5.1) is, if both a component and its input data flows are normal, the corresponding output data flow must be normal. The main difference between the original MBD and AMBD is that there is no need to describe components' concrete behavior in AMBD while it is essential to describe every component's behavior in the original MBD.

OBS is defined as a function of $ok(s)$ for $s \in DF$ in AMBD. Unlike for the original MBD, we need some criteria to judge whether or not data flows in the system are normal for AMBD. For example, a criterion is derived based on a periodic feature that data flow s_1 must be produced every n milliseconds [67].

Example 2 SD for Figure 5.1 is given by (5.1) as:

$$\begin{aligned} SD = &\{ok(C_1) \wedge ok(a) \wedge ok(b) \rightarrow ok(c)\} \\ &\wedge \{ok(C_1) \wedge ok(a) \wedge ok(b) \rightarrow ok(d)\} \\ &\wedge \{ok(C_2) \wedge ok(c) \rightarrow ok(e)\} \\ &\wedge \{ok(C_3) \wedge ok(d) \rightarrow ok(f)\}. \end{aligned} \tag{5.2}$$

And if we observe that data flows $\{a, b, f\}$ are normal, $\{c, e\}$ are abnormal, and the state of $\{d\}$ is unknown in Figure 5.1, we get the following OBS:

$$OBS = ok(a) \wedge ok(b) \wedge \neg ok(c) \wedge \neg ok(e) \wedge ok(f). \tag{5.3}$$

Note that literals related to data flows whose states are unknown do not appear in OBS, such as data flow d in (5.3).

5.1.3 Diagnoses

We first define a function $\mathcal{D}$ in order to define diagnoses.

Definition 16 $\mathcal{D}$ is a function that assigns every component in a system as either normal or abnormal. $\mathcal{D}$ is given as follows:

$$\mathcal{D}(C_f) = \left[\bigwedge_{c \in C_f} \neg ok(c)\right] \wedge \left[\bigwedge_{c \in COMP \setminus C_f} ok(c)\right],$$

where C_f is a set of components assigned to abnormal.

We define a diagnosis as follows:

Definition 17 $C_f \subseteq COMP$ is a diagnosis if the following is satisfiable:

$$SD \wedge OBS \wedge \mathcal{D}(C_f). \tag{5.4}$$

Given both SD and OBS, we calculate diagnoses from them using (5.4). However, there may be an exponential number of diagnoses ($2^{|COMP|}$). We define a minimal diagnosis as follows:

Definition 18 A diagnosis C_f is a minimal diagnosis if no subset of C_f is a diagnosis.

Example 3 Let OBS in Figure 5.1 be (5.3). Then we obtain $\{C_1\}$ as a minimal diagnosis by using (5.2), (5.3), and (5.4). Thus, it is possible to locate the origin of errors in a system in which the error that occurred in C_1 propagated to c and e through C_2.

5.1.4 AMBD with Time Consideration

It is possible to incorporate the notion of time in AMBD. Let $TIME$ be a set of discrete time points to be considered, e.g., $TIME = \{0, \ldots, T\}$. Then, we can derive SD that includes delays between input and output as follows:

$$SD = \bigwedge_{t \in TIME} \bigwedge_{c \in COMP} \bigwedge_{s \in OUT(c)} \left[ok(c,t) \wedge \bigwedge_{s' \in IN(c,s)} ok(s',t) \rightarrow ok\left(s, t + \delta(s,c)\right)\right], \tag{5.5}$$

where a predicate $ok(x,t)$ means that x is normal at time t and $\delta(s,c)$ means a delay of output s in component c. OBS is extended to include time and defined as a function of $ok(s,t)$ where $s \in DF$ and $t \in TIME$. We also need to update $\mathcal{D}$ as:

$$\mathcal{D}(C_f^t) = \left[\bigwedge_{\{c,t\} \in C_f^t} \neg ok(c,t) \right] \wedge \left[\bigwedge_{\{c,t\} \in CT \setminus C_f^t} ok(c,t) \right], \tag{5.6}$$

where $CT = \{\{c,t\} | c \in COMP, t \in TIME\}$ and $C_f^t \subseteq CT$. Then, we can calculate diagnoses based on (5.4) where C_f is replaced with C_f^t.

Considering time in AMBD has the advantage that we can deal with intermittent errors because component errors are detected with time-indices. However, we need information about delays between input and output and time-indexed observations to incorporate time in AMBD as mentioned above. For an automotive system, it is quite difficult to estimate input-output delays in each ECU because it changes depending on a lot of factors. Moreover, if we use a periodic feature to judge whether or not data flows in the system are normal, it is not easy to assign the result to particular time point observations. For that reason, we do not consider time in AMBD in the following sections.

5.2 Diagnosing Systems with Loops

AMBD without time consideration works well when a system has no data flow loops, as in Figure 5.1. However, a problem can arise when using AMBD to diagnose systems that have data flow loops. In this section, we explain the problem and propose a modified approach.

5.2.1 A problem of Existing Methods

Figure 5.2 is an example of a system that has data flow loops. By using (5.1),

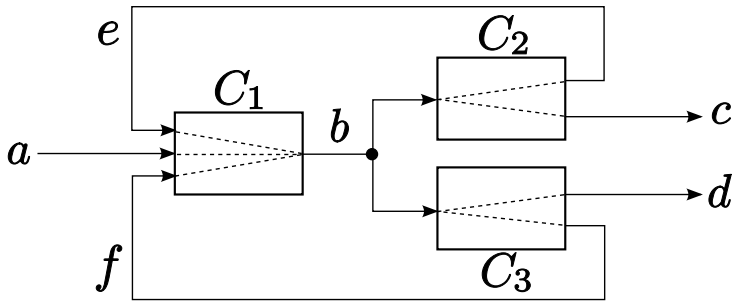


Figure 5.2: An example of a system with data flow loops.

we calculate SD for Figure 5.2 as follows:

$$\begin{aligned} SD = &\{ok(C_1) \wedge ok(a) \wedge ok(e) \wedge ok(f) \rightarrow ok(b)\} \\ &\wedge \{ok(C_2) \wedge ok(b) \rightarrow ok(c)\} \\ &\wedge \{ok(C_2) \wedge ok(b) \rightarrow ok(e)\} \\ &\wedge \{ok(C_3) \wedge ok(b) \rightarrow ok(d)\} \\ &\wedge \{ok(C_3) \wedge ok(b) \rightarrow ok(f)\}\,. \end{aligned} \tag{5.7}$$

Now, assume that we have the following OBS:

$$OBS = ok(a) \wedge \neg ok(b) \wedge ok(c) \wedge ok(d) \wedge \neg ok(e) \wedge ok(f). \tag{5.8}$$

Then, we get an empty set as a minimal diagnosis from (5.4), (5.7), and (5.8). However, it is inappropriate to conclude that no components are abnormal even though abnormal data flows are observed in the system. As this example shows, AMBD described in section 5.1 does not work well when the system has a data flow loop and all data flows on the loop are abnormal.

5.2.2 A Modified Approach

We first define a nonrecurrent loop, and then propose our modified approach. A path p in the system is denoted as:

$$p = (s_1, c_1, s_2, c_2, \ldots, s_k), \tag{5.9}$$

where $s_i \in DF$ and $c_i \in COMP$. A loop is a path such that $s_1 = s_k$ in (5.9). For example, (b, C_2, e, C_1, b) is a loop in Figure 5.2. We define a nonrecurrent loop as follows:

Definition 19 A nonrecurrent loop is a loop that does not contain the same loop more than once.

Example 4 In Figure 5.2, the path $(b, C_2, e, C_1, b, C_3, f, C_1, b)$ is a nonrecurrent loop, whereas the path $(b, C_2, e, C_1, b,\ C_2, e, C_1, b)$ is not one because it contains the loop (b, C_2, e, C_1, b) twice.

Definition 20 Let L_n be a set of all nonrecurrent loops in the system. For $l \in L_n$, $\pi(l)$ is a function that returns a set of unique components in l, $\phi(l)$ is a function that returns a set of unique data flows in l, and $in(l)$ is a function defined as follows:

$$in(l) = \bigcap_{c \in \pi(l), s \in \phi(l)} IN(c, s) \setminus \phi(l).$$

Example 5 In Figure 5.2, there are three nonrecurrent loops: $l_1 = (b, C_2, e, C_1, b)$, $l_2 = (b, C_3, f, C_1, b)$, and $l_3 = (b, C_2, e, C_1, b, C_3, f, C_1, b)$. For example, $\pi(l_1) = \{C_1, C_2\}$, $\phi(l_1) = \{b, e\}$, and $in(l_1) = \{a, f\}$.

In order to overcome the problem mentioned above, we derive SD' as:

$$SD' = \Phi_c \wedge \Phi_l, \tag{5.10}$$

where Φ_c is given by the right side of (5.1) and Φ_l is given as follows:

$$\Phi_l = \bigwedge_{l \in L_n} \left[\bigwedge_{c \in \pi(l)} ok(c) \wedge \bigwedge_{s \in in(l)} ok(s) \rightarrow \bigwedge_{s' \in \phi(l)} ok(s') \right]. \tag{5.11}$$

The idea behind (5.11) is, if all components on a loop and input data flows into the loop are both normal, all the data flows on the loop must be normal. While Φ_c represents obvious relationships that hold in each component, Φ_l represents obvious relationships that hold in each nonrecurrent loop.

Example 6 SD' for Figure 5.2 is given as follows by using (5.10):

$$\begin{aligned} SD' = {} & \text{(the right side of (5.7))} \\ & \wedge \{ok(C_1) \wedge ok(C_2) \wedge ok(a) \wedge ok(f) \rightarrow ok(b) \wedge ok(e)\} \\ & \wedge \{ok(C_1) \wedge ok(C_3) \wedge ok(a) \wedge ok(e) \rightarrow ok(b) \wedge ok(f)\} \\ & \wedge \{ok(C_1) \wedge ok(C_2) \wedge ok(C_3) \wedge ok(a) \rightarrow ok(b) \wedge ok(e) \wedge ok(f)\}. \end{aligned} \tag{5.12}$$

Let OBS be (5.8). Then we obtain $\{\{C_1\}, \{C_2\}\}$ as a set of minimal diagnoses from (5.4), (5.8), and (5.12). Thus, when all data flows on a loop are abnormal, at least one of the components on the loop is abnormal.

It is essential to take nonrecurrent loops into consideration, otherwise problems occur when a system has overlapping elementary loops and all data flows on those loops are abnormal. We show this through the following example.

Example 7 Let SD for Figure 5.2 be (5.13), where clauses related to nonrecurrent loops are removed from (5.12):

$$\begin{aligned} SD = &\text{ (the right side of (5.7))} \\ &\wedge \{ok(C_1) \wedge ok(C_2) \wedge ok(a) \wedge ok(f) \rightarrow ok(b) \wedge ok(e)\} \\ &\wedge \{ok(C_1) \wedge ok(C_3) \wedge ok(a) \wedge ok(e) \rightarrow ok(b) \wedge ok(f)\}. \end{aligned} \tag{5.13}$$

Assume that we have the following OBS:

$$OBS = ok(a) \wedge \neg ok(b) \wedge ok(c) \wedge ok(d) \wedge \neg ok(e) \wedge \neg ok(f). \tag{5.14}$$

Then, we get an empty set as a minimum diagnosis from (5.4), (5.13), and (5.14). This happens because all elementary loops can claim that they output abnormal data due to the abnormal input data from other elementary loops.

5.2.3 Finding All Nonrecurrent Loops

In our modified approach, it is necessary to find all nonrecurrent loops in a system. In this section, we propose an algorithm to find all nonrecurrent loops in a system based on the notion of an elementary loop.

Definition 21 An elementary loop is a loop that does not contain the same data flow more than once except for the beginning and the end of the loop.

Example 8 In example 5, l_1 and l_2 are elementary loops, but l_3 is not an elementary loop because it contains data flow b more than once.

Algorithm 7 Finding all nonrecurrent loops.

1: Calculate all elementary loops L_e in the system.
2: Generate a graph G_e in which a node corresponds to an elementary loop in L_e.
3: **for** every node pairs $\{l_i, l_j\}$ $(i \neq j)$ in G_e **do**
4: Add an edge between them if $\phi(l_i) \cap \phi(l_j) \neq \emptyset$.
5: **end for**
6: Calculate all connected subgraphs in G_e, and denote the set of those subgraphs as S_n.
7: **for** each subgraph $G_n \in S_n$ **do**
8: Calculate π and ϕ of the corresponding nonrecurrent loop l_n as:

$$\pi(l_n) = \bigcup_{l_e \in N(G_n)} \pi(l_e),\ \phi(l_n) = \bigcup_{l_e \in N(G_n)} \phi(l_e),$$

where $N(G_n)$ represents a set of nodes in G_n.
9: **end for**

We propose Algorithm 7 for finding all nonrecurrent loops. We first find all elementary loops in the system, which is done effectively using the BACKTRACK algorithm [68]. Then we express the overlapping of elementary loops as a graph and calculate the connected subgraphs of that graph. Note that each connected subgraph corresponds to a nonrecurrent loop in the system. Finally we calculate a component set and a data flow set of the corresponding nonrecurrent loop.

5.3 Formulation into the Partial Maximum Satisfiability Problem

Two-stage approaches based on the notions of conflict and diagnosis are used traditionally to diagnose systems in MBD [25, 58] and in AMBD [35, 67]. In this section, we propose a new one-stage approach for AMBD. We formulate the problem of diagnosing systems into the partial maximum satisfiability problem that can be solved effectively using state-of-the-art solvers.

5.3.1 The Maximum Satisfiability Problem and Its Extensions

The maximum satisfiability problem (MaxSAT) is an optimization problem of assigning variables so as to satisfy as many clauses in a given Boolean formula as possible. The input Boolean formula is given in conjunctive normal form (CNF). The partial MaxSAT (PMaxSAT) is an extension of MaxSAT where a certain subset of clauses in a given formula is treated as a hard constraint that must be satisfied. In PMaxSAT, unsatisfiable (UNSAT) is returned as a solution if it is impossible to satisfy all hard constraints simultaneously. The weighted MaxSAT (WMaxSAT) is another extension of MaxSAT where each clause has a weight and variables are assigned so as to maximize the sum of weights of satisfied clauses. The weighted partial MaxSAT (WPMaxSAT) is a problem where both hard constraints and weights are considered.

Example 9 Let us consider the following formula with four clauses:

$$(a \vee \neg b) \wedge (\neg a \vee c) \wedge (b \vee c) \wedge \neg c.$$

Assume that the first and second clauses are hard constraints, and the weights of the third and fourth clauses are 3 and 4 respectively. Then, the assignment $(a, b, c) = (0, 0, 0)$ is a solution of WPMaxSAT where the first, second, and fourth clauses are satisfied.

MaxSAT is an NP-hard problem, so it would be intractable as the size of the input formula gets larger. However, due to technical advancements based on the Davis-Putnam-Logemann-Loveland algorithm [24] and its recent extensions, state-of-the-art MaxSAT solvers can solve large practical problems strictly. We used YICES [30] as a MaxSAT solver in our experiments in section 5.4. Note that YICES is a solver for the satisfiability modulo theories problem, but can also deal in MaxSAT (including PMaxSAT, WMaxSAT, and WPMaxSAT).

5.3.2 Formulating AMBD into PMaxSAT

We first need to convert (5.4) into CNF because the input formula of MaxSAT must be in CNF. Suppose that OBS is given in CNF as (5.3). Then we need

only to convert SD' into CNF because $\mathcal{D}(C_f)$ is already CNF. We easily convert SD' into CNF using the following equation:

$$\bigwedge_{x \in X} x \rightarrow \bigwedge_{y \in Y} y = \bigwedge_{y \in Y} \left[\bigvee_{x \in X} \neg x \vee y \right], \tag{5.15}$$

where X and Y are arbitrary sets of variables. We denote the CNF conversion of SD' as SD'_c.

Now, define Δ as:

$$\Delta = SD'_c \wedge OBS \wedge \mathcal{D}(\emptyset), \tag{5.16}$$

where clauses in both SD'_c and OBS are treated as hard constraints. Let x^* be a solution of PMaxSAT of Δ. Then a set of components that are assigned to $\neg ok$ in x^* is a minimal diagnosis of the corresponding AMBD because x^* is a solution that satisfies (5.4) and in which as many components are assigned to ok as possible.

However, we can get only one of the whole minimal diagnoses by solving (5.16) as PMaxSAT. Hence we propose Algorithm 8 for calculating all minimal diagnoses with a PMaxSAT solver. In Algorithm 8, a new clause is added to Δ as a hard constraint in Step 7 so as not to obtain the same diagnosis as a solution of PMaxSAT again. If we do not need to get all minimal diagnoses but only a subset of them, we can add a statement before Step 6 such as "go to Step 8 if the size of C_a exceeds a given threshold." Furthermore, if we would like to get not only the minimal diagnoses but also all possible diagnoses, it is enough to modify the equation for updating Δ' in Step 7 as follows:

$$\Delta' \leftarrow \Delta' \wedge \left[\bigvee_{c \in COMP \setminus C^*} \neg ok(c) \right].$$

5.3.3 Consideration of Error Probabilities

Suppose that a system has n components C_i $(i = 1, \ldots, n)$ and the error probability of C_i is given as p_i. We define x_i as:

$$x_i = \begin{cases} 1 & \text{if } ok(C_i), \\ 0 & \text{otherwise.} \end{cases} \qquad (i = 1, \ldots, n) \tag{5.17}$$

Algorithm 8 Calculating all minimal diagnoses with a PMaxSAT solver.

1: Let $C_a = \emptyset$ and $\Delta' = \Delta$.
2: Solve Δ' as PMaxSAT.
3: **if** Δ' is unsatisfiable **then**
4: Go to Step 8.
5: **end if**
6: Let x^* be the assignment obtained by the PMaxSAT solver, and C^* be a set of components assigned to $\neg ok$ in x^*.
7: Update C_a and Δ' as:

$$C_a \leftarrow C_a \cup C^*,$$
$$\Delta' \leftarrow \Delta' \wedge \left[\bigvee_{c \in C^*} ok(c) \right],$$

where the added clause in the second equation is treated as a hard constraint. Go back to Step 2.
8: Output C_a as a set of minimal diagnoses.

If each component is assumed to break down independently, then the occurrence probability of $x = (x_1, \ldots, x_n)$ is given as follows:

$$P(x) = \prod_{i=1}^{n} (1 - p_i)^{x_i} \, p_i^{1 - x_i}. \tag{5.18}$$

By taking logarithms of both side of (5.18) and transforming its right side, we obtain the following equation:

$$\log P(x) = \sum_{i=1}^{n} \left\{ \log \frac{1 - p_i}{p_i} \right\} x_i + \text{Const}, \tag{5.19}$$

where $\text{Const} = \sum_{i=1}^{n} \log p_i$. From (5.17), (5.19), and the fact that log is a monotone increasing function, it follows that we can get the diagnosis that has the highest occurrence probability by setting the weights of clauses $ok(C_i)$ $(i = 1, \ldots, n)$, which appears in $\mathcal{D}(\emptyset)$ of (5.16), as

$$\log \frac{1 - p_i}{p_i} \quad (i = 1, \ldots, n)$$

respectively and solving (5.16) as WPMaxSAT. Moreover, we can get diagnoses in the order of their occurrence probabilities by applying Algorithm 8 where PMaxSAT is replaced with WPMaxSAT.

5.4 Experiments

In this section, we present experimental results to show how the proposed method works. The experiment is done on a Microsoft Windows XP machine with an Intel Core i7 CPU (2.80 GHz, 3.5 GB RAM).

5.4.1 Diagnosis of An Automotive Control System

System Overview

We assume the automotive control system in Figure 5.3, which is quoted from [60] and modified herein for the sake of simplicity. There are 8 ECUs in the system and they communicate with one another via the controller area network (CAN), which is one of the commonly used automotive networks. The system has 20 data flows, labeled 9 to 28, which are shown in the figure as the numbers on the arrows. The dashed lines in each ECU in the figure indicate dependencies between the input data and the output data of the ECU. We presume the dependencies, because we could not extract information about them from the original article [60].

Observations and Error Probabilities

Assume that we observe the abnormalities of the data flows in the system as shown in Figure 5.3 where solid, dashed, and dotted arrows mean that the data flow is normal, abnormal, and unknown, respectively. Then, OBS is derived in a form like (5.3). We assume that the error probability of ECU i ($i = 1, \ldots, 8$) equals $0.001 \times i$ ($i = 1, \ldots, 8$), respectively.

Deriving the System Description

We implemented the algorithm to derive SD'_c automatically from ($COMP$, DF, IN, OUT) in the system according to (5.10), (5.15), and Algorithm 7.

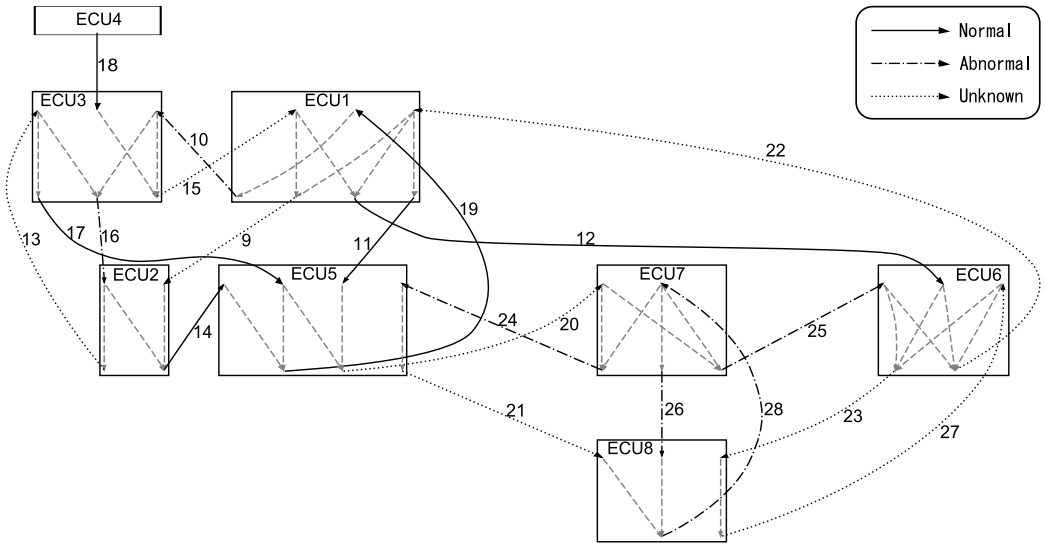


Figure 5.3: Automotive control system that has 8 ECUs and 20 data flows (labeled 9 to 28). The solid arrow means that the data flow is normal, the dashed arrow means that it is abnormal, and the dotted arrow means that its state is unknown.

We found 10 elementary loops and 77 nonrecurrent loops for the system in Figure 5.3, resulting in an SD'_c with 946 clauses. SD'_c became large for the size of the system because the system has many overlapping elementary loops. If the elementary loops were less overlapped, SD'_c would be much smaller.

Diagnosing the System

We implemented the diagnosing algorithm described in section 5.3 and calculated the minimal diagnoses of the system in Figure 5.3. The result is shown in Table 5.1, in which obtained minimal diagnoses are sorted in the order of their occurrence probabilities. We could get minimal diagnoses in the same order as in Table 5.1 one after another by using Algorithm 8. All minimal diagnoses in the table contain ECU1 because the abnormalities of the data flows $\{10, 16\}$ can be explained by error propagation from the abnormality of ECU1. Conversely, the abnormalities of the data flows $\{24, 25, 26, 28\}$ can be

Table 5.1: Diagnostic results of the system in Figure 5.3

Minimal Diagnosis	Probability
{ECU1, ECU8}	7.8×10^{-6}
{ECU1, ECU7}	6.8×10^{-6}
{ECU1, ECU5}	4.9×10^{-6}

explained by error propagation from either ECU7 or ECU8. Moreover, ECU5 can be the origin of error propagation through the data flows $\{24, 25, 26, 28\}$ if we assume that data flow 21 is abnormal. So {ECU1, ECU5} is also regarded as a minimal diagnosis.

The computational time required to get each minimal diagnosis by solving WPMaxSAT is about 0.2 second. Consequently, it takes about 0.6 second in total to get all minimal diagnoses in Table 5.1.

5.4.2 Scalability Test with Synthetic Systems

Another experiment is done to check the scalability of the proposed method. We generated nine synthetic systems randomly. Table 5.2 shows an overview of these systems. In Table 5.2, $|COMP|$ and $|DF|$ means the number of components and data flows in the system respectively. Nonrecurrent loops are calculated by Algorithm 7, then SD'_c is calculated according to (5.10) and (5.15) for each system. In Table 5.2, $|L_n|$ means the number of nonrecurrent loops in the system and $|SD'_c|$ means the number of clauses in SD'_c. We generate an OBS randomly and calculate a minimal diagnosis by solving WPMaxSAT of Δ in (5.16). This procedure was repeated a hundred times for each system. The average time of obtaining a diagnosis is shown in Table 5.2 as $E[time]$ on the second time scale, and its standard deviation is inside the parentheses. Also, the average of peak memory usage is shown in Table 5.2 as $E[peakmem]$ on the kilobyte scale. From Table 5.2, we can see that $|SD'_c|$ largely depends on $|L_n|$, and the average time of diagnosis increases with respect to $|SD'_c|$. The average of peak memory usage increases slightly depending on $|SD'_c|$, on the other hand, its standard deviation increases rapidly as $|SD'_c|$ gets large. We analyzed the results of the systems {F, H,

Table 5.2: An Overview of nine synthetic systems and the average of computational time and peak memory usage for diagnosing these systems.

ID	$\|COMP\|$	$\|DF\|$	$\|L_n\|$	$\|SD'_c\|$	$E[time]$		$E[peakmem]$	
A	50	250	16	571	0.21	(0.01)	564.2	(1.5)
B	50	250	64	2331	0.21	(0.01)	586.8	(38.2)
C	50	250	104	2915	0.21	(0.01)	582.5	(33.0)
D	100	500	8	674	0.21	(0.01)	602.4	(5.3)
E	100	500	75	3640	0.21	(0.01)	623.6	(51.0)
F	100	500	504	23473	0.43	(0.04)	734.0	(333.2)
G	200	1000	19	1403	0.22	(0.01)	679.3	(10.6)
H	200	1000	254	17437	0.53	(0.09)	802.2	(360.6)
I	200	1000	352	22815	0.73	(0.07)	741.4	(349.6)

I} in detail and found that the peak memory usage occasionally gets large in these systems depending on the OBS. It seems that the proposed method is applicable to a system whose size is comparable to that of a system in Table 5.2, because it will take less than a second to diagnose the system with less than a megabyte of memory consumption on average.

5.5 Summary

In this chapter, we proposed a method for diagnosing complex embedded systems that include many software components based on the abstract model-based diagnosis. We modified the existing method for deriving the system description in order to diagnose systems that have data flow loops. We also propose a one-stage approach for solving the abstract model-based diagnosis based on its formulation into the partial maximum satisfiability problem.

It is possible to use the result of system reliability analysis, such as failure mode and effect analysis (FMEA) or fault tree analysis (FTA), to derive the criteria for judging whether or not data flows are normal. Papadopoulos et al. [53] in particular proposed the Interface Focused-FMEA (IF-FMEA) which is a modification of classical FMEA that analyzes how a hardware or software component reacts to failures generated by other components. The

result of IF-FMEA can be applied to compose the diagnostic criteria more effectively. We also need information about the system architecture to derive the system description. There are several languages to describe the system architecture, such as EAST-ADL [28] or architecture analysis and design language (AADL) [33]. Therefore it would be useful to develop tools to extract the system description automatically from source files written in such architecture description languages. Moreover, some of these languages have error modeling functions [34], making it possible to construct the diagnosis system automatically from source files that include information about both system architecture and system errors.

Chapter 6

Conclusion

In this thesis, we developed several mathematical methods for enhancing system reliability. In Part I, we discussed an approach for automatically generating test cases that can be used to test software in the system. In Part II, a one-class classification method and an outlier detection method are proposed, both of which are data-driven methods for detecting anomalous behavior happened in the system. In Part III, a diagnostic method is developed for locating the root cause of errors that are redundantly detected in the system due to the error-propagation problem. Abstraction and logical representation are common technical features of the proposed methods, which plays a key role for constructing efficient algorithms.

In Chapter 2, we proposed a novel approach for test-case generation of software that includes mathematical functions in the scheme of bounded model checking with SMT solvers. The relationship between the input and the output of a mathematical function is abstracted so that the resulting SMT formula can efficiently be solved by existing solvers. The abstraction is refined adaptively based on the previous assignment. We also proposed a method for automatically abstracting mathematical functions by means of machine learning. We would consider several future directions for this work. We have to develop a method to determine which software element to abstract as a function. According to our experimental results, the efficiency of the test-case generation seems to be improved by abstracting as large an element of the software as possible. However, it is not always possible to estimate a

good abstraction using the method proposed in this chapter. In particular, it is difficult to estimate a good abstraction in the case where the function to be abstracted has many inputs. The proposed method raises a new problem in the machine learning field: a new machine learning method that leads to more efficient abstraction is required. Although there are many methods that can achieve more precise prediction than decision tree induction, it is necessary to develop a method whose result can be expressed in a formula that is easily treated by SMT solvers. There still remains the possibility that the efficiency of test-case generation can be improved by using a different strategy in the refinement of abstraction. For example, the fineness of abstraction may be changed in each refinement step, although it was fixed in our experiment. Further, when the formula becomes unsatisfiable, we can slightly extend the region to be abstracted instead of resetting it to the initial region, if the previous assignment is spurious but nearly satisfies the property.

In Chapter 3, we developed a novel one-class classifier using BDDs. Numeric data in a training data set is abstracted by discretization and the region of the data set is expressed as a Boolean function. Then, the region is over-approximated efficiently by manipulating a BDD that represents the Boolean function. The proposed method is parameter-free. In other words, a one-class classifier can be learned automatically from an unlabeled training data set, including the parameter tuning. A large training data set can be efficiently dealt with by the proposed method, because the computational complexity is almost linear with respect to the number of training data. This work can be extended in the following directions: Some of the data set used in the experiments can be considered to face an imbalanced situation. Raskutti et al. [56] showed that one-class learners outperforms two-class learners in some heavily imbalanced situations. Therefore, it is worth investigating whether the proposed method works well with other imbalanced situations. The proposed method may encounter a serious issue, called the curse of dimensionality [69], when the data set includes a large number of attributes. The over-approximating algorithm may fail to work in such a situation, because the density is zero in almost every subregion of the whole hypercube. One simple solution toward these problems is to embed some dimension-reduction techniques, such as principal component analysis,

into an example normalizer. Another possible solution is to select attributes based on the minimum description length principle.

In Chapter 4, we proposed a new method for outlier detection, in which the leave-one-out density is used as a score of being an outlier. We developed an efficient algorithm to evaluate the leave-one-out density using BDDs. The proposed method can deal with a large data set efficiently, because the time complexity is near linear in terms of the size of the data set. This work can be extended in several ways. First, the region set that is used in the evaluation of the leave-one-out density can be enriched by using various example normalizers. Then, the accuracy of outlier detection is expected to improve. A simple approach to generate various normalizers is to incorporate a random rotation into a normalizer. Another extension is to employ nonlinear normalizers. If we use nonlinear normalizers, a hypercube in a projected space corresponds to a nonlinear region in the original space, which may lead to more precise outlier detection.

In Chapter 5, we discussed a method for detecting the root cause of errors in a system in which the error propagation may occur. The abstract model-based diagnosis is used to deal with systems with complex software components. We proposed a method to diagnose systems that have data flow loops. We formulated the problem of solving the abstract model-based diagnosis into the partial maximum satisfiability problem to benefit from the sophisticated solvers. In order to apply the abstract model-based diagnosis, we need criteria to judge whether or not data flows in a system are normal. Therefore it is necessary to develop methods to derive these criteria effectively. One possible approach is to log the data communicated among components and derive the criteria from them using data mining methods, such as those proposed in Chapter 3 and Chapter 4. Another possible approach is to model software components using the framework of formal verification and certify if the criteria holds in the system without any errors, for which the method proposed in Chapter 2 can be utilized. Consequently, the abstract model-based diagnosis can be a bridge between methods proposed in this thesis.

Bibliography

[1] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, TACAS'11/ETAPS'11, pages 254–257, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] A. Asuncion and D. Newman. UCI machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.

[3] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[4] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[5] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[6] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

[7] H. Bengtsson. *R.matlab: Read and write of MAT files together with R-to-MATLAB connectivity*, 2013. R package version 2.0.5.

[8] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 2nd edition, 1999.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[10] M. Borges, M. d'Amorim, S. Anand, D. Bushnell, and C. S. Pasareanu. Symbolic execution with interval solving and meta-heuristic search. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 111–120, Washington, DC, USA, 2012.

[11] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *the 27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[12] B. A. Brady, R. E. Bryant, and S. A. Seshia. Learning conditional abstractions. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, FMCAD '11, pages 116–124, Austin, TX, 2011. FMCAD Inc.

[13] B. A. Brady, R. E. Bryant, S. A. Seshia, and J. W. O'Leary. ATLAS: automatic term-level abstraction of RTL designs. In *Proceedings of the Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 31–40, July 2010.

[14] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[15] M. Breunig, H. Kriegel, R. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *SIGMOD Conference*, pages 93–104, 2000.

[16] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *SIGMOD Conference*, pages 93–104. ACM, 2000.

[17] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[18] B. Caputo, K. Sim, F. Furesjo, and A. Smola. Appearance-based object recognition using svms: Which kernel should I use? In *NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision*, 2002.

[19] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 2009.

[20] R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.

[21] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001.

[22] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. Emerson and A. Sistla, editors, *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer Berlin Heidelberg, 2000.

[23] E. Clarke, A. Gupta, J. Kukula, and O. Strichman. Sat based abstraction-refinement using ilp and machine learning techniques. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 265–279. Springer Berlin Heidelberg, 2002.

[24] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[25] J. de Kleer and J. Kurien. Fundamentals of model-based diagnosis. In *Proceedings of the Fifth IFAC Symposium on Fault Detection, Supervision, and Safety of Technical Processes (Safeprocess)*, pages 25–36, Washington, D.C., USA, 2003. Elsevier.

[26] J. de Kleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[27] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Confer-*

ence on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] V. Debruyne, F. Simonot-Lion, and Y. Trinquet. East-adl – an architecture description language. In *18th IFIP World Computer Congress, ADL Workshop*, pages 53–62, Toulouse, France, 2004.

[29] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2009. R package version 1.5–19.

[30] B. Dutertre and L. de Moura. The yices smt solver. Technical report, SRI International, 2006.

[31] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.*, 69(1-3):35–45, Dec. 2007.

[32] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[33] P. Feiler, D. Gluch, and J. Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, Carnegie Mellon University, 2006.

[34] P. Feiler and A. Rugina. Dependability modeling with the architecture analysis & design language (aadl). Technical report, Carnegie Mellon University, 2007.

[35] G. Friedrich, M. Stumptner, and F. Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(1–2):3–39, 1999.

[36] Y. Fukano, K. Goto, M. Matsubara, Y. Sugure, and Y. Miyazaki. Advanced electronic platform technologies supporting development of complicated vehicle control software. *Hitachi Review*, 63(2):134–137, 2014.

[37] P. Godefroid, N. Klarlund, and K. Sen. Dart: Directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference*

on Programming Language Design and Implementation, pages 213–223, 2005.

[38] S. Hawkins, H. He, G. J. Williams, and R. A. Baxter. Outlier detection using replicator neural networks. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, *DaWaK*, volume 2454 of *Lecture Notes in Computer Science*, pages 170–180. Springer, 2002.

[39] N. He, P. Rummer, and D. Kroening. Test-case generation for embedded simulink via formal concept analysis. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 224–229, 2011.

[40] S. ichi Minato and H. Arimura. Efficient method of combinatorial item set analysis based on zero-suppressed bdds. In *WIRI*, pages 4–11. IEEE Computer Society, 2005.

[41] L. Jaulin, M. Kieffer, and O. Didrit. *Applied interval analysis : with examples in parameter and state estimation, robust control and robotics*. Springer, London, 2001.

[42] T. Kanamori, S. Hido, and M. Sugiyama. Efficient direct density ratio estimation for non-stationarity adaptation and outlier detection. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *NIPS*, pages 809–816. MIT Press, 2008.

[43] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[44] H. Kelly J., V. Dan S., C. John J., and R. Leanna K. A practical tutorial on modified condition/decision coverage. Technical report, NASA, 2001.

[45] E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, editors, *KDD*, pages 206–215. ACM, 2004.

[46] J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.

[47] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of SIAM Conference on Data Mining*, 2003.

[48] E. Loekito and J. Bailey. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos, editors, *KDD*, pages 307–316. ACM, 2006.

[49] P. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[50] K. McCord. *Automotive Diagnostic Systems*. S-A Design Workbench Series. CarTech, 2011.

[51] N. S. Nedialkov, V. Kreinovich, and S. A. Starks. Interval arithmetic, affine arithmetic, taylor series methods: Why, what next? *Numerical Algorithms*, 37(1-4):325–336, 2004.

[52] T. Nolte, H. Hansson, and L. L. Bello. Automotive communications - past, current and future. In *Proceedings of 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA05)*, pages 985–992. IEEE Industrial Electronics Society, September 2005.

[53] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety*, 71:229–247, 2001.

[54] P. Peranandam, S. Raviram, M. Satpathy, A. Yeolekar, A. Gadkari, and S. Ramesh. An integrated test generation tool for enhanced coverage of simulink/stateflow models. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 308–311, 2012.

[55] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[56] B. Raskutti and A. Kowalczyk. Extreme re-balancing for svms: A case study. *SIGKDD Explor. Newsl.*, 6(1):60–69, June 2004.

[57] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and decomposition of binary decision diagrams. In *DAC*, pages 445–450, 1998.

[58] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[59] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[60] Y. Sano, H. Fukatsu, Y. Yahata, K. Sakai, R. Hino, and T. Arashida. Development of new in-vehicle communications system. Technical report, Mitsubishi Motors Technical Review (in Japanese), 2004.

[61] M. Satpathy, A. Yeolekar, and S. Ramesh. Randomized directed testing (redirect) for simulink/stateflow models. In *Proceedings of the 8th ACM International Conference on Embedded Software*, pages 217–226, 2008.

[62] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[63] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[64] K. Sen, D. Marinov, and G. Agha. Cute: A concolic unit testing engine for c. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 263–272, 2005.

[65] F. Somenzi. CUDD: CU decision diagram package. `http://vlsi.colorado.edu/~fabio/CUDD/`.

[66] F. Somenzi. Binary decision diagrams. In *Calculational System Design*, volume 173, pages 303–366. IOS Press, 1999.

[67] G. Steinbauer and F. Wotawa. Detecting and locating faults in the control software of autonomous mobile robots. In *16th International Workshop on Principles of Diagnosis*, pages 13–18, 2005.

[68] R. Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing*, 2:211–216, 1973.

[69] D. Tax. One-class classification: Concept-learning in the absence of counter-examples. *Technische Universiteit Delft, Netherlands*, 65, 2001.

[70] The MathWorks, Inc. http://www.mathworks.com.

[71] T. M. Therneau and B. Atkinson. *rpart: Recursive Partitioning*, 2011. R port by Brian Ripley.

[72] L. Torgo. *Data Mining with R, learning with case studies*. Chapman and Hall/CRC, 2010.

[73] M. van der Laan, S. Dudoit, and S. Keles. Asymptotic optimality of likelihood-based cross-validation. *Statistical Applications in Genetics and Molecular Biology*, 3(1), 2004.

[74] K. Yamanishi, J. ichi Takeuchi, G. J. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Min. Knowl. Discov.*, 8(3):275–300, 2004.

[75] A. Yeolekar, D. Unadkat, V. Agarwal, S. Kumar, and R. Venkatesh. Scaling model checking for test generation using dynamic inference. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 184–191, 2013.

Publications by the Author

Publications related to this thesis

Journal Papers

[J1] T. Kutsuna, Y. Ishii, and A. Yamamoto. Abstraction and Refinement of Mathematical Functions Toward SMT-based Test-case Generation. *The International Journal on Software Tools for Technology Transfer*, The final publication is available at Springer via http://dx.doi.org/10.1007/s10009-015-0389-7.

[J2] T. Kutsuna and A. Yamamoto. A Parameter-Free Approach for One-Class Classification Using Binary Decision Diagrams, *Intelligent Data Analysis - An International Journal*, 18(5):889–910, 2014.

[J3] T. Kutsuna, S. Sato, and N. Chujo. Efficient Root Cause Detection in Complex Embedded Systems with Abstract Model-Based Diagnosis, *Journal of Information Processing*, 20(3):767–773, 2012.

Peer-reviewed Conference Proceedings

[P1] T. Kutsuna and A. Yamamoto. Outlier Detection based on Leave-one-out Density using Binary Decision Diagrams, Tseng, Vincent S., Ho, Tu Bao, Zhou, Zhi-Hua, Chen, Arbee L.P., and Kao, Hung-Yu (eds.), *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science 8444, pp. 486-497, Springer, 2014 (*Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2014)*, Tainan, Taiwan, May 13–16, 2014), The final

publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-06605-9_40.

[P2] T. Kutsuna. A Binary Decision Diagram-Based One-Class Classifier, In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010)*, pp. 284–293, Sydney, Australia, December 13–17, 2010.

[P3] T. Kutsuna, S. Sato, and N. Chujo. Diagnosing Automotive Control Systems Using Abstract Model-Based Diagnosis, In *Proceedings of the 20th International Workshop on Principles of Diagnosis (DX '09)*, pp. 99–105, Stockholm, Sweden, June 14–17, 2009.

Other Publications by the Author

Journal Papers

- T. Kutsuna. Hybrid Models and Reliability Measures for Choice Based Conjoint Analysis (in Japanese), *Journal of Marketing Science*, 17(1,2):31–50, 2009.

- T. Kutsuna, Y. Kai, and M. Fukushima. Optimal Design of PAC-companion Structure for Mortgage Backed Securities Using Cash Reserve (in Japanese), *Transactions of the Operations Research Society of Japan*, 49:62–88, 2006.