Ph.D. DISSERTATION

Approaches to the Design of Machine Learning System

기계학습 시스템 설계를 위한 방법

DECEMBER 2015

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

COLLEGE OF ENGINEERING

SEOUL NATIONAL UNIVERSITY

Kyounghoon Kim

Approaches to the Design of Machine Learning System

기계학습 시스템 설계를 위한 방법

지도교수 최 기 영

이 논문을 공학박사학위논문으로 제출함

2015 년 12 월

서울대학교 대학원

전기·컴퓨터 공학부

김 경 훈

김경훈의 박사학위논문을 인준함

2015 년 12 월

Approaches to the Design of Machine Learning System

위 원 장 : _____김태환_____ (인)

부위원장 : _____최기영_____ (인)

위 원 : _____이혁재_____ (인)

위 원 : _____유승주_____ (인)

위 원 : _____이종은_____ (인)

Abstract

Approaches to the Design of Machine Learning System

Kyounghoon Kim
Department of Electrical Engineering and Computer Science
The Graduate School
Seoul National University

Machine learning has been paid attention because intelligence such as recognition, decision making, and recommendation is a helpful utility in industrial, medical, transportation, entertainment systems, and others that human need to interact with. As machine learning techniques are extensively applied to various areas, the needs for more robust algorithms and more efficient hardware have been increased. In order to develop an efficient machine learning system, we have researched from high-level algorithm down to low-level hardware logic; the main focus of our work is on ensemble machine learning and stochastic computing (SC).

The first work is to combine multiple components, i.e., multiple feature extractors (FE) and multiple classifiers in the aspect of pattern recognition. Ensemble of multiple components is one of challenging approaches for constructing a more accurate classifier [3]. It can handle difficult problems where a single classifier easily makes a wrong decision due to lack of training or parameter optimization [4]. Combining the decisions of participating classifiers statistically reduces the risk of wrong decision. We suggest a hierarchical ensemble framework of multiple feature extractors and multiple classifiers (MFMC).

The second work is to construct efficient hardware building blocks for machine learning in order to reduce system complexity and generate high area- and energy-efficient logic, where we exploit the property of machine learning systems that does not require accurate computations. We select stochastic computing (SC),

which is an alternative paradigm to conventional binary arithmetic computing [5]. SC can boost efficiency in terms of area, power, and error tolerance [6], while relaxing the accuracy of computation.

The third work is to combine both machine learning and stochastic computing, where we select deep learning. This work presents an efficient DNN design with stochastic computing. Observing that directly adopting stochastic computing to DNN has some challenges including random error fluctuation, range limitation, and overhead in accumulation, we address these problems by removing near-zero weights, applying weight-scaling, and integrating the activation function with the accumulator. The approach allows an easy implementation of early decision termination with a fixed hardware design by exploiting the progressive precision characteristics of stochastic computing, which was not easy with existing approaches. Experimental results show that our approach outperforms the conventional binary logic in terms of gate area, latency, and power consumption.

keywords : Machine learning, Stochastic computing, Ensemble learning, Deep learning, Deep neural networks.

student number : 2012-30193

Contents

List of Figures

List of Tables

1. Introduction

1.1 Hierarchical Ensemble Learning Framework

Classification systems can be classified into three categories according to the number of FEs and classifiers as shown in Fig. 1: a single FE and a single classifier (a), multiple classifiers sharing the same feature vector (b), and multiple FEs as well as multiple classifiers (c) which we call MFMC.

As illustrated in Fig. 1 (d), the proposed ensemble system consists of three steps for MFMC: constructing all possible FE-classifier pairs, building a set of local combinations from the set of pairs using reinforcement machine learning, and making a final decision by constructing a global combination based on Bayesian network. In the first step, each FE generates a feature set in a vector format from an input image. The feature vectors from an FE are used by each classifier pairing with the FE for training and testing, which is identical to conventional approach for creating individual recognizers. In the following steps, hierarchical approach is adopted in order to reduce the complexity due to the exponential number of possible combinations. Thus, in the second step, a limited number of FE-classifier pairs are combined to make a group (there can be many different combinations and each combination generates its own group), and weights for the pairs in the group are adjusted according to their effectiveness by using reinforcement learning. Then, in the last step, for a final decision, the groups are merged into a single decision structure called Bayesian network. Experimental results show that the proposed approach gives accuracy higher than any other existing approaches.

1.2 Hardware Building Block for Machine Learning By Using Stochastic Computing

For applications such as machine learning that tolerate a certain level of

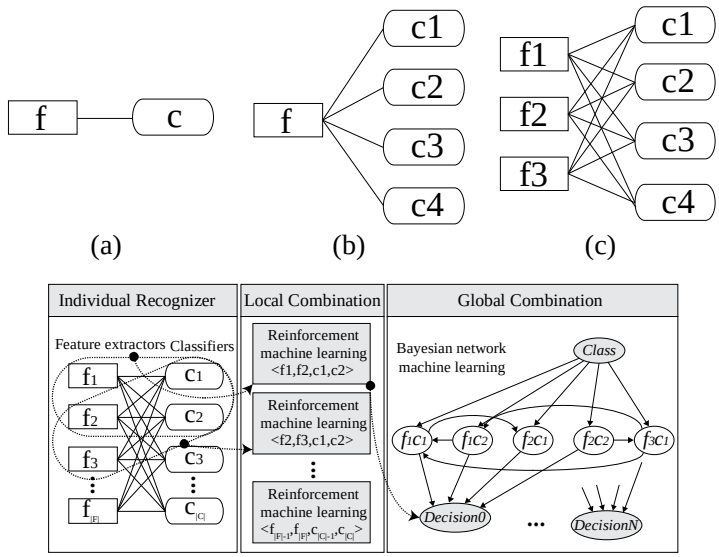


Fig. 1. Example of machine learning system regarding the number of components. ‘f’ stands for a feature extractor; ‘c’ stands for a classifier. (a) Traditional single feature extractor (FE) and single classifier. (b) Single FE and multiple classifiers represented by AdaBoost. (c) Multiple feature extractors and multiple classifiers (MFMC). (d) Proposed hierarchical ensemble of MFMC by using local and global combinations.

inaccuracy, stochastic computing (SC) can be a good alternative to conventional binary arithmetic. SC uses the probability of 1′s in a (pseudo) random bit stream to represent a number as shown in Fig. 2 (a), and allows for an extremely efficient implementation of complex functions (such as multiplication and exponentiation), typically with a few logic gates. (b) shows a complex arithmetic operation using a small number of gates.

Given expressions such as kernels in machine learning, it is very important to generate efficient SC circuits. Thus, we present a SC logic synthesis scheme. As illustrated in Fig. 2, the overall process for the proposed method consists of three

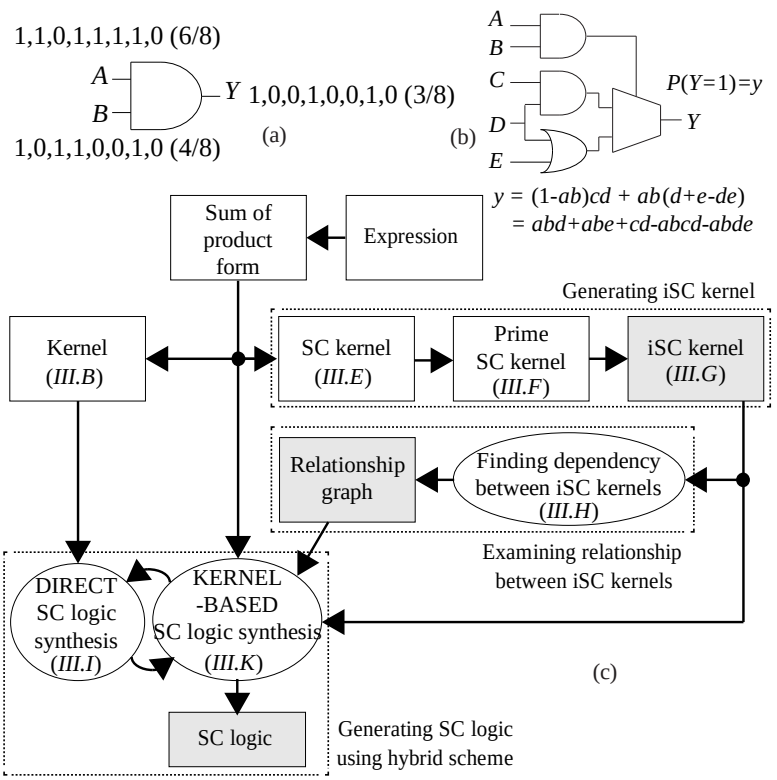


Fig. 2. Stochastic computing (SC) and the proposed algorithm for SC logic synthesis. (a) Stochastic number representation and the multiplication of SC numbers. (b) SC logic example with operations representing y= abd+abe+cd−abcd−abde. (c) Overall process for the proposed SC logic synthesis algorithm.

parts: i) generating iSC kernels, i.e., implementable SC kernels, ii) finding relationship between iSC kernels, and iii) synthesizing SC logic from the original input expression using the iSC kernels and their relationships.

The basic idea is to decompose the input expression into iSC kernels, each of which can be implemented using the SC gates. If some decomposition is derived from the original expression, it is accepted as a solution. There can be many different solutions, and for the exploration, the algorithm tries to divide the given polynomial expression by each iSC kernel. The algorithm pre-examines the

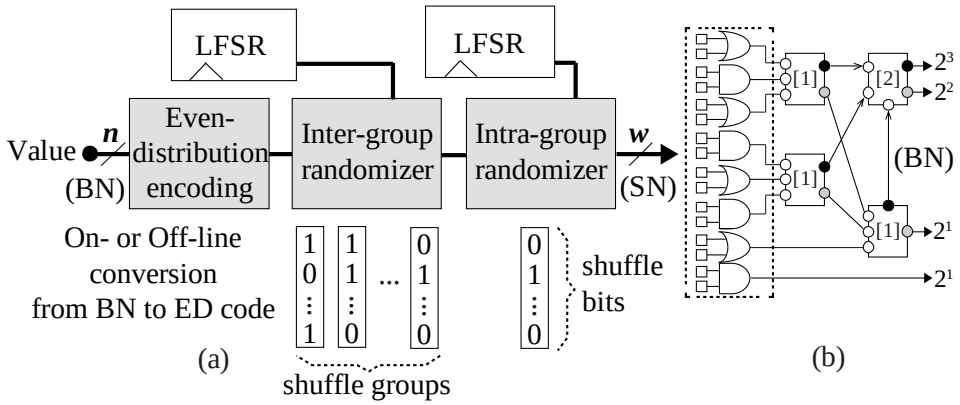
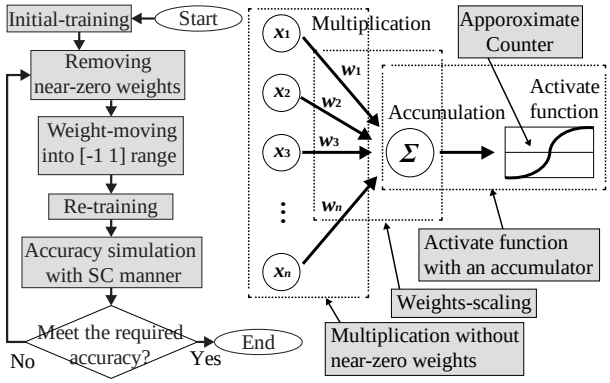


Fig. 3. Overview of the proposed randomizer and de-randomizer for SC. (a) The proposed stochastic number generator (SNG) consists of three parts such as even-distribution (ED) encoding, inter-group, and intra-group randomizer with linear feedback shift register (LFSR) input. (b) The proposed de-randomizer using an approximate unit (AU), converting 16-bit stochastic number (SN) into 4-bit binary number (BN).

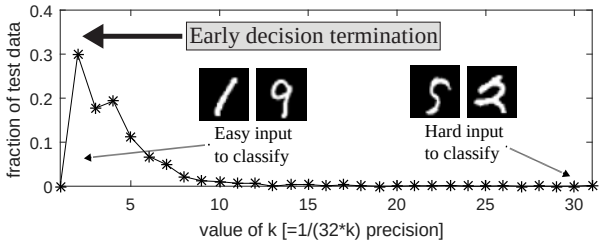
relationships between iSC kernels and exploits them during the search. In the final step, an SC logic network is synthesized for the original expression by using the iSC kernels and the relationships represented as a graph.

Since SC is based on random numbers, a randomizer and a de-generation are very important components. In case of stochastic number generator (SNG) as a randomizer, instead of generating new 0s and 1s, the proposed SNG shuffles 1s in the existing bit stream by using a random source. The basic idea of the proposed SNG is to evenly distribute 1s over the entire bit stream, which is named as *low-discrepancy* (LD). Fig. 3(a) shows the outline of the proposed SNG, which consists of three parts: even-distribution (ED) encoding, inter-group randomizer, and intra-group randomizer.

In case of de-randomizer, we propose an approximate parallel counter (PC) as shown in Fig. 3. (b), which consists of two parts: an approximation unit (AU) and a conventional accurate PC. The approximate PC exploiting an AU is shown in (b).



(a) Training procedures (b) Testing with the SC neuron



(c) Early decision termination

Fig. 4. Overview of the proposed procedures and main idea for deep neural networks (DNNs). (a) Training procedure for DNN using SC with 32-bit floating-point computation. (b) SC neurons are operated with SC exploiting the suggested solutions in testing phase. (c) Early decision termination by using progressive precision of SC.

The input weight of AU is 2^0 while the output weight becomes 2^l, where l is the number of layers.

1.2.1 Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks

Since bringing to break-through in terms of classification accuracy, deep neural

networks (DNNs) have been recently paid great attention. This work presents a method of implementing a DNN using stochastic computing (SC), Based on the observation that directly adopting stochastic computing to DNN has some challenges such as random error fluctuation, range limitation from -1 to 1, and overhead in accumulating many products of inputs and synaptic weights, we address these problems by removing near-zero weights, applying weight-scaling, and using state machine based activation function integrated with the accumulator. We also suggest the early decision termination (EDT) which is very useful in terms of energy and decision speed because most of test inputs are far from decision boundary. The experimental results demonstrate that the accuracy of DNN using SC meets that of conventional floating-point system and the gains in terms of area, power, critical path delay, and energy is meaningful compared with conventional fixed-point arithmetic.

2. A Design Framework for Hierarchical Ensemble of Multiple Feature Extractors and Multiple Classifiers

2.1 Introduction

Ensemble of multiple classifiers is one of promising approaches for constructing a more accurate classifier. It can handle difficult problems where a single classifier easily makes a wrong decision due to lack of training or parameter optimization. Combining the decisions of participating classifiers statistically reduces the risk of wrong decision. In addition, such an ensemble system can generate a sensible solution in a special environment, where several classifiers should be trained with different training datasets due to temporal or spatial constraints. It can also solve instability problems that frequently occur in a single classifier like neural networks with different initial conditions. Another benefit comes from a fact that no single classifier solution can tackle all problems according to the no free lunch theorem (NFL) [3] [4] [7]. Due to these advantages, classifier ensemble has been an active research area in the literature of machine learning and pattern recognition [8] [9] [10]. According to these researches, an ensemble system generates more stable and accurate results compared to conventional single classifier systems.

Considering multiple feature extractors (FEs) and classifiers used for constructing an ensemble system, classification systems can be classified into three categories according to the number of FEs and classifiers as shown in Fig. 1. Conventional classification systems use a single FE and a single classifier as shown in Fig. 1(a). There are systems that use multiple classifiers sharing the same feature vector generated by an FE as shown in Fig. 1(b). AdaBoost is a well-known machine

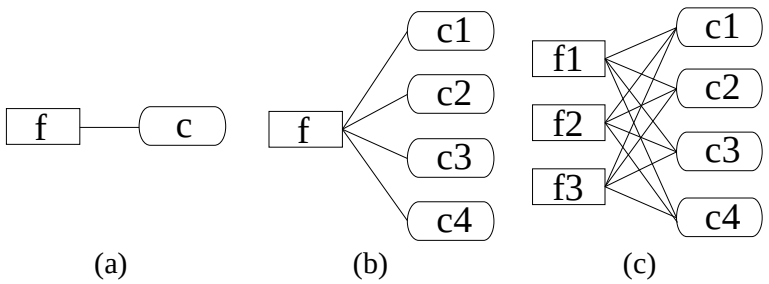


Fig. 5. Example of detection system regarding the number of components. 'f' stands for a feature extractor; 'c' stands for a classifier. (a) Traditional single feature extractor and single classifier. (b) Single feature extractor and multiple classifiers represented by AdaBoost. (c) Multiple feature extractors and multiple classifiers (MFMC).

learning algorithm that supports this model [11]. There have been various researches to extend the concept of AdaBoost for better performance [12] [13]. The third category shown in Fig. 1(c) has been introduced [14] [15] [16]; it uses multiple FEs as well as multiple classifiers and thus we call it MFMC.

One of most representative applications of classifier ensemble is pedestrian detection, which is a key problem in transportation, surveillance, robotics, entertainment systems, and other systems that need to recognize and interact with human [17] [18] [19] [20]. In pedestrian detection, vision based approach is the most effective and popular way. However, it is still quite challenging due to large variations in many aspects such as human clothing, pose, size, background, weather, and illumination. In order to overcome the difficulty, many studies have been conducted in many different ways [21]. However, the achieved accuracy is still insufficient to be used for real applications including advanced driver assistance system (ADAS), thus leaving room for improvement as mentioned by Dollar *et al.* [22]. Especially, in case that a pedestrian is far from the camera or under partial occlusion, the accuracy degrades dramatically. In order to improve the accuracy or detection rate, many studies have tried to find more effective extractors and classifiers such as those in [23] [24] [25] [26] [27] and [28]. The researches have

focused on finding good features as well as good classifiers. Meanwhile, utilizing combination(s) of multiple FEs and classifiers has also been studied; it has a strong advantage compared to single FE, single classifier counterparts.

In this paper, we focus on MFMC since it provides results superior to both single classifier and multiple classifiers with single FE as shown in the previous researches [14] [15] [16] [29]. Contrary to the previous studies that try to find a manually optimized fixed combination of existing FEs and classifiers, we try to optimize automatically the FEs and classifiers as well as their combinations. In particular, we suggest a novel ensemble framework that can manage the complexity generated from MFMC by using a hierarchical method that integrates reinforcement machine learning and Bayesian network modeling. This paper is organized as follows. Section 2.2 gives a brief overview of the related work. Section 2.3 presents the proposed hierarchical ensemble framework for MFMC. Section 2.4 shows experimental results and Section 2.5 concludes the paper.

2.2 Related work

Many ensemble methods have been proposed for a past few decades in the literature of pattern recognition and machine learning. The methods for combining multiple classifiers include weighted majority vote (WMV), naïve Bayes combination (NB) [9], behavior knowledge space (BKS) [30], Wernecke [31], and SVD combination [32]. Moreno-Seco *et al.* [33] suggested extensions to weighted majority vote such as rescaled weighted vote (RSWV), best-worst weighted vote (BWWV), and quadratic best-worst weighted vote (QBWWV). One of other distinguishing approaches is based on genetic algorithm (GA) [34] [35] which use GA to find a better combination without exhaustive search. One study [36] presented an ensemble method with a trained fuser using weights of classifiers, where the optimization problem and solver are proposed. Fuzzy combiner is also used in order to aggregate multiple instances of a classifier for multi-class classification [37]. There are also probabilistic models with posterior estimators [38]

[39].

However, most of existing ensemble schemes to combine decisions do not consider multiple FEs but consider multiple classifiers with a single FE. For MFMC, only a few ensemble methods were presented. Chen *et al.* [40] proposed three general methods—linear combination, winner-take-all, and evidential reasoning—to combine multiple classifiers with different features. They applied them to text-independent speaker identification, where the linear combination with different features (LCDF) outperformed the other two methods. LCDF performs linear combination of the decisions from multiple classifiers, where the learning algorithm uses maximum likelihood estimation with the expectation maximization (EM) algorithm. Zenobi *et al.* [41] presented ensemble of classifiers with different feature subsets considering their diversity. To find the best members of the ensemble, they suggested a hill-climbing algorithm based on the relationship between ensemble accuracy and ambiguity.

There have been a few studies on MFMC in the context of pedestrian detection. Ludwig *et al.* [14] employed both histogram of oriented gradients (HOG) and covariance matrices (COV) consisting of pixel coordination, derivatives, magnitude, and gradient as FEs. They adopted neural networks (NN) and support vector machines (SVM) as classifiers. They also suggested using an ensemble method called 'Training of Fusion Algorithm (TFA)'. Oliveira *et al.* [15] used HOG and local receptive fields (LRF) provided by convolutional neural networks (CNN) as FEs, in which NN and SVM were also employed as classifiers. The work in [16] used HOG and HOF (histogram of optical flow) as FEs and SVM and MPLBoost extended from AdaBoost as classifiers. Those approaches rely on manual optimization of the combinations of MFMC and thus make it hard to have many FEs and classifiers (only two or three of them are allowed). However, our observation is that the FEs and classifiers are complementary with one another, which makes it necessary to combine many of them in order to realize a high

accuracy system. Thus the scalability of an ensemble system is important. Moreover, whenever a new FE or a classifier is added, the entire process of finding the best combination will be repeated and thus automating the optimization process is crucial. The aim of this paper is to construct a framework to overcome the limitations of the previous approaches to MFMC. We investigate this challenging problem and suggest efficient methodologies including experiments for the application of pedestrian detection.

In case of ensemble schemes based on weak-learners (or simple prediction rules) such as bagging [42], boosting [43], and error correcting output codes (ECOC) [44] [45] [46], a large number of the weak-learners commonly participate in the ensemble (i.e., the number of weak-learners can be several thousands), where even a single feature vector can have its own classifier. They focus on selecting better ones among many weak-learners or reformulating the dimensionality such as principal components analysis (PCA) and linear discriminant analysis (LDA). Whereas, in our approach, we focus on finding the best combination among multiple FEs and classifiers. We use a relatively small number of relatively strong learners; simultaneously using a small number of different FEs (e.g., HOG [28], CENTRIST [47], HAAR [48]) and classifiers (e.g., SVM, KNN, and decision tree [49]) can effectively increase the accuracy in practical applications such as pedestrian detection. In the experimental section, we compare the performance of our scheme to that of AdaBoost, a representative ensemble scheme based on weak-learners.

The objective of our work is not to find just the best performed combination of a few FEs and classifiers for pedestrian detection as the work done in [14] [15] [16], but to suggest a general *framework* for designing an ensemble system like the ones in [9] [40] [33]. Note, however, that our approach allows arbitrary numbers of FEs as well as classifiers, whereas the approaches in [11] are limited to systems with a single FE (although they have multiple classifiers), which are much easier to optimize.

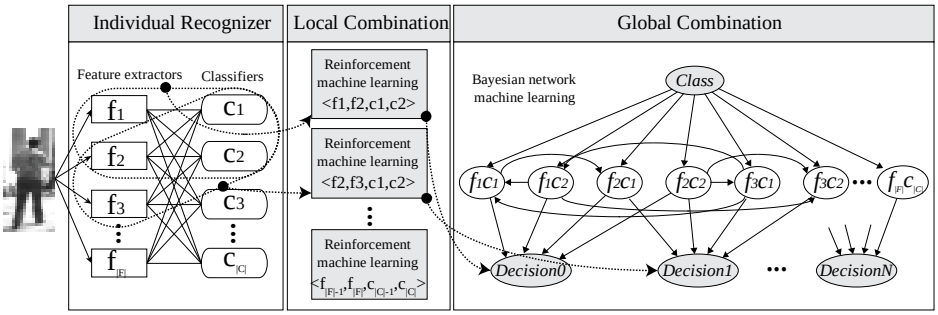


Fig. 6. Overview of the design framework for hierarchical ensemble of MFMC.

2.3 Proposed hierarchical ensemble system

As illustrated in Fig. 6, the proposed ensemble system consists of three steps: constructing all possible FE-classifier pairs, building a set of local combinations from the set of pairs using reinforcement machine learning, and making a final decision by constructing a global combination based on Bayesian network. In the first step, each FE generates a feature set in a vector format from an input image. The feature vectors from an FE are used by each classifier pairing with the FE for training and testing, which is identical to conventional approach for creating individual recognizers. In the following steps, hierarchical manners are adopted in order to reduce the complexity due to the exponential number of possible combinations. Thus, in the second step, a limited number of FE-classifier pairs are combined to make a group (there can be many different combinations and each combination generates its own group), and weights for the pairs in the group are adjusted according to their effectiveness by using reinforcement learning. Then, in the last step, for a final decision, the groups are merged into a single decision structure called Bayesian network.

2.3.1 Local Mapping Block and Global Mapping Block

One of the challenges in ensemble systems for pedestrian detection is that the

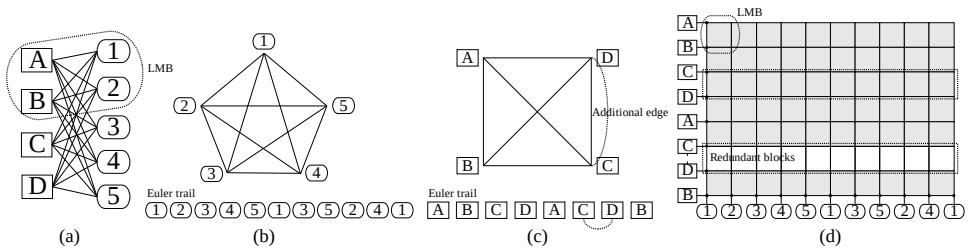


Fig. 7. Constructing GMB from LMBs. (a) 4x5 ensemble of MFMC. (b) Euler trail for a complete graph containing odd number vertexes. (c) Euler path for a graph having even number vertexes where additional edges are inserted in order to build an Euler trail. (d) GMB consisting of LMBs, where redundant blocks show up due to additional edges.

system complexity increases exponentially as an FE or a classifier is added. For instance, Fig. 7 (a) shows a combination composed of four FEs and five classifiers in which all the FEs are fully connected with all the classifiers (it can be modeled as a fully connected bipartite graph). If we consider all different combinations of connections between the FEs and the classifiers, the total number of combinations is $2^{|F||C|}$, where |F| is the number of FEs and |C| is the number of classifiers; therefore, the system has 1,048,576 combinations. Thus finding the best combination looks like an intractable problem. Our suggested scheme is invented for tackling that complexity, satisfying automatic adjustment for ensemble of MFMC.

Since the complexity of MFMC is exponential with respect to the number of possible connections between FEs and classifiers, we consider only a small number (two) of FEs and a small number (two) of classifiers at a time for scalability of the system. For this, we combine each pair of FEs with each pair of classifiers to make a cluster named as a *local mapping block* (LMB). Thus within an LMB, we have

only 2x2=4 possible connections and 2^{2x2}=16 combinations.[1] The LMBs become basic building blocks and merge together into a *global mapping block* (GMB) represented in a two-dimensional space. Fig. 7(d) depicts a GMB for the FEs and classifiers in Fig. 7(a). Each small block represents an LMB for two FEs and two classifiers. For example, the small block on the top left corner represent an LMB with FEs A and B and classifiers 1 and 2.[2] The LMBs in the figure show all different combinations of all different pairs of FEs and classifiers.

To consider all different pairings of classifiers (FEs), we use a complete graph model. Fig. 7(b) (Fig. 7(c)) shows the graph model for the classifiers (FEs) in Fig. 7(a). In Fig. 7(b), for example, each edge represents a different pair of classifiers and any pair is represented by its own edge. To consider all different combinations of pairings as was done in Fig. 7(d), we construct an Euler trail by visiting every edge in the complete graph model. The order of vertices (or nodes) thus obtained is used for constructing the GMB as shown in Fig. 7(d). However, if the number of odd vertices (vertices with odd degree) is greater than two, no single Euler trail can cover all the edges in the graph. In order to make an Euler trail in that case, our scheme inserts additional edges between odd vertices until the number of odd vertices becomes two (c). Note that the additional edges are used to construct redundant blocks in that GMB, which can be ignored in real computations. Since the number of additional edges is at most *(N-2)/2*, the length of the Euler trail is at most *N(N-1)/2 + (N-2)/2*. Hence, the complexity of the proposed scheme (i.e., GMB size) becomes $O(|F|^2 * |C|^2)$; significant improvement is achieved compared to the original combinatorial complexity, $O(2^{|F||C|})$.

---

[1] Thus, if we use only 0 or 1 as the weights for the 2x2=4 classifier instances, we need to consider only those 16 combinations. In reality, however, since we consider real values for the weights, there can be infinite number of different ensembles to be considered.

[2] Note that this LMB has two feature extractors and two classifiers, and thus can have up to 2x2=4 classifier instances: A-1, A-2, B-1, and B-2.

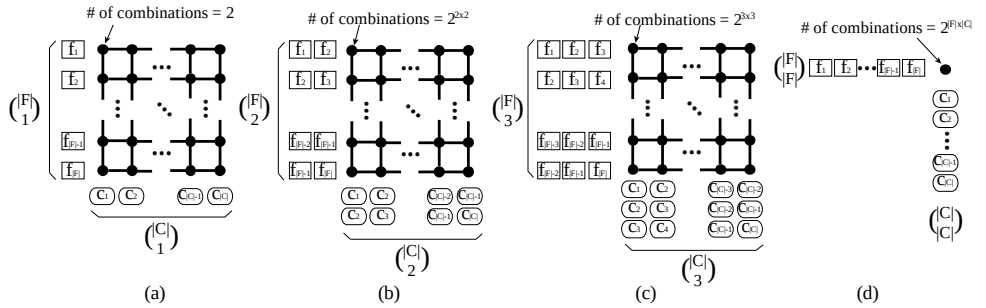


Fig. 8. Different ways of combining FEs and classifiers to make an LMB and using it as the building block for making a GMB. (a) An LMB contains only one FE and one classifier. Although it reduces the complexity, it cannot consider the interaction between FEs or between classifiers. (b) An LMB contains two FEs and two classifiers, which is the proposed scheme. (c) Three FEs and three classifers belong to an LMB. (d) An LMB has all the relations for |F| FEs and |C| classifiers; and thus there is only one LMB in the GMB.

2.3.2 Complexity comparison according to composition of LMB

Any two different FEs may have some relationship so that the two should be considered together in an LMB for more efficient object recognition. The same is true for any pair of classifiers. The relationship is represented by an edge between every pair of vertexes in the complete graph models shown in Fig. 3(b) and (c). If considering three FEs (or classifiers) together is preferred, then the three FEs (or classifiers) are put into an LMB. The triplet corresponds to a clique of size three in the graph model. If we consider only the LMBs with *f* FEs (or *c* classifiers), then we can find all the LMBs by finding all the cliques of size *f* (or *c*) in the graph model of size |*F*| (or |*C*|). The number of cliques consisting of *f (c)* vertexes within a complete graph having |*F*| (|*C*|) vertexes is $\binom{|F|}{f}$. Then the number of LMBs in a GMB is $\binom{|F|}{f} \times \binom{|C|}{c}$, while the number of combinations in an LMB becomes $2^{f \times c}$ when

TABLE I

COMPLEXITY OF A GMB ACCORDING TO LMB COMPOSITIONS

Construction of an LMB		The number of combinations in an LMB	The size of a GMB when $\|F\|=10$ and $\|C\|=10$
The number of FEs	The number of classifiers		
1	1	2 ($=2^{1x1}$)	100 ($=\binom{10}{1}\binom{10}{1}$) [a]
2	2	16 ($=2^{2x2}$)	2,025 ($=\binom{10}{2}\binom{10}{2}$) [b]
3	3	512 ($=2^{3x3}$)	14,400
4	4	65,536 ($=2^{4x4}$)	44,100
5	5	33,554,432 ($=2^{5x5}$)	63,504 ($=\binom{10}{5}\binom{10}{5}$)) [c]
10	10	$2^{10\times10}$ [d]	1

[a] All the relations between FEs and classifiers are ignored.

[b] The proposed scheme considering all couples of relations between FEs and classifiers.

[c] The maximum size of a GMB, when $|F|=10$ and $|C|=10$.

[d] The ideal case representing all combinations; it is intractable.

considering Boolean weights (i.e., 0 or 1). If we consider W discrete weights, $W^{f\times c}$ combinations exist in an LMB. Since our point is to investigate complexity of an LMB with respect to both *f* and *c* rather than the number of quantization steps of weights, hereafter, we assume Boolean weights for the discussion of complexity.

Fig. 8 shows the GMB structure varying with different compositions in terms of the number of FEs and classifiers within an LMB. Fig. 8(a) depicts a GMB consisting of LMBs, each of which has only one FE and one classifier. The GMB comprises $|F|\times|C|$ LMBs, where the synergistic effect of multiple FEs (or

multiple classifiers) is not considered. If the number of FEs and classifiers in an LMB increases, the complexity of the LMB grows to $2^{f\times c}$ (finding the optimum among $2^{f\times c}$ combinations). Fig. 8(a) has 2^{1x1} combinations, while the number of combinations of (b), (c), and (d) become 2^{2x2}, 2^{3x3}, and $2^{|F|x|C|}$, respectively. Note that (d) corresponds to the original full combination problem, where the GMB has only a single LMB. A general formula for the number of LMBs in a GMB is given by the number of cliques within two complete graphs, one for FEs and the other for classifiers. The number of cliques for the FEs is given by

$$\binom{|F|}{f} = \frac{|F|!}{f!(|F|-f)!} = \frac{|F|(|F|-1)\cdots(|F|-f+1)}{f(f-1)\cdots 1} = \frac{|F|}{f}\frac{|F|-1}{f-1}\cdots\frac{|F|-f+1}{1} \le |F|^f. \qquad (1)$$

The number of cliques for the classifiers can be obtained similarly. Therefore, the complexity of a GMB is $O(|F|^f|C|^c)$. For *f*=*c*=2 as suggested in this paper (see Fig. 8(b)), an LMB has 16 (=2^{2x2}) possible combinations and the GMB has complexity of $O(|F|^2|C|^2)$ in terms of the number of LMBs included in the GMB. When both numbers of FEs and classifiers in an LMB are three[3] as shown in Fig. 8(c), an LMB has 512 (=2^{3x3}) combinations, while the GMB has $O(|F|^3|C|^3)$ complexity. Thus, assigning small values to *f* and *c* renders a polynomial complexity of the problem. As either *f* or *c* grows, however, the number of possible combinations for an LMB increases exponentially ($2^{f\times c}$). The size of a GMB also tends to grow as *f* or *c* grows toward $\left\lfloor\frac{|F|}{2}\right\rfloor$ or $\left\lfloor\frac{|C|}{2}\right\rfloor$, respectively, as shown in Table I. Note that the suggested 2x2 combination considers all the pairs of FEs and classifiers.

2.3.3 Motivation for differentiating local and global mappings

[3] For simplicity, we assume here that the number of FEs and that of classifiers in an LMB are the same. However, they do not necessarily have to be the same. For instance, two FEs and four classifiers are acceptable.

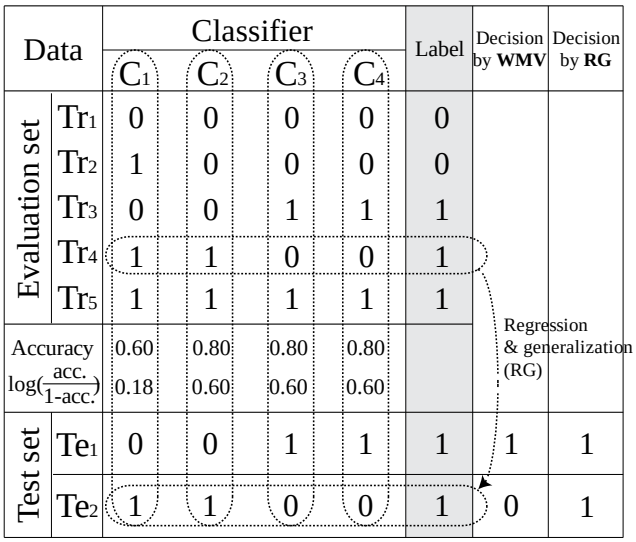


Fig. 9. Two directions for combining multiple classifiers. Weighted majority vote takes the vertical direction, while regression and generalization takes the horizental direction. Two methods can generate diffierent results even for identical inputs.

Fig. 5 illustrates two directions of utilizing the evaluation results of classifiers for making a decision and shows that the final results can be different depending on the direction. The example has five data inputs in the evaluation set and two data inputs in the test set; four classifiers participate in the ensemble system. According to the figure, the four classifiers give the evaluation results of <0, 0, 0, 0>, <1, 0, 0, 0>, <0, 0, 1, 1>, <1, 1, 0, 0>, and <1, 1, 1, 1> for the five evaluation data with labels 0, 0, 1, 1, 1, respectively. Thus, the accuracies of the classifiers are 0.60, 0.80, 0.80, and 0.80, which are calculated by the number of correct decisions over the number of total data. Using the weighted majority vote (WMV) with the optimal value $\log(\frac{acc.}{1-acc.})$, the ensemble decision for test data Te_1 (based on the individual results of the classifiers) is ′1′, which is correct. In the same way, the ensemble decision by WMV for test data Te_2 is ′0′. However, the individual results <1, 1, 0, 0> of the

classifiers are the same as those for Tr_4 in the evaluation set, and thus the desirable ensemble decision is not ′0′ but ′1′, as made by the regression and generalization (RG) method. WMV takes the vertical direction for calculating the accuracy (or priority) of each classifier and thus does not consider characteristics of each data input. On the contrary, RG takes the horizontal direction to combine all the decisions made by the classifiers for each data input. An interesting question is which direction provides higher performance. Instinctively, if a decision system is mixed with weak classifiers as well as strong classifiers, WMV is more beneficial than RG, because it restrains a weak classifier from confusing the decision making process by assigning a low priority to the weak classifier. On the other hand, if a decision system consists of enough strong classifiers, RG can be more efficient, since it considers mutual relationships between classifiers for a specific data input as shown in Fig. 5. For such a reason, the two directions have their own advantage.

Our hierarchical framework considers all the directions for the combination of classifiers. Since an LMB can be comprised of weak and strong classifiers, it is suitable for the vertical direction method such as WMV. On the other hand, GMB is suitable for the horizontal direction because the LMBs generated by the first-level ensemble of classifiers are in general stronger than the original classifiers.

2.3.4 Reinforcement learning for LMB

One of the most popular methods for combining classifiers is WMV which gives more weight to classifiers having higher accuracy. Given the accuracy of each classifier, the optimal weight is determined by $w_i \propto log\frac{p_i}{1-p_i}$, where w_i is the weight value of classifier instance i and p_i is the accuracy obtained by evaluating the classifier using a training set [9]. However, this is correct only under the condition that all classifiers have no dependency on others, that is only when the original posterior probability

$$P(\boldsymbol{D}|h_k) = P\left(D_{m_1}\middle|h_k\right)P\left(D_{m_2}\middle|D_{m_1}, h_k\right)$$

$$\ldots P(D_{m_L}|D_{m_1}, \ldots, D_{m_{L-1}}, h_k) \quad (2)$$

can be treated as

$$P(\boldsymbol{D}|h_k) = \prod_i^L P(D_{m_i}|h_k), \quad (3)$$

where L is the number of classifiers, $\boldsymbol{D}$ is the event where the decision vector from the L classifiers is $(m_1, m_2, \ldots, m_L)$, D_{m_i} is the event where the decision by the i-th classifier is m_i, and h_k is the event where the input is in class k. In practice, the assumption is not satisfied. The experimental results in Section 2.4 show that the assumption of independent classifiers is wrong and the bias caused by it adversely affect the final decisions.

In order to overcome the limitation of the original WMV, we utilize *Reinforcement learning* (RL) to compute real weight based on trial-and-error. RL was invented for tackling problems with an environment that is unknown or cannot be modeled. It solves such problems by communicating with the environment through interactions [50]. The RL task commonly consists of five components: An agent, set S of states, set A of actions, rewards, and an environment. In order to get the best reward, an agent explores the space of environment with two strategies: exploitation and exploration. While the former is to follow an existing policy[4] to get the best reward, the latter is to search for a new policy in order not to trap into a local optimum (known as exploration and exploitation trade-off). At time step t, the agent makes a decision to take action $a_t \in A(s_t)$ at state $s_t \in S$ of the environment. The environment gives a reward r_{t+1} to the agent that updates the current state value and moves to new state s_{t+1}. The objective of the RL task is to

[4] Mapping states to actions is called policy π. An agent picks an action a under policy π in current state s.

maximize the expected value of return R_t ,

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad (4)$$

where $\gamma, 0 \leq \gamma \leq 1$ is a discount factor.

RL performs learning on what to do in the current state in order to maximize the final return value. In WMV, for example, suppose the weights of two classifiers, *A* and *B*, are w_A and w_B, respectively. RL can give an answer to a question on which classifier should increase or decrease its weight in order to achieve the maximum accuracy, given w_A, and w_B (i.e., the current state). Let us assume that $v(w_A) = \alpha, \beta,$ and γ ($\alpha > \beta$ and $\alpha < \gamma$) at $w_A = 0.3$, 0.4, and 0.9, respectively, where $v(w_A)$ is the return value function of weight w_A. The action to be taken at $w_A = 0.3$ is to increase w_A to 0.4, even though $v(0.3)$ is greater than $v(0.4)$ (i.e., $\alpha > \beta$), because further increase of w_A to 0.9 will achieve the maximum return value γ. In this case, the future value $v(0.9)$ affects the current decision (to increase the weight). The objective of RL is to move rapidly towards the optimal state while updating the values for actions so that it can search of the solution space more efficiently.

Let $\boldsymbol{w_{RL}} = (w_1^*, w_2^*, ..., w_L^*)$ denote the vector of weights induced by RL for WMV of multiple classifiers considering the dependency in (2), and $\boldsymbol{w_0}$ denote the original optimal vector of weights obtained by $log\frac{p_i}{1-p_i}$. Then the accuracy of $P(\boldsymbol{D}|h_k)$ with given $\boldsymbol{w_{RL}}$ is greater than or equal to the accuracy of $P(\boldsymbol{D}|h_k)$ with given $\boldsymbol{w_0}$. The reason is that the weights of classifiers are generated by RL considering the dependency among classifiers, because RL reflects the environment including dependency. In the experimental section, with UCI dataset [51], we show

the effectiveness of RL for adjusting weight values of classifiers, which is not possible with conventional WMV.RL is adopted to get weights of 2x2 classifiers in an LMB to compute the best weights considering dependency among classifiers. A challenge of using RL in calculating weights for classifiers in an LMB is that the number of states is infinite, because the value of weights is continuous. To cope with the challenge, function approximation was introduced in [50], where training task updates parameter vector $\vec{\theta}$ representing the function; the state action value Q_t depends on a parameter vector $\overrightarrow{\theta_t}$ at time t. Especially, tile-coding method as a kind of gradient-descent approach and eligibility tracing were adopted in order to boost up training speed. For tile-coding, a continuous space is managed by *tiling* which splits the space of state into several partitions called *tiles* [52]. We first formulate the problem of calculating weights of classifiers for the LMB in the RL as follows:*S*: Considering that each LMB has four classifier instances, we define the total state space as $S = w_1 \times w_2 \times w_3 \times w_4$ with constraint $\sum_i^4 w_i = 1$, where w_i is the weight of classifier instance i. Initial weights are set to the optimal weights under the independence condition in order to have an efficient convergence in the training phase. *A*: There are eight actions for four classifier instances within an LMB per state. Each classifier has two actions: increasing and decreasing the weight of the current state, where an amount of changing the weight is calculated from gradient of current state. The weights are then normalized according to the constraint $\sum_i^4 w_i = 1$. In order to explore the state space of weights of four classifiers during the action selection, we use ε-greedy action selection method at time *t* as follows:

$$a_t = \begin{cases} \text{argmax}_a Q(s_t, a) \text{ with probability } 1 - \varepsilon \\ \text{random action with probability } \varepsilon \end{cases}$$

REINFORCEMENT-LEARNING-FOR-WEIGHT-OF-LMB

Initialize $\vec{\theta}$ and $\vec{e} = 0$

s $\leftarrow \{ \ log\frac{p_i}{1-p_i} | p_i \forall_{c_i} \}$, where p_i is accuracy of classifier c_i

For all $a \in A(s)$

    $\vec{b}(s,a) \leftarrow$ set of features of tile coding in s, a

    $Q(s,a) \leftarrow \sum_{i=1}^{n} \theta_i \times b_i(s,a)$, where n is the number of tilings

Repeat (for each step):

    With probability 1-ε:

        $a \leftarrow \text{argmax}_a Q(s,a)$

        $\vec{e} \leftarrow \lambda \vec{e}$

    else

        $a \leftarrow$ a random action $\in A(s)$

        $\vec{e} \leftarrow 0$

    For all $\{i | b_i(s,a) = 1\}$:

        $e_i \leftarrow e_i + 1$

    Set next state $\acute{s}$, with the action a

    For all $a \in A(\acute{s})$:

        $\vec{b}(\acute{s},a) \leftarrow$ set of features of tile coding in $\acute{s}$, a

        $Q(\acute{s},a) \leftarrow \sum_{i=1}^{n} \theta_i \times b_i(\acute{s},a)$

    Calculate a reward computed by a difference of accuracy for an LMB between state s, $\acute{s}$

    $\vec{\theta} \leftarrow \vec{\theta} + \alpha(reward - Q(s,a) + \max_a Q(\acute{s},a))\vec{e}$

    $s \leftarrow \acute{s}$

Until $\vec{\theta}$ converges.

Fig. 10. Psedo-code of the reinforcement learning algorithm for calculating weights for an LMB.

Reward at a state: Reward value corresponding to an action at a state, which utilizes the accuracy of an LMB, is calculated as the amount of the increase in the

sum of number of true positives and number of true negatives divided by the number of samples.

Fig. 10 is a pseudo-code of RL for computing weights in LMBs, where the RL task has only one episode that is terminated when $\vec{\theta}$ converges. In line 2, initial state is induced from p_i for efficient and fast training, which is an optimal weight under the assumption that all the classifiers are independent. The set of features of tile coding $\vec{b}(s,a)$ is generated at the current state and for all the actions, where $\vec{b}(s,a)$ contains binary values (1 or 0) about all the tiles; if a tile fits into the state, the value of $\vec{b}(s,a)$ for the tile is one; otherwise, the value is zero. State action values $Q(s,a)$ are computed by taking the sum of products of $\vec{\theta}$ and $\vec{b}(s,a)$ in lines 3 to 5, which is repeated in lines 16 to 18. In lines 7 to 12, the state space is exploited and explored with ε-greedy action selection and eligibility trace $\vec{e}$ is executed. The values of eligibility trace used in the step are updated in lines 13 and 14. After choosing next state with the selected action and calculating the state action value with respect to the next state, the reward is computed by taking the difference of accuracy between the current state and the next state in lines 15 to 19. Finally, the parameter vector $\vec{\theta}$ is modified in line 20, which is followed by updating the current state; the task is repeated until $\vec{\theta}$ converges.

2.3.5 Construction of Bayesian network from GMB

Once the weights of the FE-classifier pairs in the LMBs are adjusted using RL as explained in Section 2.3.4, each LMB can make its own decision based on WMV. Then the decisions from all the LMBs in the GMB are used to make a final decision. We consider each LMB as a strong classifier since it has been constructed by the first-level ensemble of classifiers, which is supported by the experimental results in Section 2.4.

Although the LMBs are considered as stronger classifiers, they are still dependent

on one another and thus the second-level ensemble is constructed by adopting RG approach to consider horizontal direction of combination of classifiers (i.e., LMBs) as mentioned in Section 2.3.3. For this, we adopt Bayesian network (BN) that can be used to make a final decision considering the dependencies among LMBs. The dependencies are represented by the conditional probabilities of the first-level decisions made by the FE-classifier pairs. Since BN gives the posterior probability for a given occurrence of events, it can be used as a classifier [53]. The reason for selecting BN is that not only can it infer the class of an arbitrary input with a generalization method, but also it can consider the dependencies of LMBs. BN is a graph model with nodes and arcs for representing conditional probabilities among random variables. Each node represents a random variable with a conditional probability function and each arc represents a dependency between two random variables.

To formulate the dependency between LMBs within a GMB, let $F = \{f_1, f_2, f_3, \dots, f_M\}$ denote a set of FEs where M is the number of FEs, and $C = \{c_1, c_2, c_3, \dots, c_N\}$ denote a set of classifiers, where N is the number of classifiers; the total number of pairwise connections between FEs and classifiers becomes $M \times N$. Suppose complete graphs composed of F and C are G_f and G_c respectively as mentioned in Section 2.3.1, and the sets of edges for G_f and G_c are E_f and E_c, respectively. Then the total number of LMBs becomes $E_f \times E_c$. An LMB is defined as a tuple $< f_i c_k, f_i c_l, f_j c_k, f_j c_l >$, representing 2x2 combination containing two FEs and two classifiers, where $f_i, f_j \in F$and $c_k, c_l \in C$; $(f_i, f_j) \in E_f$and $(c_k, c_l) \in E_c$ are edges in the complete graphs.

Assuming that the set of classes is $H = \{h_1, h_2, h_3, \dots, h_L\}$, the classification problem using GMB is to find a class of maximum probability as follows,

$$decision = \text{argmax}_{h_l} P\left(h_l \middle| D_{LMB_1}, D_{LMB_2}, \dots, D_{LMB_K}\right)$$

$$= \text{argmax}_{h_l} \frac{P(h_l) \cdot P\left(D_{LMB_1}, D_{LMB_2}, \dots, D_{LMB_K} \middle| h_l\right)}{P\left(D_{LMB_1}, D_{LMB_2}, \dots, D_{LMB_K}\right)}$$

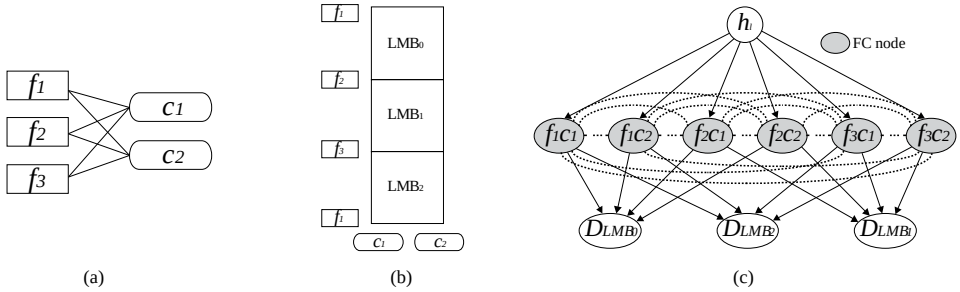


Fig. 11. Construction of Bayesian networks. (a) The system consists of three feature extractors and two classifiers. (b) GMB is created by LMBs for the system. (c) Bayesian network is composed of 1) class node denoted as hl, 2) FC nodes denoted as fick, which correspond to FE-classifier pairs, and 3) decisions of LMBs denoted as DLMBk .

$$\cong \text{argmax}_{h_l} P\left(D_{LMB_1}, D_{LMB_2}, \dots, D_{LMB_K} \middle| h_l\right), \quad (5)$$

where D_{LMB_k} is a decision from an LMB_k. Because the prior probability $p(h_l)$ is unknown in the testing and joint probability $P(D_{LMB_1}, D_{LMB_2}, \dots, D_{LMB_K})$ is constant, they are removed.

There are two key tasks to perform to construct a BN. One is building a BN structure to represent the data model and the other is finding conditional probability functions of nodes. Building the BN structure is to find pairs of nodes having a significant dependency and let the network structure satisfy the properties required to solve given problems. The problem of finding an optimal structure of a BN has an exponential complexity, since the total number of possible combinations for the BN is

$$f(n) = \sum_{i=1}^{n} (-1)^{(i+1)} \frac{n!}{(n-1)!i!} 2^{i(n-1)} f(n-1), \quad (6)$$

where *n* is the number of nodes [54]. For example, 3,781,503 possible graph

structures can be generated for a network having only six nodes.

In order to tackle the difficulty, in our solution, constructing a BN structure has two steps: building an initial skeleton network with prior information and refining the structure iteratively using a score function that represents how well the structure fits. The initial BN consists of three kinds of nodes: nodes for representing classes, nodes for classifier instances (FE-classifier pairs), and nodes for decisions of LMBs such as D_{LMB_k}. After preparing an initial BN, the structure is refined based on dependencies between FE-classifier pairs. Fig. 11 shows an example of building a BN structure, where a system is composed of three basic FEs and two basic classifiers (a). The system first builds three LMBs: LMB_0 with $<f_1c_1, f_1c_2, f_2c_1, f_2c_2>$, LMB_1 with $<f_2c_1, f_2c_2, f_3c_1, f_3c_2>$, and LMB_2 with $< f_3c_1, f_3c_2, f_1c_1, f_1c_2>$ (b). The initial skeleton network structure is generated as shown with solid arrows (directed edges) in (c). Each solid edge represents direct influence from the predecessor to the successor. The edges are fixed in the network, because they are expected to have strong relationships between nodes.

The number of nodes in a BN generated for a GMB is $|F||C| + \binom{|F|}{2}\binom{|C|}{2} + 1$, where the three terms respectively account for the number of nodes corresponding to f_ic_ks (called FC nodes), the number of LMBs, and the node for class h. Once the initial skeleton network is constructed, the edges to be added are only between FC nodes and thus the number of nodes to be considered reduces to $|F||C|$.[5] This is meaningful because the second term for LMBs, $\binom{|F|}{2}\binom{|C|}{2}$, dominates the complexity for building a BN when both $|F|$ and $|C|$ are large. For example, the

[5] The FC nodes representing f_ic_ks construct a BN as a subset of the entire BN for a GMB, and thus they still have a constraint that no cycle exists. Since h_l has only outgoing edges and D_{LMBs} have only incoming edges as shown in Fig. 10 (c), no cycle containing either h_l or D_{LMBs} can be made by adding an edge between f_ic_ks (a cycle containing only f_ic_ks can still happen).

number of nodes is reduced from 19 to 9, given that $|F|$ and $|C|$ are three; it is reduced from 53 to 16 when $|F|=|C|=4$; the gain from this method is further magnified beyond four. Thus the complexity of the problem of building connections between nodes is also reduced. For the case of $|F|=|C|=3$, the number of possible ways to construct a BN is diminished from about $3.3\text{x}10^{65}$ (19 nodes) to $1.2\text{x}10^{15}$ (9 nodes) as calculated by (6). Nevertheless, the problem is still intractable and thus requires an efficient heuristic method. There have been many heuristic algorithms for building a BN including Markov chain Monte Carlo (MCMC), variational inference, K2, Chow–Liu trees methods, and their extensions [55] [56] [57] [58]. However, most of the algorithms treat each node in the BN with same priority. On the other hand, we pay attention to difference characteristics of different FC nodes; the FC nodes sharing the same FE and/or classifier tend to have higher dependency on each other. In addition, some FC nodes have higher accuracy than others and thus have higher priority. Finally, the BN that we propose can be constructed from an initial skeleton network, which we can also exploit.

Thus, to devise our own heuristic method, we consider meaningful relationships among FC nodes. Each dotted line in Fig. 7(c) illustrates the relationship between two FC nodes. As mentioned above, two FC nodes possibly have strong association with each other when they share either the same FE or the same classifier. For instance, since f_1c_1 and f_1c_2 share the same FE, the two FC nodes have a high chance of getting an edge between them when building the BN. However, if they have a weak dependency having little effect on the performance of the system, the edge is removed. This is done through learning for which there are two well-known algorithms: score-based approach and constraint-based approach. The score-based approach defines a scoring function for the BN structures to represent the fitness for the given case and attempts to find a structure that maximizes the score. The constraint-based approach exploits constraint conditions such as independency between two nodes. We use the score-based method since it shows better performance in general and the complexity can be controlled for applications having

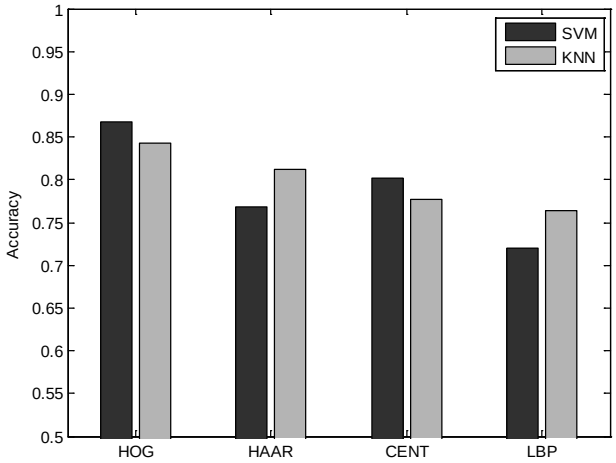


Fig. 12. Accuracy comparison between feature extractors (HOG, HAAR, CENT, LBP) and classifiers (SVM, KNN) in DaimlerChrysler dataset. It depicts that feature extractors are more significant than classifiers.

confined BN structures[6] like GMBs. One of the most effective scoring functions is Bayesian information criterion (BIC) score [59], which can reflect not only the fitness of the structure but also the complexity of the structure.

In order to decrease the complexity of building an optimal (or near-optimal) BN structure for the GMB, we consider accuracy of FC nodes first, followed by FEs and then classifiers instead of considering them at the same time.[7] Between FEs and classifiers, we consider FEs first since they have greater influence on the accuracy than classifiers as shown in Fig. 12.

[6] The BN for a GMB consists of distinct components, the types of which are confined to hypothesis (class), FE-classifier pairs, and LMBs. Thus it is easy to construct a set of candidate BNs and select the one with the highest score.

[7] The reason why we consider FEs and classifiers is that nodes having the same FE or the same classifier tend to have higher dependency with each other compared with nodes having different FEs and classifiers.

STRUCTURING-OF-BAYESIAN-NETWORK-FOR-GMB

1: Connect class node h_l and every basic FE-classifier node $f_i c_j$ (FC node) with an edge.

2: For each FC node, connect it with edges to D_{LMB}s that use the node.

3: *SortedListOfFcNodes* ← sort all the FC nodes by accuracy in descending order

4: For each $Node_i$ in SortedListOfFcNodes for i from 1 to |F||C|-1

5: For each $Node_j$ in *SortedListOfFcNodes* for *j* from *i*+1 to |*F*||*C*|

If $Node_j$ shares an FE with $Node_i$

6: Calculate BIC score with an edge from $Node_i$ to $Node_j$ if the edge does not form a cycle

7: Calculate BIC score with an edge from $Node_j$ to $Node_i$ if the edge does not form a cycle

8: Calculate BIC score without edge insertion

9: Select the best one among the above three cases

10: For each $Node_j$ in *SortedListOfFcNodes* for *j* from *i*+1 to |*F*||*C*|

If $Node_j$ uses the same classifier type as $Node_i$

11: Repeat the lines from 6 to 9

12: For each $Node_j$ in *SortedListOfFcNodes* for *j* from *i*+1 to |*F*||*C*|

If $Node_j$ does not satisfy the condition in 5 or 10

13: Repeat the lines from 6 to 9

Fig. 13. Learning algorithm of Bayesian network for GMB.

The algorithm for building the BN is illustrated in Fig. 13. After the solid edges for the initial skeleton network are built in line 1 and 2, the dotted edges are constructed in lines 3 to 13. All the FC nodes are sorted by accuracy in descending order in line 3, and then picked up in order in line 4 ($Node_i$). Within the nested loops

in line 5, nodes ($Node_j$) sharing the same FE with $Node_i$ are selected and edges are added to connect $Node_i$ and $Node_j$. Then, in line 10, nodes using the same classifier type are selected. Finally, in line 12, the remaining nodes are selected. For example, if HOG-SVM is examined in line 4, HOG-KNN sharing the same FE (HOG) and HAAR-SVM using the same classifier type (SVM) are selected in line 5 and 10, respectively, whereas HAAR-KNN that has nothing in common is selected last in line 12. The nodes are processed in descending order of accuracy, since more accurate nodes should have higher chance to participate in building edges. In short, investigation of relations between FC nodes is performed by accuracy first, then FE sharing, and then classifier types. Each FC node, $Node_i$, in line 4 is connected to another FC node, $Node_j$, located below $Node_i$ in *SortedListOfFcNodes* and evaluated. The BIC score-based method is used for the evaluation. From line 6 to 9, the algorithm selects the best one among forward, backward, and null edges (note that the forward or backward edge should not generate any cycle, because a BN does not allow a cycle in it). Calculating BIC scores with the forward, backward, and null edges in lines from 6 to 9 is repeated in line 11 and 13 in the same manner. The complexity of examining a cycle in a BN in line 6, 7, 11, and 13 was reported to be $O(|V|)$ with incremental method, where $|V|$ is the number of vertexes (nodes) [60]. Because the number of nodes in our case is $|F||C|$ and the pairwise comparison has complexity of $O(\frac{(|F||C|-1)\cdot|F||C|}{2})$, the complexity of building a BN is bounded by $O(|F|^3|C|^3)$.[8]

[8] A recent study on incrementally examining a cycle in a directed acyclic graph (DAG) suggested more efficient algorithm, which is bounded by $O(\sqrt{E})$ for sparse graphs and $O(V^{2.5}/E)$ for dense graphs [61], which means that the complexity of our proposed method in building a BN can be lower than $O(|F|^3|C|^3)$.

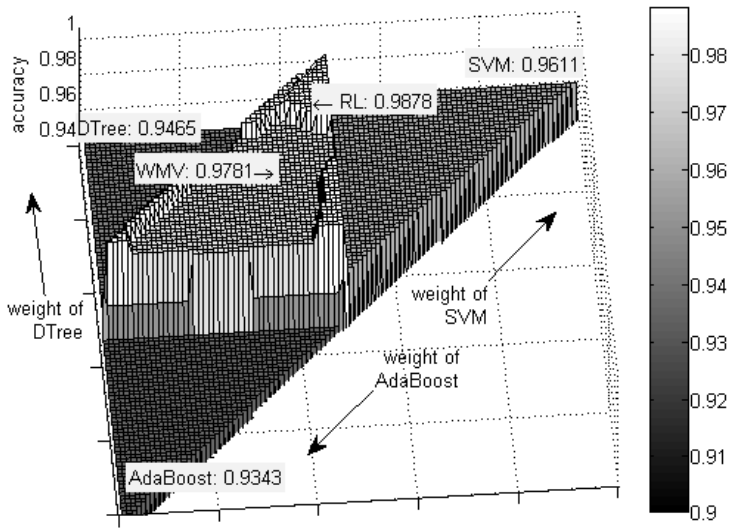


Fig. 14. Accuracy of individual classifiers (SVM, AdaBoost, and decision tree), ensemble using optimal WMV weight not considering dependency, and RL weight for Banknote Authentication in UCI dataset. For the illustration, the plane wS+wA+wD=1 (the sum of weights should be one) is projected into xy-plane, where wS, wA and wD are the weights of SVM, AdaBoost, and decision tree, respectivly; z-axis represents the accuracy.

2.4 Experimental results

We took pedestrian detection and recognition of handwritten numerals in the UCI dataset [51] as the applications used for evaluating the proposed schemes (both GMB and LMB). We compared them with existing ensemble schemes proposed in [14], [40], [33] and AdaBoost as well as basic FE-classifier pairs. As the performance metric for the comparison, we used receiver operating characteristics (ROC) curves for two class classification and used accuracy and error rate for multiple classes classification problems.

TABLE II

COMPARISON OF ACCURACY OF THREE CLASSIFIERS, WMV, AND RL ENSEMBLE FOR UCI DATASET *

Dataset	SVM	AdaBoost	Decision tree	WMV Ensemble	RL Ensemble
Banknote Authentication	0.9611	0.9343	0.9465	0.9781	0.9878
QSAR Biodegradation	0.5365	0.7333	0.6540	0.7333	0.7778
Breast Cancer Wisconsin	0.9234	0.9330	0.9474	0.9522	0.9657
Musk (Version 2)	0.9889	0.9384	0.8236	0.9949	0.9949
Madelon	0.5643	0.5242	0.7579	0.7579	0.7579
Gamma Telescope	0.7166	0.8266	0.7262	0.7443 [a]	0.8266
Gisette	0.9583	0.9322	0.8794	0.9472 [b]	0.9711

[*] The accuracy is measured in evaluation dataset.

[a,b] The accuracy of WMV ensemble is even lower than that of a single component classifier in some cases.

2.4.1 Measure of effectiveness for WMV and RL

In order to see the effectiveness of using RL for deciding weights of classifiers, we compared the WMV using RL with the original WMV. Hereafter, we call 'the

Fig. 15. Examples of DaimlerChrysler Pedestrian dataset. The upper ones are pedestrian samples while the lower ones are non-pedestrian samples.

WMV using RL' simply 'RL' and call 'the original WMV' simply 'WMV' for convenience. WMV does not consider dependency among classifiers when deciding weights as mentioned in Section 2.3.4, while RL explores the space of weights considering the dependency in a trial-and-error manner. TABLE II shows the accuracy measured for seven datasets from UCI Machine Learning Repository [51] for WMV and RL using five-fold cross validation, where the accuracy is measured in evaluation dataset (rows).[9] The table also shows the accuracy of three individual classifiers including SVM, AdaBoost, and decision tree.

In case of Banknote authentication, QSAR biodegradation, Breast cancer and Wisconsin dataset, RL is superior to WMV. An interesting result is shown in Gamma telescope and Gisette dataset; the accuracy of WMV ensemble is lower than even that of a single component classifier such as AdaBoost and SVM. In Madelon and Gamma telescope, some single classifiers show the same accuracy as the ensemble schemes, which means that ensemble of classifiers is not effective in the

[9] Because an overfitted classifier is given a higher priority in training dataset, investigating the accuracy of each classifier with training data cannot precisely estimate the performance of the classifier. Thus, all the accuracy values were measured in evaluation dataset.

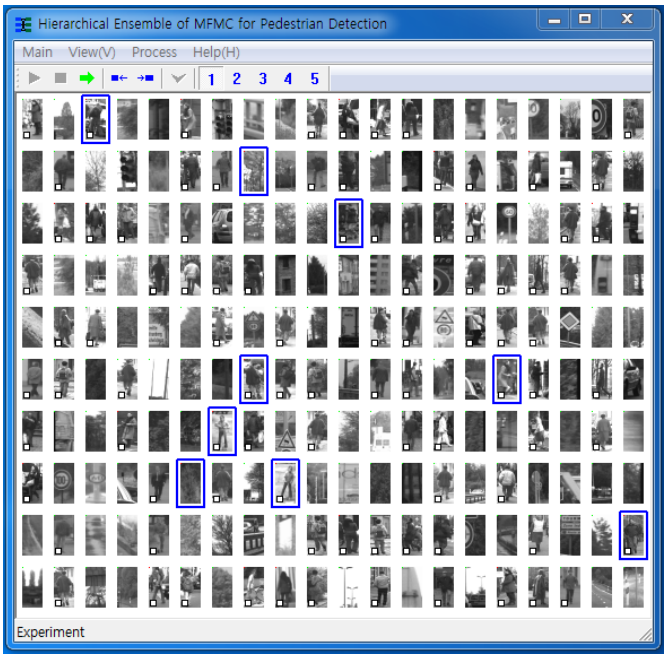
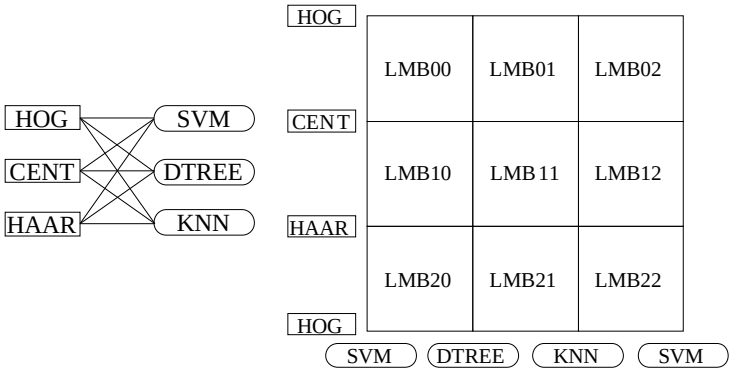


Fig. 16. Screenshot of an experimental application of hierarchical ensemble of MFMC to pedestrian detection using DaimlerChrysler dataset, which is developed based on OpenCV with the C++ language.

datasets. RL ensemble always outperforms original WMV ensemble and single classifiers in all cases. Fig. 14 illustrates the resulting weights space of SVM, AdaBoost, decision tree, and their ensembles (WMV and RL) for Banknote authentication dataset, which illustrates accuracy values for varying combinations of classifier weights. RL shows the best ensemble accuracy of 0.9878, which is higher than that of WMV (0.9781), SVM (0.9611), AdaBoost (0.9343), and decision tree 0.9465).

2.4.2 Pedestrian detection dataset

The experiments use DaimlerChrysler Pedestrian dataset [62], which is composed of 49,000 18x36-resolution images; 29,400 images are used for training and 19,600



(a) (b)

Fig. 17. Experimental system in the paper. (a) The system consists of full connections between feature extractors and classifiers: HOG, CENT, HAAR as feature extractors and SVM, DTREE, KNN as classifiers. (b) GMB is comprised of nine LMBs, each of which has 2x2 combinations.

images are used for testing. The training set contains 14,400 samples for pedestrians and 15,000 samples for non-pedestrians, while the test set has 9,600 and 10,000 samples respectively. For evaluation of the classifiers, a bagging method is adopted; after 80% of the samples in the training set have been used for training, the classifiers are evaluated with the entire training set, and then the results are examined with the test set. Fig. 15 depicts examples of DaimlerChrysler Pedestrian dataset.

The objective of this work is to show that how the proposed hierarchical ensemble scheme can improve the performance compared with baseline ensemble systems. Therefore, we choose well-known FEs and classifiers. As shown in Fig. 17, HOG [28], CENTRIST [47], HAAR [48] are used as FEs and SVM, KNN, DTREE [49] are used as classifiers which have been popular for pedestrian detection systems. The three feature extractors are complementary to each other as follows. HOG uses histograms to count occurrences of gradient orientation for local positions of an image, where local information is well represented; it has been used in many human detection applications [63] [64] [65]. Because CENTRIST focuses on contours such as human body outline, human detection researches have utilized

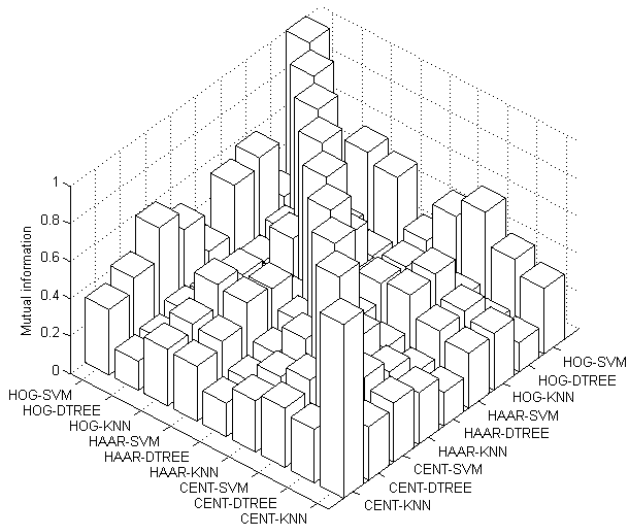


Fig. 18. Mutual information for all pairs of nine classifiers. Dependency between classifiers actually exists.

it [66] [67] [68]. In case of HAAR, rectangular windows detect the pixel intensities of corresponding image regions; it concentrates on block information; many human detection studies have used HAAR based on this reasoning [69] [70] [71]. Meanwhile, GMB is divided into nine LMBs, each of which is created by combining two FEs and two classifiers. For instance, LMB00 is comprised of HOG and CENTRIST as features, and SVM and DTREE as classifiers, resulting in four combinations: <HOG, SVM>, <HOG, DTREE>, <CENTRIST, SVM>, and <CENTRIST, DTREE>. HOG was implemented with 2x2 blocks, each block containing 3x3 pixels, and block stride of 3 pixels. In case of HAAR, we used two wavelets, one for 4x4 pixels and the other for 8x8 pixels, as was done by [62]. There is no such specific parameter used for the case of CENTRIST, which is one of the characteristics of CENTRIST. Since the size of feature vectors for HOG, HAAR and CENTRIST are quite large (972, 7038 and 656, respectively), we used a linear kernel in the case of SVM to increase simulation speed.

The entire system was implemented in the C++ language, based on OpenCV [72]

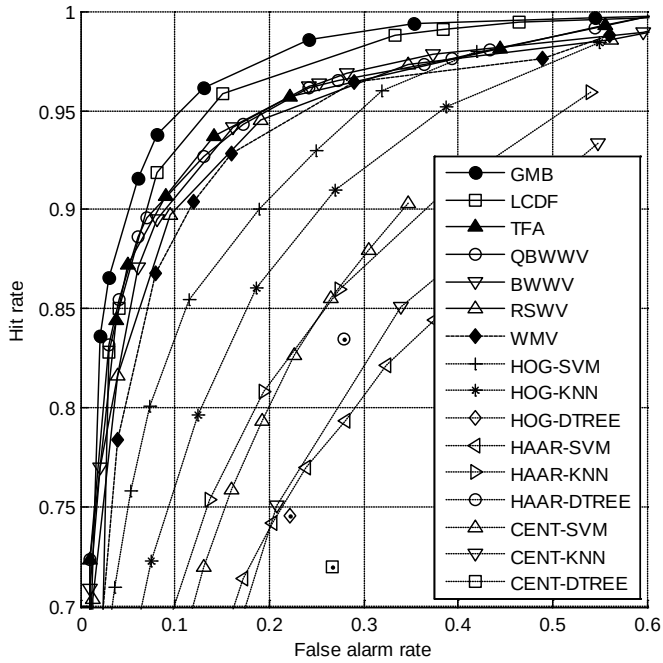


Fig. 19. Performance comparison between the proposed framework and the other schemes.

vision library. Fig. 16 shows a screenshot of experimental examples in which thick bounding boxes depict missing detections. Small boxes in the left bottom corner represent pedestrian cases, while non-pedestrian cases have no such marking.

Mutual information (MI) $I(x_i, x_j)$ given by $\sum_{x_i,x_j} P(x_i, x_j)\, log(\frac{P(x_i,x_j)}{P(x_i)P(x_j)})$ can be used to measure a mutual dependence of two random variables, where x_i and x_j are the variables. $p(x_i, x_j)$ is a joint probability and $p(x_i)$ and $p(x_j)$ are marginal probabilities of x_i and x_j, respectively. The value of mutual information is zero if two variables are independent. Fig. 18 shows that the values of mutual information for all classifier instances; it depicts that no classifier instance is independent of others. It justifies the superiority of the proposed RL compared to the original WMV.

As shown in Fig. 19, all the ensemble methods including GMB, LCDF [40], TFA

TABLE III

RESULTS OF ENSEMBLE METHODS FOR PEDESTRIAN DETECTION

Ensemble Method	Hit rate	
	FAR **0.1**	FAR **0.2**
GMB	0.947	0.977
LCDF	0.928	0.966
TFA	0.912	0.951
QBWWV	0.911	0.948
BWWV	0.906	0.951
RSWV	0.899	0.946
WMV	0.886	0.939

[14], QBWWV [33], BWWV, RSWV, and WMV outperform all the pairs of a single FE and a single classifier such as HOG-SVM, HOG-KNN, HAAR-SVM, etc. In the case of TFA, the weight values of classifiers are generated from the training matrixes obtained by likelihood probabilities.

The figure shows that GMB using BN is superior to all other methods. The hit rate (HR)[10] of GMB achieves 0.947 at 0.1 false alarm rate (FAR)[11] while those of LCDF, TFA, QBWWV, BWWV, RSWV, and WMV are respectively 0.928, 0.912, 0.911, 0.906, 0.899, and 0.886. Table II summarizes the performance results. The hit rate of GMB is 0.947 at 0.1 FAR whereas the hit rate of WMV is 0.886. Considering that the hit rate is close to the maximum hit rate of 1.000, the improvement in miss rate

[10] Hit rate is the ratio of the number of correct detections of pedestrians (i.e., number of true positives) over the total number of pedestrians (i.e., number of true positives + number of false negatives).

[11] False alarm rate is the ratio of the number of incorrect decisions for non-pedestrians (i.e., number of false positives) over the total number of non-pedestrians (i.e., number of false positives + number of true negatives).

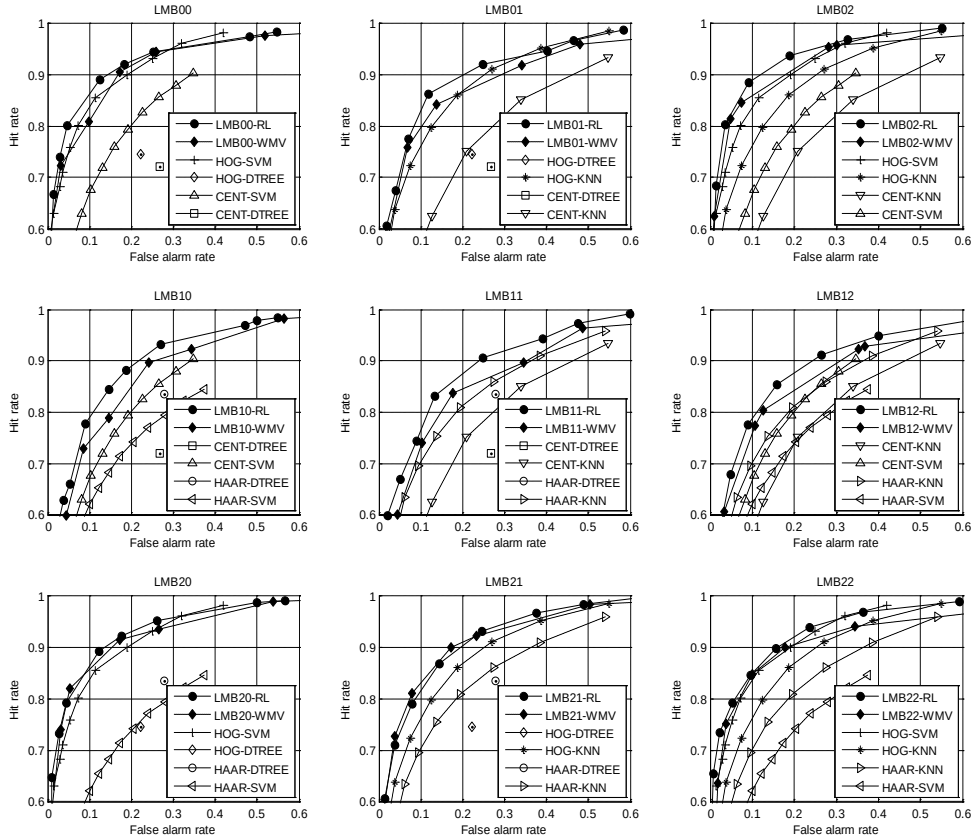
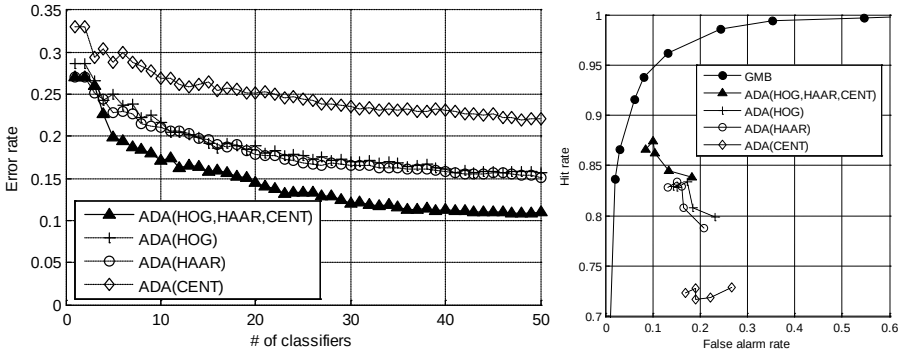


Fig. 20. ROC curves of all the LMBs. Construction of the LMBs is identical to that in Fig. 17.

(false negative) by about 50% (from 0.114 down to 0.053) is quite significant. Meanwhile, the main drawback of EM approach used in LCDF is that the accuracy depends on the initialization and usually converges to some local maximum of the likelihood. In the experiment, we tried many times with different initial values and took the best result.

Fig. 20 depicts ROC curves of each LMB in the GMB with 3x3 LMBs. Every LMB gives higher accuracy compared to any simple FE-classifier pair, which means that an LMB can work as a classifier stronger than basic classifiers. Each chart in the figure compares an LMB using the proposed reinforcement learning



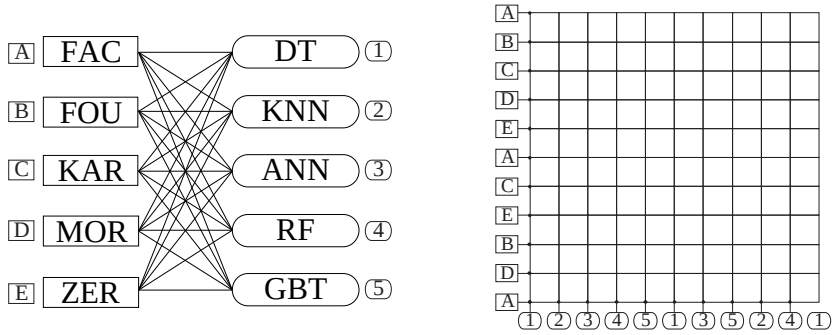
(a) **(b)**

Fig. 21. Performance comparison between GMB and AdaBoost. (a) Error rate of AdaBoost according to the number of classifiers; the parentheses represent the used feature set(s). (b) ROC curve, where the points of AdaBoost depict 10, 20, 30, 40, and 50 classifiers, respectively.

(RL) and another one using the weighted majority vote (WMV) and shows that RL outperforms WMV in most cases, which is due to the fact that WMV ignores all dependencies of MFMC. In case of LMB21, however, WMV shows slightly better HR (e.g., higher by 0.015 at 0.1 FAR). One possible reason is that the implementation of RL is not perfect since the states are based on quantized weights.

2.4.3 Comparison between GMB and AdaBoost

For an experiment with AdaBoost, we selected four cases: three cases using a single feature set of HOG, HAAR, and CENTRIST, respectively and the one using all the feature sets simultaneously. We prepared 50 weak classifiers for each case. Fig. 21 depicts the performance comparison between AdaBoost and GMB; Fig. 21 (a) shows error rate of AdaBoost according to the number of classifiers, where error rate decreases as the number of participating classifiers increases. However, the enhancement saturates when 50 classifiers join into the algorithm. AdaBoost using all the feature sets, named as ADA(HOG, HAAR, CENT) in the graph, achieves the



(a) **(b)**

Fig. 22. GMB composition for UCI Multiple Feature dataset, where five FEs and five classifiers are used. (a) MFMC combination between FEs and classifiers. (b) GMB consists of 10x10 LMBs.

best performance. On the other hand, when compared to GMB, it is inferior as shown in Fig. 21 (b), where the numbers of weak classifiers in AdaBoost are 10, 20, 30, 40, and 50, respectively.

2.4.4 UCI Multiple Features dataset

In order to investigate multiclass cases, we experimented with UCI Multiple Features dataset [51] which consists of features for handwritten numerals from zero to nine (i.e., ten classes) with 2000 patterns per feature set; the resolution of source image is 15x16 pixels. We used five feature sets generated from character shapes as follows (numbers in parentheses indicate the size of each feature vector):

- FAC: profile correlations (216)
- FOU: Fourier coefficients (76)
- KAR: Karhunen-Love coefficients (64)
- MOR: morphological features (6)

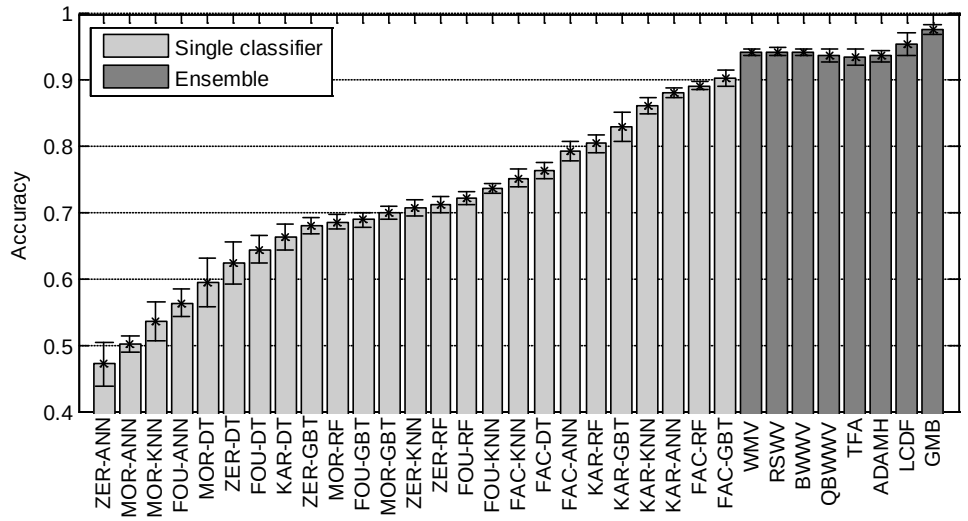


Fig. 23. Experimental result for UCI Multiple Features dataset for single classifiers and ensemble methods.

- ZER: Zernike moments (47)

For multiclass classification, we utilized five classifiers including decision tree (DT), k-nearest neighbors (KNN), artificial neural networks (ANN), random forest (RF), and gradient boost tree (GBT). Thus, there are 25 pairs between FEs and classifiers as illustrated in Fig. 22 (a); the entire system for GMB has 10x10 LMBs as shown in Fig. 22 (b). We measured the average accuracy with the 10-fold cross validation, where two-, three-, and five-folds are used for training, evaluation, and test set, respectively.

The experimental results with Multiple Features dataset are shown in Fig. 23. The first group (light gray) contains the single FE and classifier pairs sorted by its accuracy which is distributed from 0.472 to 0.904. The second group (dark gray) contains the ensembles of classifiers as mentioned in Section 4.2. The difference from the pedestrian detection is the use of AdaBoost.MH [73] (ADAMH) for the multiclass problem instead of the original AdaBoost classifier. Because LCDF based on the EM algorithm is sensitive to initial values, we choose the best one among

TABLE IV

RESULTS OF ENSEMBLE METHODS FOR UCI MULTIPLE FEATURE DATASET

Ensemble Method	Accuracy	
	Average	Standard deviation
GMB	0.977	0.006
LCDF	0.955	0.017
ADAMH	0.936	0.008
TFA	0.936	0.012
QBWWV	0.937	0.009
BWWV	0.943	0.005
RSWV	0.943	0.006
WMV	0.942	0.005

many trials. The classifier ZER-ANN is removed when ensemble is constructed because its accuracy is under 0.5. As a result, all the ensemble schemes outperform all single classifiers. In particular, GMB is more efficient than other ensembles in terms of accuracy as shown in TABLE IV.

2.4.5 LMB selection

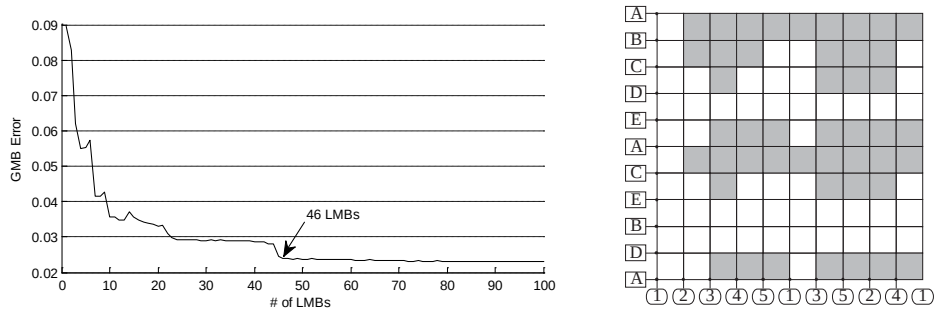
Considering the disadvantage of GMB in complexity, we tried to select automatically a subset of LMBs while maintaining the same level of accuracy as the original version using all the LMBs. The best (optimal) LMB selection can be generated by exhaustive search having complexity of $\binom{|LMB|}{K}$, where $|LMB|$ is the number of LMBs belonging to GMB and K is the number of selected LMBs. In

order to avoid exhaustive search, we adopted $O(K)^{12}$ scheme which selects LMBs in descending order of their individual accuracy. Fig. 24 (a) shows how GMB error changes according to the number of selected LMBs. The first 46 LMBs cover almost all the accuracy (error rate of 0.024) of the original GMB. The 46 LMBs finally selected are displayed in Fig. 24 (b) with gray color in GMB plane. While all the FEs are used in the selected LMBs, FE relations between MOR (D) and ZER(E) are not important in terms of GMB error; <ZER(E) , FOU(B)> and <FOU(B), MOR(D)> do not affect the GMB performance. In other words, although none of the five FEs can be removed in order to cover the original GMB performance, GMB complexity can be reduced by eliminating 54 weak LMBs colored white in Fig. 24 (b). On the other hand, most relations between classifiers are significant to the GMB performance.

2.4.6 Discussion

The hierarchical ensemble scheme proposed in this paper maximally utilizes two ensemble directions when classifiers are gathered into ensemble as illustrated in Section 2.3.3 and Fig. 9. At the same time, it provides a way to efficiently solving the combinatorial problem of mapping multiple FEs to multiple classifiers as mentioned in Section 2.3.1 and 2.3.2. By using this hierarchical ensemble of LMB and GMB, we increase scalability of MFMC with polynomial complexity. LMB is designed to operate in vertical direction based on WMV optimized by using reinforcement learning while GMB is constructed for horizontal direction as Bayesian networks. TABLE V shows which ensemble scheme takes which direction(s) when combining its components. WMV, RSWV, BWWV, QBWWV and AdaBoost, which are based on weight assignments to their components, use the vertical direction, while LCDF and TFA, which are based on generalization and

[12] If considered a sorting problem, the complexity becomes $O(K\log K)$ when using quick sort.



(a) **(b)**

Fig. 24. LMB selection while closely maintaining the accuracy of the original GMB containing all the LMBs. (a) GMB error decreases as the number of LMBs increases, but beyond 46 LMBs the decrease saturates. (b) The selected 46 LMBs with gray color.

regression, utilize the horizontal direction; our scheme takes into account both directions simultaneously. If many classifiers exist in an ensemble but they make same decisions, there may not be any performance improvement over single classifiers. For the same reason, the diversity among classifiers becomes a key to enhance accuracy of the ensemble system. Intuitively, ensemble diversity will increase only under the condition that the participating components (MFMC) use different methods (horizontal and vertical decisions) as shown in Fig. 19 and Fig. 23. In addition, the previous weight-based (vertical) methods such as WMV ignoring dependency between classifiers can be enhanced through RL as shown in TABLE II.

The proposed method using two directions simultaneously inevitably increases complexity of the optimization problem. According to our calculation, however, the asymptotic upper bound of the complexity is $O(|F|^3|C|^3)$ as mentioned in Section 2.3.5. We can further reduce the practical complexity of GMB by using LMB selection as shown in the experimental section.

2.5 Conclusion

TABLE V

COMPARISON OF ENSEMBLE METHODS ACCORDING TO THE DIRECTIONS DISCUSSED IN SECTION 2.3.3

Ensemble Method	Ensemble direction	
	Vertical	Horizontal
GMB	Yes	Yes
LCDF	No	Yes
AdaBoost	Yes	No
TFA	No	Yes
QBWWV	Yes	No
BWWV	Yes	No
RSWV	Yes	No
WMV	Yes	No

This paper presents a hierarchical ensemble framework using multiple feature extractors and multiple classifiers (MFMC). Whereas the previous studies have concentrated on manually finding the best combination of a small number of existing FEs and classifiers, this paper proposes a systematic and automatic optimization framework for arbitrary numbers of FEs and classifiers that can also easily integrate newly developed FEs and classifiers. To manage the complexity of the optimization problem, which increases exponentially as new FEs or classifiers join the combination, this paper proposes to reformulate the problem in a hierarchical manner. In the proposed scheme, the problems are divided into a set of LMBs optimized by reinforcement machine learning, and then final decision is induced on GMB by constructing a Bayesian network. The results show that the hierarchical ensemble framework outperforms the previous approaches such as WMV, RSWV, BWWV, QBWWV, TFA, AdaBoost, and LCDF.

Although the proposed ensemble framework increases the accuracy, utilizing many features and classifiers inevitably requires more computation slowing down the system. As a way of mitigating the problem, we consider hardware supports

such as many-core and/or GP-GPU. In case of critical detection systems such as ADAS, medical service, automatic navigation, and security system, the most important metric is not the hardware cost but the accuracy of the system. Actually, ensemble of classifiers such WMV, AdaBoost were invented by the same motivation. Thus the next step of our research is to design a proper hardware architecture that supports ensemble of various vision processing tasks including Bayesian network manipulations.

3. Synthesis of Efficient Stochastic Logic for Many-Variable Expressions

3.1 Introduction

Stochastic computing (SC) is an alternative to conventional binary arithmetic computing, which computes bit values with probability for applications that tolerate a certain level of inaccuracy. SC uses occurrence probability of 1′s in a (pseudo) random bit stream to represent a number, and allows for an extremely efficient implementation of complex functions (such as multiplication and exponentiation), typically with a few logic gates. For example, a conventional multiplier such as the one shown in Fig. 25 (c) can be replaced by a single AND gate such as the one in Fig. 25 (a). In addition, SC has advantages such as bit-level parall elism and error tolerance. Due to these features of SC, orders of magnitude improvement in terms of area, power, and error resilience has been reported by previous researches applying SC to neural networks [1], image processing [2], electr onic filters [3], and error tolerant systems [4]. Thanks to the equal positional weight of the bits in a bit stream for a value, the precision in SC can be tu ned by varying the number of bits for a value without hardware modification, which is known as progressive precision [5]. Because of massive bit-level pa rallelism from inherent independency between bits, critical path delay of SC l ogic is very small compared to conventional binary arithmetic logic, which m akes a design more efficient in terms of clock frequency and power consump tion. Moreover, SC logic can be implemented in a serial or (partially) paralle l manner as shown in Fig. 25 (a) and (b), where the former can operate mul tiplication on the input bit streams with a single AND gate by scarifying

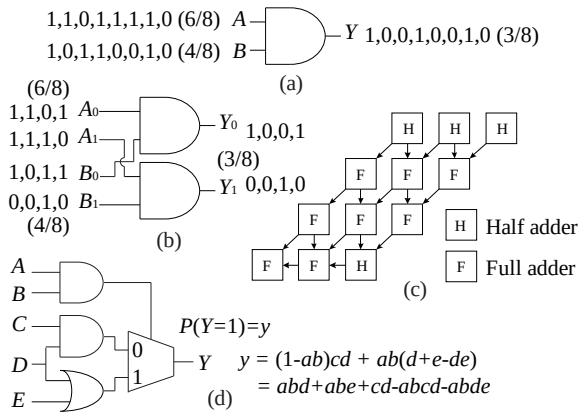


Fig. 25. Example of stochastic logic. (a) Multiplication with a single AND gate. (b) Partially parallel version for multiplication with two AND gates. (c) Three-bit multiplication using half adders and full adders with conventional binary radix encoding. (d) SC logic example with operations representing *y= abd+abe+cd−abcd−abde*, where simple Boolean gates are mapped to compound arithmetic operations.

performance (increased number of clock cycles), while the latter takes a half of the clock cycles with two AND gates (i.e., two parallel units). SC is also tolerant to errors arising from bit-flips, which is recently focused on by emerging technologies such as better-than-worst-case (BTWC) design (considering process variation, aging degradation, and supply voltage clock frequency scaling) [6] and nanometric design (implementing with carbon nanotubes, silicon nanowires, graphene, and molecular electronics, which inevitably generates errors) [7].

As mentioned above, SC requires very small logic area compared to conventional binary logic since it uses very small building blocks for arithmetic operations. One Boolean gate in SC as the one in Fig. 25(a) has the same function as a conventional arithmetic logic block implemented with many gates as shown in Fig. 25(c). However, SC has a critical problem that it requires 2K bits to represent only K bit conventional binary numbers. For example, let us assume two K x K bits multipliers,

one with fully parallel SC logic[13] and the other with conventional binary arithmetic logic like the one in Fig. 25 (c). Let us also assume that the complexity (or area cost) of an AND gate is A while the complexity of a full adder is *F*. Then the complexity is given by

$$C_{SC} = O(A \cdot 2^K) \quad (1)$$

and

$$C_{BIN} = O(F \cdot K^2) \quad (2)$$

for SC logic and binary logic, respectively. Since *A* is much less than *F*, SC logic is smaller than binary logic in case of small *K*. However, the gain decreases fast as *K* (i.e., precision) increases because the complexity of SC logic increases exponentially whereas the complexity of binary logic increases only quadratically. Thus, considering arithmetic operation such as multiplication, SC logic has a benefit when the precision is not high.

What makes SC more effective in implementing arithmetic logic is that only a single SC gate can implement a compound arithmetic function. For instance, an OR gate and a multiplexer (MUX) implement expressions such as $y=a+b-2ab$, $y=(1-c)a+cb$, respectively; a MUX, for example, performs one subtraction, two multiplications, and one addition. Fig. 26 shows traditional logic gates and the corresponding arithmetic expressions implemented by using them as SC logic elements, which we call *SC gates*. Such SC logic can implement a complex expression when the gates are connected together. **Fig. 25** (d) shows that the combination of two ANDs, an OR, and a MUX implements in SC logic a relatively complex arithmetic expression,

---

[13] Fully parallel SC logic takes one clock cycle working with 2^K gates while fully serial SC logic takes 2^K clock cycles with one gate. The two versions of SC logic consumes about the same energy, but the serial version consumes much lower static power. Partially parallel versions are in between the two.

y = abd+abe+cd−abcd−abde, (3)

which, in binary logic, implements a function given by

Y = MUX(AND(A, B), MUX(AND(C, D), OR(D, E))). (4)

As shown above, a complex conventional binary logic implementation of an arithmetic function can be replaced by a very simple SC logic implementation. However, it is difficult to find such an SC logic that efficiently implements a given arithmetic expression, which has prevented usage of SC logic. Moreover, because SC logic has limited capabilities, which are stemming from their probabilistic characteristics, an arbitrary arithmetic expression cannot be simply converted into an equivalent SC logic. In this paper, we present an automatic logic synthesis approach that can find a suitable SC logic from a given arithmetic expression in an application. We try to solve the SC logic synthesis problem inspired by the traditional multi-level logic optimization techniques [8] and their extensions [9, 10]. They are based on exhaustive search among lots of candidates and thus require many sophisticated algorithms to avoid excessive runtime. For the SC logic synthesis, however, our algorithm investigates the structure of a given expression and effectively explores only a limited set of candidates. More specifically, our scheme prunes the useless search space by using common forms of SC logic. Experimental results demonstrate that our technique can generate SC logic circuits that outperform the conventional binary logic circuits in terms of area, critical path delay, and power consumption.

This paper is organized as follows. Section 3.2 gives a brief overview of the related work. Section 3.3 describes the proposed logic synthesis schemes for SC. Section 3.4 shows experimental results, and finally, Section V concludes the paper with some remarks.

3.2 Related Work

There have been abundant previous researches on SC. Unipolar and bipolar

encoding, and some basic stochastic operations were introduced early in the 60′s [11]. Much later, Brown and Card [1] extended the set of stochastic operations introducing important special functions based on state machines. Although stochastic computing had been applied to implementing neural networks even in the 90′s [12], recent emphasis on approximate computing applications rekindled the interest in SC [13]. From the synthesis point of view, [14] and [15] set an important milestone by introducing general methods of designing univariate functions using combinational and sequential circuits, respectively. Another study [16] suggested transform approach for Boolean operation into SC operation in spectral domain. [17] used a regularized polynomial form called binary combination polynomial for synthesizing a given expression. Because those approaches use basis functions to express a polynomial, the SC logic obtained by those schemes is in the form of sum of basis terms like sum of product terms in two-level logic. In particular, as variables are added into an expression, the complexity exponentially grows, because basis functions should represent all combinations of variables.

There have been many researches investigating error estimation and propagation for a given multi-level SC logic circuit [7, 18, 19, 20, 21]. The main concern of those researches is to analyze variance propagation and the effect of correlation between input bit streams in SC logic. [22] proposed a programmable processor that took advantage of stochastic functional units in the aspect of computer architecture. Many researchers worked on practical applications such as image processing [2] [13] [23], low-density parity-check (LDPC) [25] [26] [27] [28] [29], median filter [30], FIR filter [31], Viterbi decoder [32], Turbo-decoder [33], DCT [34], MIMO detector [35], and neural networks [36]. Those researches proposed only a dedicated solution for a specific application.

The methods proposed in [14] [16] [17] can synthesize SC logic circuits for univariate polynomials or polynomials with only a few variables. Although polynomials with a few variables are also an important class of computation, there

are applications that use functions with multiple input signals (i.e., many variables). For example, trilinear interpolation of volume rendering in computer graphics has 11 variables and 27 terms in its sum-of-products form. For this problem instance, the previous approach [17] needs to generate 2,048 ($=2^{11}$) *bases* in the worst case, and moreover, perform stochastic addition of at most 2,048 terms, which not only incurs quite significant an overhead, but also *considerable* loss in the precision of the computation. For such many-variable expressions, the ability to handle multiple input signals at once can lead to a more efficient SC logic, as demonstrated in our experiments. To the best of our knowledge, this is the first work that applies the ideas of multi-level logic synthesis to the SC synthesis problem.

3.3 SC Logic Synthesis for Multivariate Expressions

The heart of our multi-level SC logic synthesis is to find a logic network comprised of basic SC operations corresponding to the SC gates shown in Fig. 26 for a given conventional expressions consisting of basic arithmetic operations. In addition, some special SC logic circuits such as division or integration [1] [11] and other optimized modules (e.g., LDPC [25] and polynomials [14] [16] [17]), which we call SC *macro-circuits*, can be integrated as sub-circuits into the final SC logic circuit for the entire expression during our synthesis process. We do not break a macro-circuit, but use it as a whole as if it were a variable in the expression. A macro circuit can be a sequential circuit, and thus the final SC circuit may include sequential logic as well as combinational logic. In this paper, we consider the problem of synthesizing stochastic logic for an arbitrary multivariate expression. As shown in Fig. 27 (a), from a given algorithm, the design flow generates basic blocks (BBs) and the corresponding data flow graphs (DFGs). Then for each of the BBs (and DFGs), it generates an arithmetic expression in the form of sum-of-products (SOP) possibly including the SC macro-circuits. Then our scheme proposed in this

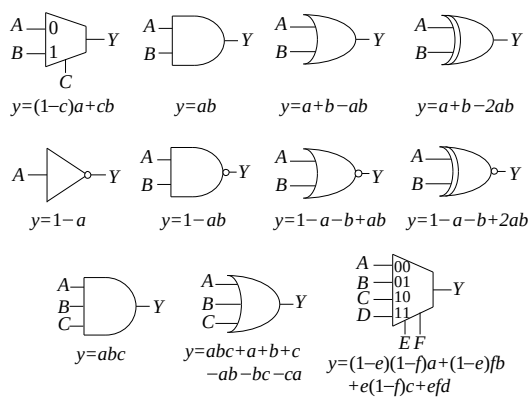


Fig. 26. SC gates and their arithmetic operations in unipolar encoding.

paper starts from the SOP form. [14]

3.3.1 Probabilistic Logic

In unipolar encoding, real number x in the range of [0 1] is represented by a binary random variable X, whose probability of being 1 (sometimes called signal probability) equals p_X. That is, $x = p_X = P(X=1)$, which is obtained by dividing the number of 1s by the total number of bits in the bit stream. Given two independent random variables A and B (representing two real numbers a and b), the output Y of the AND operation on A and B has the signal probability expressed by $p_Y = p_{A \wedge B} = p_A \times p_B = ab$, meaning that a two-input AND gate implements the product of two stochastic numbers in unipolar encoding as shown in Fig. 25(a). In case of two-input OR gate with input A and B or input pair <AB>, because the inputs <01>, <10>, and <11> makes the output 1, the probabilistic value of the output becomes the probability sum of the input pairs, $P(A=0, B=1) + P(A=1, B=0) + P(A=1, B=1) = (1$

[14] Since bipolar encoding representing the range of [-1 1] has the same mechanism to unipolar encoding [0 1], this paper consider unipolar encoding in most cases for simplicity.

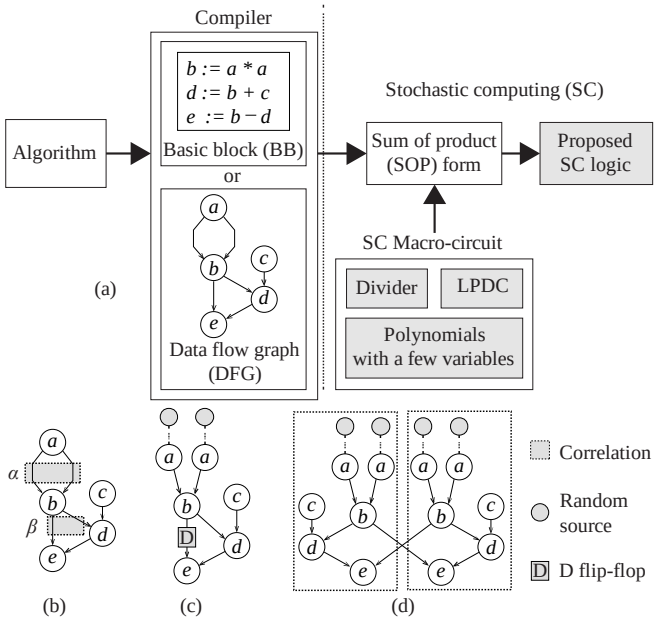


Fig. 27. Overview of the proposed SC logic synthesis with many variables. (a) The proposed scheme begins with a basic block (BB) and its data flow graph (DFG). (b) Example of correlation. (c) Method to solve the correlation problem by using different random sources and a D flip-flop. (d) Swapping the wire can remove the correlation in the parallel version.

$-a)b+a(1-b)+ab = a+b-ab$. Fig. 26 shows the basic SC gates and the functions they implement in unipolar encoding, assuming that the inputs are independent. Note that simple additions such as $a+b$ cannot be implemented by a simple SC logic circuit because they may result in values greater than one; scaled additions such as $0.5(a+b)$ can be accomplished by a MUX, which can implement $(1-c)a+cb$ with c set to 0.5.

Because stochastic logic works in a probabilistic manner, it inherently has non-deterministic errors in contrast to conventional binary radix logic. Regarding *bias* and *variance*, they represent respectively the bias from mean value p for a signal in the stochastic logic and the range of errors from the mean. Because converting a given expression into an SC circuit by using the arithmetic expressions

of Fig. 26 generates expressions equivalent to the original expression, the bias of the stochastic circuit is zero in our approach; for example, (4) represents (3) without bias. In case of variance, some studies suggest the probability models from a simple one [11] (considering the error of Bernoulli sequence no matter how the SC circuit is large) to a complex one [18] (considering propagation of the variance through gates). Another type of error is due to correlation between bit-streams.

Two correlation cases are shown in Fig. 27 (b); in case of α type, the statement $b=a*a$ is implemented by B = AND(A, A) = A, which is not $A*A$. In order to resolve the problem, we can use different random sources A_1 and A_2 (i.e., AND(A_1, A_2) = A_1*A_2) as shown in Fig. 27 (c). In case of β type in (b), because the same signal B branches and reconverges at node e, the input signals of e may be correlated. This type of correlation can be reduced by using D flip-flops as shown in (c) [1] [11] [20] [22]. In the parallel version, the correlation can be reduced by swapping the wires as shown in (d). Meanwhile, methods such as probabilistic transfer matrix (PTM) [24] and SC correlation (SCC) [19] can be used in order to find and measure the correlation [20].

We assume some conditions in this paper as follows,

The input is an expression with a finite number of real variables and real coefficients.

The expression can be converted into an SOP form,

$$Z_{SOP} = \sum_i \prod_j L_{i,j}, \tag{5}$$

where $L_{i,j}$ is a literal or an SC macro-circuit.

The inputs and the outputs are within [0 1] range by using linear scaling such that $(V - V_{min}) / (V_{max} - V_{min})$, where V is a current value and V_{min} and V_{max} are the minimum and maximum values, respectively.

TABLE VI

KERNEL, CO-KERNEL, AND SC KERNEL OF EXPRESSION (6)

Kernel	Co-Kernel	Level	SC Kernel
$1 - a$	bef	0	**Yes**
$1 - e$	$acdf, abf$	0	**Yes**
$b - cd$	aef	0	No
$1 - a - e + ae = 1 - a - e(1 - a)$	bf	1	**Yes**
$be + cd - b - cde = cd - cde - b(1 - e)$	af	1	No
$ab - b - acd = - b + a(b - cd)$	ef	1	No
$- ab - be + abe + acd - acde$	f	2	No
$abef + acdf + b - abf - acdef - bef$	1	3	No

Different variables A and B are independent.

Two variables A_0 and A_1 representing the same input variable A can be made independent by using different random sources such as Fig. 27 (c).

Note that Z_{SOP} is an expression possibly containing an SC macro-circuit as a variable; we focus on SC logic synthesis for a multivariate expression having many variables, which is different from [14] [16] [17].

3.3.2 Definitions

The following terms are from multi-level logic minimization [8]. A *literal* is either a variable or a constant. A *cube* is a product of literals. An expression is called a *kernel* if, when written in SOP form, there is no literal (other than 1) dividing all the cubes simultaneously, aka *cube-free*. A kernel of a given expression can be obtained by dividing the given expression by some cube, in which case the cube is called a *co-kernel*. Kernels have levels; a level-0 kernel has no inner kernels, and a level-n kernel contains one or more level-(n－1) kernels. For example, in the following expression,

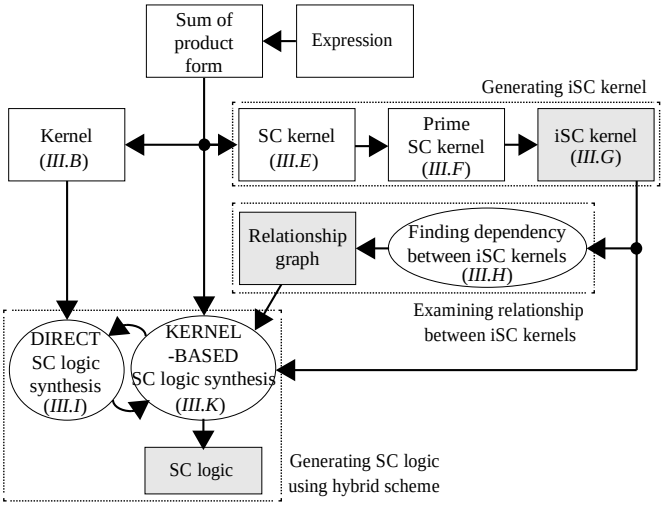


Fig. 28. Overall process for the proposed algorithm.

$$x = abef + acdf + b - abf - acdef - bef$$

$$= b + f(acd - ab + e(-acd - b(1 - a))) \quad (6)$$

a, *b*, *c*, *d*, *e*, *f*, and 1 are literals; *abef* and *acdf* are cubes; the first line of an expression *x* is an SOP form while the second line is a factored form. The SOP form has six variables and 20 literals while the factored form has 14 literals. Table VI lists kernels, co-kernels, and levels of expression (6).

We define *SC logic network* as a set of SC gates with a set of interconnections among them. We say that a conventional expression can be implemented by an SC logic network, if it is possible to convert the expression into an SC logic network by using only the SC gates shown in Fig. 26. When an expression *D* divides another expression *E*, i.e., $E=Q\cdot D+R$, we call *D* a divisor of *E*. We call *D* a *factor* when *D* evenly divides *E*, i.e., there is no remainder ($R=0$). In this paper, we adopt the division method used in [8]; the complexity of the division is only $O(n\log n)$, where *n* is the number of product terms. More terms are defined in the following sections

as necessary.

3.3.3 Overview of the Proposed Method

As illustrated in Fig. 2, the overall process for the proposed method consists of three parts: i) generating iSC kernels, i.e., implementable SC kernels (Sections 3.3.5, 3.3.6, and 3.3.7), ii) finding relationship between iSC kernels (Section 3.3.8), and iii) synthesizing SC logic from the original input expression (Section 3.3.9, 3.3.10, and 3.3.11). The basic idea is to decompose the input expression into iSC kernels, each of which can be implemented using the SC gates. Refer to Sections 3.3.5 and 3.3.7 for the definition of SC kernel and iSC kernel.

If some decomposition is derived from the original expression, it is accepted as a solution. There can be many different solutions, and for the exploration, the algorithm tries to divide the given expression by each iSC kernel. Depending on the result of the division, the algorithm constructs a partial solution and calls itself recursively on the quotient and the remainder. Since the quotients and the remainders are simpler than the dividends, the algorithm must terminate after finite iterations. The search space is combinatorial and defined by the number of iSC kernels and the number of ways of combining them. To speed up the search, the algorithm pre-examines the relationships between iSC kernels and exploits them during the search. In the final step, the SC logic network is synthesized for the original expression by using iSC kernels and the relationship graph.

Our scheme supports expressions with degrees higher than two (e.g., x^3 is supported). Note that the dependency problem in high degree terms can be mitigated by using the isolation scheme [20] (see Section 3.3.10).

3.3.4 Direct Synthesis VS. Kernel-based Synthesis

Our first method, called *direct synthesis*, arranges the input expression in an

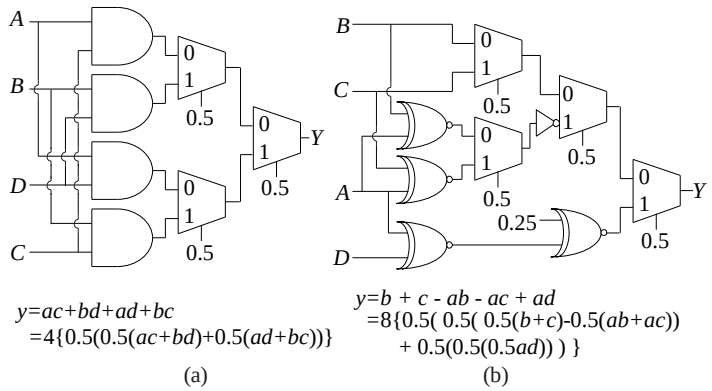


Fig. 29. Examples of direct synthesis. (a) *ac+bd+ad+bc*, where scale factor is 4. (b) *b+c−ab−ac+ad*; the scale factor is 8. It is implemented with bipolar encoding because of scaled subtraction.

arithmetic (not logical) SOP form, which is then mapped to SC logic in a straightforward way. This method can handle any multivariate expression, but may incur some loss in precision. Given Fig. 29, the definition of direct synthesis is implementing each product term using an AND gate (or XNOR in bipolar encoding), and then combining the outputs of the AND gates using a MUX (i.e., scaled addition or scaled subtraction); Fig. 29 shows examples, where $y = ac+bd+ad+bc$ is implemented with $y = 4\{ 0.5(0.5(ac+bd) + 0.5(ad+bc)) \}$ in unipolar encoding while $y = (b+c-ab-ac+ad)$ is implemented with $y=8\{ 0.5(0.5(0.5(b+c)-0.5(ab+ac)) + 0.5(0.5(0.5ad))) \}$ in bipolar encoding.

Direct synthesis can also be performed on a factored form of the same expression, which reduces the number of gates and literals. In this case, the original expression $(ac+bd+ad+bc)$ is algebraically divided by $(a+b)$ or $(c+d)$ to obtain a factored form $(a+b)(c+d)$. [15]

[15] The factored form is implemented by using the conventional multilevel logic optimization technique.

TABLE VII

SC-LOGIC REPRESENTATION FOR TWO-INPUT GATES

Gate	SC Operation	$(1-P)$ Representation	$(1-P)M+P'N$ Format
MUX	$(1-S_0)S_1+S_0S_2$	$(1-S_0)S_1+S_0S_2$	$(1-P)M+PN$
INV	$1-S_0$	$(1-S_0)$	$(1-P)$
NAND	$1-S_0S_1$	$(1-S_0S_1)$	$(1-P)$
OR	$S_0+S_1-S_0S_1$	$(1-S_0)S_1+S_0$	$(1-P)M+P$
NOR	$1-S_0-S_1+S_0S_1$	$-\{(1-S_0)S_1+S_0-1\}$ *	$(1-P)M+P'$
XOR	$S_0+S_1-2S_0S_1$	$(1-2S_0)S_1+S_0$	$(1-P)M+P'$
XNOR	$1-S_0-S_1+2S_0S_1$	$-\{(1-2S_0)S_1+S_0-1\}$	$(1-P)M+P'$
AND	S_0S_1	N/A	N/A

* NOR gate can also be represented by $(1-S_0)(1-S_1)$ or $(1-P)M$ form; the two versions are identical in terms of complexity in SC logic synthesis process.

Such a use of MUX requires scaling as well as a new constant input for the MUX control (i.e., 0.5 in Fig. 29). Thus in order to recover the original function, the output of the MUX needs to be scaled back up (e.g., using coefficient 8 in Fig. 29). However, not only is it complex, slow, and expensive, but also it introduces loss of precision. For an expression with N variables, there can be up to 2^N product terms in the worst case, which implies that the data can be scaled by up to N levels (one level corresponds to 2X scaling) resulting in the corresponding precision loss. This problem can seriously degrade the accuracy even for expressions with a small number of terms, which exists in all similar previous approaches [17].

In this paper, we try to avoid scaled addition as much as possible. Our approach is to search the design space created by the combinations of basic SC gates to find a match while suppressing the use of MUXs as much as possible. The key then is speeding up the search, for which we use techniques from multi-level logic minimization. This method is a very elaborate scheme that explores many possible combinations of basic logic elements to find the most compact SC logic matching

the input expression. To avoid excessive runtime, we consider only possible structures of SC logic that can be generated from the original expression, which allows us to explore only a limited set of candidates. It is inspired by conventional multi-level logic optimization [8, 9, 10]. We call this method *kernel-based synthesis*, since it tries to generate SC logic that is minimized using the concept of kernels. For example, the SC logic in Fig. 25(d) is the result of kernel-based synthesis for the expression in (3), which has a gain compared to that of direct synthesis in terms of area, critical path delay, and accuracy. [16] The problem is how to find an optimal SC logic network from a given expression, which is explained in the following sections.

3.3.5 SC Kernel

In the traditional multi-level logic synthesis, kernels are extracted to select good divisor candidates for algebraic division, since the kernels can divide the original expression. Most schemes used in conventional multi-level logic optimization including the one in [8] try to find a good division result through exhaustive search using divisors obtained from kernels [10]. However, the complexity is prohibitively high, since even a reasonably sized expression can generate many kernels and thus a huge number of divisors. In addition, every subset of cubes in a kernel can be a divisor for the given expression. For example, if an expression generates N kernels and the number of cubes in the i-th kernel is K_i, the number of candidate divisors is $\sum_{i=1}^{N}(2^{K_i}-1)$ in the worst case. Thus, the number of ways of decomposing the given expression becomes $\{\sum_{i=1}^{N}(2^{N_i}-1)\}!$, which is obviously intractable. However, in our SC logic synthesis, we prune a large number of candidates that are not suitable for SC logic networks. Thus, we can find candidates rapidly even with a simple search. This contrasts to the conventional multi-level logic synthesis.

[16] More compact SC logic network is in general more accurate due to limited error propagation.

TABLE VIII

SC KERNEL AND PRIME SC KERNEL OF EXPRESSION T IN EXAMPLE 2

SC Kernel	Prime SC Kernel
$1+bchi+bcm+hm-bc-bchm-hi-m$	No
$1+abc-a-bc$	No
$1+bch-bc-h$	No
$1+hm-hi-m$	**Yes**
$1-a$	**Yes**
$1-bc$	**Yes**
$1-h$	**Yes**

The question to be answered is how to find good divisors to decompose a given expression into sub-expressions that are eventually transformed into SC gates. For this, we define *SC kernel* as an expression that has one constant term no less than 1 and at least one product term with negative coefficient.[17] Simply speaking, it has a $(1-P)$ form, where P is a sub-expression or other than a constant. The reason for considering the $(1-P)$ form is that, due to the probabilistic bound ″1″, most SC gates in Fig. 26 can be represented by a $(1-P)$ form or an expression containing a $(1-P)$ form except for multiplication xy (i.e., AND gate) as shown in Table VII. SC kernels can be obtained by dividing the original SOP by cubes in the same manner as finding kernels in multilevel logic. Identifying SC kernels among the kernels of an expression is straightforward as exemplified in Table VI. An expression without any SC kernel (e.g., dot product expression has no SC kernel in it) is synthesized by the direct synthesis method.

All the SC gate representations shown in Table VII except for multiplication (i.e.,

[17] A constant no less than 1 is allowed. For instance, (2-R) becomes 2(1-0.5R), where (1-0.5R) is a normal (1-P) form, and 2 is a scale factor of the expression.

AND gate) can be expressed as $(1-P)M+P'N$, where P, M, and N are sub-expressions and P' is a modification of P. The modification means that P' has the same expression as P but with a different coefficient and/or constant term subtracted. For example, in case of NOR, P is S_0 while P' is (S_0-1). P' can be recognized within linear time (i.e., $O(n)$, n is the number of terms in a sub-expression); thus, finding P' is very fast. We call the expression $(1-P)M+P'N$ a *common SC form based on* $(1-P)$. For example, a MUX is represented by a common SC form $(1-P)M+PN$; INV and NAND by $(1-P)$; OR by $(1-P)M+P$; NOR, XOR, and XNOR by $(1-P)M+P'$. Thus, if we find a sub-expression transformed to a common SC form, it can be mapped to an SC gate.

Example 1. Given the expression in (3), y can be divided by $(1-ab)$, which is in the form of $(1-P)$. i.e., $y = (1-ab)cd+ab(d+e-de)$. Since this is a MUX form $(1-P)M+PN$, It can be converted into SC logic such as Mux(ab, cd, $d+e-de$). □

In case of three or more inputs, the gates can be hierarchically decomposed into a set of two-input gates; it should contain a $(1-P)$ form for each two-input gate. For example, Fig. 31 shows that three-input OR gate is decomposed into two two-input OR gates, each containg an SC kernel.

3.3.6 Prime SC Kernel

A lower-level (child) SC kernel can be generated by dividing an upper-level (parent) SC kernel by a cube. The lower-level SC kernel can be further divided to generate even lower-level SC kernels. This process is repeated until no more SC

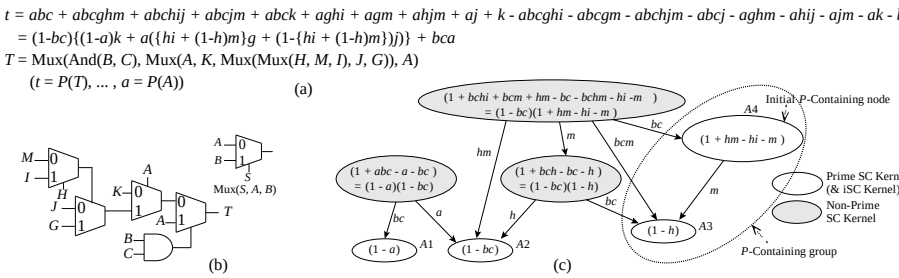


Fig. 30. SC kernels for expression *t* in Example 2. (a) SOP expression for *t* and the synthesis result T. (b) Schematic diagram for T. (c) DAG for SC-kernel.

kernels are generated. In this way, all possible SC kernels can be generated to form a DAG (Direct Acyclic Graph), where each node represents an SC kernel and each directed edge represents a link from a parent SC kernel to its child SC kernel.

If an SC kernel y can be formed by a product of some (or all) of its children x_1, x_2, ..., x_k (i.e., y is fully factored by the children), then the decompositions obtained by using y as a divisor are included in the set of decompositions obtained by using those children, and thus the synthesis process does not need to consider y. If an SC kernel cannot be formed by a simple product of its children, then the SC kernel is called a *prime SC kernel*.

Example 2. Consider the following expression (same as the one in Fig. 30 (a)).

t = abc + abcghm + abchij + abcjm + abck + aghi + agm + ahjm + aj + k − abcghi − abcgm − abchjm − abcj − aghm − ahij − ajm − ak − bck,

where the expression has nine variables, 75 literals, and 19 terms in the SOP form. Table VIII lists all the SC kernels extracted from the expression. The SC kernels form a DAG as shown in Fig. 30 (c). Some SC kernels (represented with gray ovals) are fully factored by its children and such SC kernels are excluded from the DAG (see Lemma 1). Excluding non-prime SC-kernels helps reduce the search space. The white ovals in Fig. 30 (c) represent prime SC kernels, which cannot be fully factored by their children. □

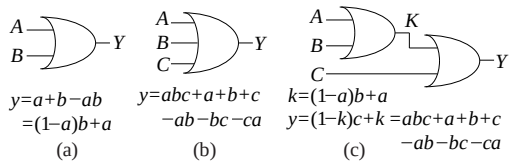


Fig. 31. Decomposition of three-input OR gate. All the gates contain (1-*P*) expression. (a) Two-input OR gate. (b) Three-input OR gate. (c) The decomposed three-input OR gate contains (1-*P*) form such as (1-*a*) and (1-*k*).

Lemma 1. (**Prime SC kernel**) A compound (non-prime) SC kernel, which is fully factored (i.e., no remainder) by prime SC kernels, can be ignored during the SC logic synthesis, since the space searched by using the prime SC kernels covers the space searched by using the compound SC kernel.

Proof. Consider an expression E that can be divided by a compound SC kernel $H=\prod_{i=1}^{n} L_i$, where L_i's are prime SC kernels and n is the number of prime SC kernels. Then

$$E = QH+R, \tag{7}$$

where Q is the quotient and R is the remainder.

Each L_i can also divide E since it is a kernel. Thus

$$E = Q_1L_1+R_1 = (Q_2L_2+R_2)L_1+R_1$$

$$= Q_2L_1L_2+(R_2L_1+R_1)=...$$

$$=Q_n \prod_{i=1}^{n} L_i+\sum_{i=1}^{n} \{R_i \prod_{j=1}^{i} L_{j-1} \}$$

$$= QH+R \tag{8}$$

where L_0 is 1. We see in (8) that various decompositions are possible by using only prime SC kernels (L_i's) and one of them is actually the same as that obtained in (7) by the compound SC kernel. Thus, compound SC kernels can be removed from

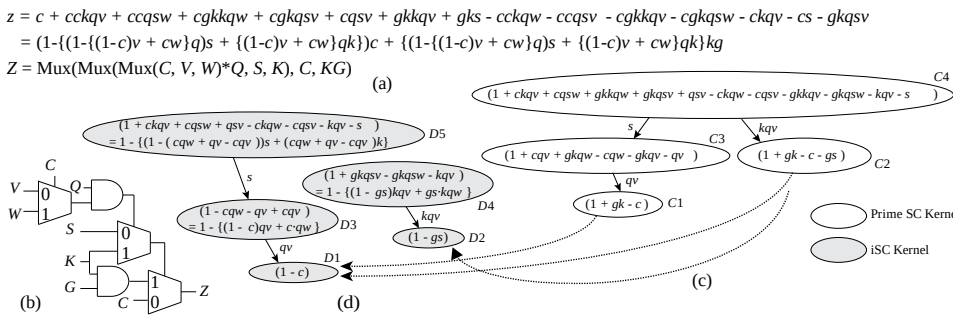


Fig. 32. Finding iSC kernels from prime SC kernels for Example 3. (a) SOP form of expression *z*. (b) SC logic for expression *z*. (c) DAG of prime SC kernels derived by expression *z*. (d) Final DAG of iSC kernels derived from (c).

consideration in the process of design space exploration. □

3.3.7 iSC Kernel

Not all prime SC kernels generated from an expression can be implemented as an SC logic network. For example, $(1-c+gk)$ can be a prime SC kernel of an expression but is not implementable. We define *iSC kernel* as a prime SC kernel that can be implemented with one or more SC gates in Table VII. Thus we find an iSC kernel of the $(1-P)$ form in a given expression to construct a common SC form, $(1-P)M+P\bar{N}$, where each of P, M, and N can also have a common SC form in a recursive manner. We call an iSC kernel $(1\text{-}P)$ a *level-0 iSC kernel*, if P is a single cube. If P has one or more iSC kernels and the highest level of the kernels is λ, then the level of the iSC kernel $(1\text{-}P)$ is $\lambda+1$. For example, $(1-a)$ and $(1-bc)$ are level-0 iSC kernels, while $(1+ac-bc-a) = 1-\{(1-c)a+cb\} = 1-\text{Mux}(c, a, b) = \text{Inv}(\text{Mux}(c, a, b))$ is a level-1 iSC kernel. Note that an iSC kernel generated from a leaf SC

kernel is always level-0 since a leaf node has no child.

The idea is to generate iSC kernels from prime SC kernels. We do this hierarchically starting from the leaf nodes in the SC kernel DAG. First we accept a leaf node as a level-0 iSC kernel if and only if it takes (1 – *single cube*) form. If a leaf node is not in this form, then it cannot be a level-0 iSC node as shown in the following lemma.

Lemma 2. (**Level-0 iSC kernel**) A leaf SC kernel is an iSC kernel, if and only if it takes the (1 −*single cube*) form.

Proof. The 'if' part is trivial; if a leaf SC kernel takes the (1-*single cube*) form, then it can be implemented with an INV or a NAND gate in Table VII and thus it is an iSC kernel. To show the 'only if' part, let's assume that a leaf SC kernel is a level-0 iSC kernel but does not take the (1 −*single cube*) form. That is, the leaf SC kernel is assumed to have (1 −*sum of multiple cubes*) form. Note that the *sum of multiple cubes* does not have another (1-*P*) form since it is from a leaf SC kernel (no child), and thus it cannot be implemented with the gates in Table VII. Therefore, the leaf SC kernel cannot be an iSC kernel, which is a conflict □

For example, in Fig. 30(c), $A1=(1-a)$, $A2=(1-bc)$, and $A3=(1-h)$ are all level-0 iSC kernels. If an SC kernel is not an iSC kernel, we can make it an iSC kernel by dropping some terms.

Example 3. Consider the following expression.

z =c + cckqv + ccqsw + cgkkqw + cgkqsv + cqsv + gkkqv + gks − cckqw − ccqsv − cgkkqv − cgkqsw − ckqv − cs − gkqsv,
where the expression has seven variables, 68 literals, and 15 terms in SOP form.

None of the prime SC kernels of the expression *z* are iSC kernels as shown in Fig. 32 (c). We first consider only level-0 prime SC kernels including *C*1 and *C*2. Node

$C1=(1+gk-c)$ is not an iSC kernel since $P=(c-gk)$ is not a single cube. If we remove gk, it becomes implementable like node $D1=(1-c)$. Since it still divides the original expression[18], it is a level-0 iSC kernel of the expression. In case of node $C2=(1+gk-c-gs)$, we obtain node $D1=(1-c)$ and node $D2=(1-gs)$ by removing terms gk and gs and terms gk and c, respectively.

Traversing toward the predecessor nodes, we identify higher level iSC kernels by dropping terms that cannot be included in the common SC form. For example, by dividing node $C3$ with level-0 iSC kernel $D1$, we obtain $C3=1-\{(1-c)qv + cqw + gkqv - gkqw\}$. If the terms $gkqv$ and $gkqw$ are removed[19], node $D3$ is obtained as an iSC kernel because $(1-c)qv + c \cdot qw$ is a common SC form. Similarly, level-2 node $C4$ is divided by $D3$ resulting in $1 - \{(1 - (cqw + qv - cqv))s + (cqw + qv - cqv)k + gkkqv + gkqsw - gkkqw - gkqsv\}$, and the terms $gkkqv$, $gkqsw$, $gkkqw$, and $gkqsv$ can be removed to obtain kernel $D5$. □

3.3.8 Relationship Between iSC Kernels

Once iSC kernels are generated, we can proceed to our synthesis algorithm. However, for effective pruning of the search space, we exploit the relationship between iSC kernels. For example, consider casting the expression t in Fig. 30 into the common SC form based on $A1=(1-a)$ to obtain $t=(1-P)M+P\,N$. It is done by using iSC kernel $A1=(1-a)=(1-P)$ as the divisor. If we know a priori from the relationship between nodes $A1$ and $A3=(1-h)$ that the common SC form based on

[18] Any subset (sub-expression) of a kernel divides the original expression.

[19] The terms to be removed can be selected easily through pattern matching with the $(1-P)M+P\,N$ form.

A3 does not exist in the *M* part of the $(1-P)M+PN$ form based on *A*1 (we call this "*M* of *A*1" for brevity), then we can prune useless trials of casting *t* into the form of $t=(1-a)\{(1-h)B + h\cdot C\}+ a\cdot D$, where *B*, *C*, and *D* are sub-expressions, since such a casting is not possible.

We define five relationships between two iSC kernels as follows, all based on the common SC form, $W=(1-P)M+PN$.

***M*-containing**: If *W* has the following pattern

$$W = (1-P_0)\{(1-P_1)A+B\}+P_0C +F \quad (9.1)$$

$$= (1-P_1)\{(1-P_0)A+P_0D\}+ (1-P_0)B+P_0E+F \quad (9.2)$$

where $A\neq 0$, *B*, and $C=(1-P_1)D+E$ are sub-expressions, and *F* can be either 0 or an arbitrary remaining subexpression. That is, if *M* of $(1-P_0)$ contains $(1-P_1)$, and *C* (i.e., *N* of $(1-P_0)$) is either 0 or a sub-expression containing $(1-P_1)$, then we say that $(1-P_0)$ is *M-containing* $(1-P_1)$.

***MO*-containing**: If *W* has the following pattern

$$W = (1-P_0)\{(1-P_1)A+B\}+P_0C+F \quad (10.1)$$

$$= (1-P_1)\{(1-P_0)A\}+ (1-P_0)B+P_0C+F, \quad (10.2)$$

where $A\neq 0$, *B*, and $C\neq 0$ are sub-expressions, and *F* can be either 0 or an arbitrary remaining subexpression. That is, if only *M* of $(1-P_0)$ contains $(1-P_1)$ but *C* (i.e., *N* of $(1-P_0)$) does not contain $(1-P_1)$, then we say that $(1-P_0)$ is *MO-containing* $(1-P_1)$.

***NO*-containing**: If *W* has the following pattern

TABLE IX

DETERMINING NON-P-CONTAINING RELATION A

Numerator	Q_X	Q_Y	R_X	
Denominator	Y	$1-X$	$1-X$	Y
M-Containing	D	–	D [b]	D [b]
			N [c]	–
MO-Containing	D	–	D	N
NO-Containing	N	D	–	D
Unrelated	N	N	–	D

[a] D: divides with or without remainder, –: don't care,
N: cannot divide (including the case of zero dividend).

[b] $C \neq 0$, [c] $C=0$ in (9.1).

$$W = (1-P_0)\{A\} + P_0\{(1-P_1)B+C\}+F \tag{11.1}$$

$$= (1-P_1)\{P_0B\} + P_0C+(1-P_0)A+F, \tag{11.2}$$

where $A \neq 0$, $B \neq 0$, and C are sub-expressions, and F can be either 0 or an arbitrary remaining subexpression. That is, if only N of $(1-P_0)$ contains $(1-P_1)$, then we say that $(1-P_0)$ is *NO-containing* $(1-P_1)$.

Unrelated: If W has the following pattern

$$W = (1-P_0)A + (1-P_1)B + F, \tag{12}$$

where $A \neq 0$, $B \neq 0$ are sub-expressions, and F can be either 0 or an arbitrary remaining subexpression. Then we say that $(1-P_0)$ and $(1-P_1)$ are *unrelated*.

***P*-containing**: If W has the following pattern

$$W = (1-P_0)A + B, P_0=(1-P_1)C+F, \tag{13}$$

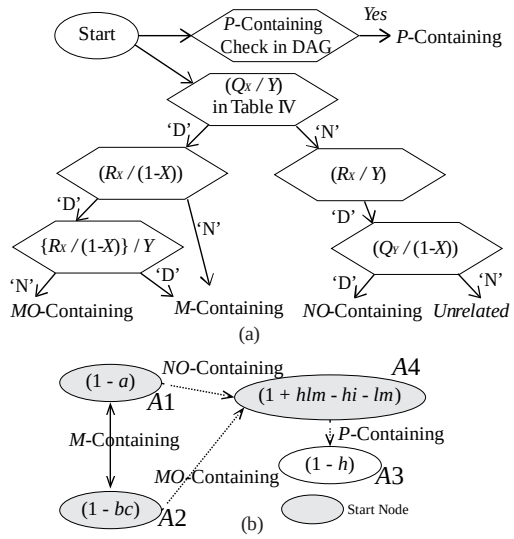


Fig. 33. Relationship between two iSC kernels. (a) The procedure to find the relationships between them. (b) An example of iSC kernel relationship graph for Fig. 30.

where $A \neq 0$, B, $C \neq 0$ are sub-expressions, and F can be either 0 or an arbitrary remaining subexpression. Then we say that $(1-P_0)$ is *P-containing* $(1-P_1)$.

Note in (9.1) and (9.2) that $(1-P_1)$ is also *M-containing* or *MO*-containing $(1-P_0)$. In (10.1) for the *MO*-containing relationship, A and C should not be zero. If C=0, it becomes the *M*-containing relationship. In the same manner, in (11.1) for the *NO*-containing relationship, A and B should not be zero and the relationship is also unidirectional. The unrelated case defined in (12), where A and B should not be zero, is not implementable with the SC gates. In our example of Fig. 30, $A1$ is *NO*-containing $A3$ and $A4$. $A2$ is *MO*-containing $A3$ and $A4$, and $A4$ is *P*-containing $A3$. Fig. 34 shows two examples of cascaded relationships.

To find the relationship between two iSC kernels $X=(1-P_0)$ and $Y=(1-P_1)$, we first check to see if X is a predecessor of Y in the iSC kernel DAG, hence Y can divide X but Y is not a factor of X. If yes, then X is *P*-containing Y (e.g., $A4$ is

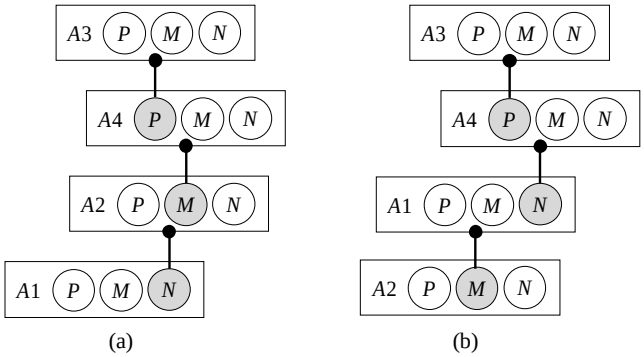


Fig. 34. Examples of relationships between iSC kernels in Fig. 30. *A*3 exists in *P* of *A*4, *M* of *A*2, and *N* of *A*1; *A*4 exists in *M* of *A*2 and *N* of *A*1.

P-containing *A*3 in Fig. 30). If *X* and *Y* are not in a *P*-containing relationship, we divide the original expression *W* by *X* and *Y* to obtain the quotients and remainders as follows:

$$Q_X \equiv W / X, \quad R_X \equiv W \,\%\, X ,$$

$$Q_Y \equiv W / Y, \quad R_Y \equiv W \,\%\, Y. \tag{14}$$

If Q_X is divided by *Y*, then *X* is *M*-containing or *MO*-containing *Y*. In this case, if R_X is not divided by (1−*X*) or if it is divided by both (1−*X*) and *Y*, then *X* is *M*-containing *Y*; otherwise, *X* is *MO*-containing *Y*. Table IX shows how we determine the relationship between *X* and *Y*. Fig. 33 (a) illustrates the process of finding the relationship between iSC kernels.

Let us take the example in Fig. 30 (c). Given $X = (1-a)$, $Y = (1 + hlm - hi - lm)$, and the original expression *t*, we examine $Q_X = t/(1-a)$ as well as Q_Y and R_X according to Fig. 33 (a), to find that (1−*a*) is *NO*-containing $(1 + hlm - hi - lm)$. This process is applied to every pair of iSC kernels and then we can get a *relationship graph* for Fig. 30 (b) as shown in Fig. 33 (b). Note that the relationship

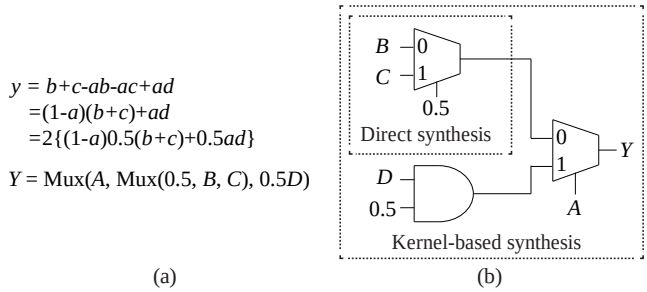


Fig. 35. Example of hybrid scheme combining both kernel-based and direct synthesis. (a) Expression of Fig. 29b and their SC logic. (b) Schematic diagram for the example in unipolar encoding.

between any pair of iSC kernels can be identified by at most four divisions (for the case of *NO*-containing and Unrelated). Actually, R_Y does not need to be calculated at all accoding to Table IX.

3.3.9 Hybrid Scheme

Our kernel-based logic synthesis approach is not applicable to some expressions. A good example is *dot product* such as (*ab*+*cd*), which does not even have any SC kernel. During SC logic synthesis, if a part of an expression is identified as not implementable with the kernel-based synthesis, then *direct synthesis* scheme (see Section 3.3.4) is invoked to generate SC logic for that part of computation. In Fig. 35, for example, the expression of Fig. 29 (b), (*b*+*c*−*ab*−*ac*+*ad*), can be implemented by MUX(*a*, *b*+*c*, *d*), where *a* is the control input to the MUX. However, since (*b*+*c*) is not implemented by the kernel-based synthesis, we use direct synthesis to obtain 0.5(*b*+*c*) and thus we have MUX(*A*, MUX(0.5, *B*, *C*), 0.5*D*) for *Y*, where the precision loss (i.e., scale factor) becomes 2× only (in case of full direct synthesis, the precision loss becomes 8×).

3.3.10 Cost Function

There can be many candidate solutions (SC logic networks) generated during the SC logic synthesis process, and among them we try to select the minimum cost solution x^* given by

$$x^* = \quad argmin_{x \in S} C_{total}(x), \tag{15}$$

where S is the set of candidate solutions. We define the total cost $C_{total}(x)$ of solution x as

$$C_{total}(x) = (1-\alpha)C_{area}(x) + \alpha C_{error}(x), \tag{16}$$

where $C_{area}(x)$ is the area of the SC logic network, $C_{error}(x)$ is the amount of error generated from it, and α is for weight distribution between them. The area can be measured by the number of literals $N_{literal}$ in an SC logic network as mentioned in [8]. For example, the SC logic network in (4) has six literals. Meanwhile, we define two types of error C_{error} and C_{error_iso} as follows:

$$C_{error}(x) = ((1-\beta)Var(x) + \beta Corr(x)) \cdot \{Scale(x)\}^2 \tag{17}$$

$$C_{error\_iso}(x) = Var(x) \cdot \{Scale(x)\}^2, \tag{18}$$

where $Var(x)$ stands for variance of the output of the SC logic network, $Corr(x)$ represents correlation between input streams to SC gates and can be measured using PTM and SCC [19][20][24], and $Scale(x)$ is a scale factor; β is for weight distribution. $C_{error}(x)$ includes these three components while $C_{error\_iso}(x)$ removes the correlation from $C_{error}(x)$.

SC-LOGIC-SYNTHESIS (E)

1: $SF \leftarrow$ set of start nodes in the iSC kernel relationship graph
2: $L_E \leftarrow$ NIL, L_E is a set of solutions for E
3: for each $F \in SF$
4: $L_E + \leftarrow$ KERNEL-BASED-SYNTHESIS (E, F)
5: $R \leftarrow \ argmin_{X \in L_E} \ (C_{total}(X))$
6: return R

Fig. 36. Pseudo-code of the top-level function for the proposed algorithm.

In case of $C_{error\_iso}(x)$, we assume that input streams to each gate are independent of each other (i.e., using different random sources) and thus $Corr(x)$ can be neglected as shown in Fig. 27 (c). Even if there exists some correlation between the streams, the effect of $Corr(x)$ can be significantly reduced by using the isolation scheme mentioned in [20], which allows us to ignore the term. The isolation scheme needs an additional D flip-flop in serial SC logic such as that in **Fig. 25** (a). However, in parallel SC logic such as that in **Fig. 25** (b), the overhead of additional D flip-flop can be avoided by just swapping one input with the other input, as shown in Fig. 27 (d).

Measuring $Var(x)$ has been studied in [7] [11] [18] [21]; we select the approach in [18] because it accurately models the propagation of variance through multiple levels of SC logic. Considering that the variance of Bernoulli sequence is maximized at 0.5, we use 0.5 as input values for the worst case. $Scale(x)$ is generated from direct synthesis as mentioned in Section 3.3.4 and 3.3.9. For instance, $Scale(x)$ is the total scale factor such as ′2′ in Fig. 35. Since the variance of a random variable K scaled by a is a^2 times the variance of K (i.e., $Var(aK) = a^2Var(K)$), we take the square of $Scale(x)$. Eventually, we define the total cost functions as,

$C_{total}(x) = (1-\alpha)N_{literal}(x) +$

KERNEL-BASED-SYNTHESIS (E, F)

1: L_E ←NIL, L_E is a set of solutions for E
2: L_S ←NIL, L_S is a set of solutions for sub-expressions of E
3: P_0 ← 1-F, F is an iSC kernel
4: (Q_0, R_0) ← DIVIDE(E, 1−P_0)
5: (Q_1, R_1) ← DIVIDE(R_0, P_0)
6: L_S ← CONTAINING-SEARCH (P_0, Q_0, Q_1)
7: if (Q_0 does not exist): then E is not an SC logic network
8: L_E ← DIRECT-SYNTHESIS(L_S)
9: else if (both Q_1 and R_1 exist): then E is not an SC logic network
10: L_E ← DIRECT-SYNTHESIS(L_S)
11: else if (Q_0 is 1) and (R_0 does not exist): then E has (1−P) form
12: L_E ← INV-NAND-AND-GATE(L_S)
13: else if (Q_0 exists) and (Q_1 is 1): then E has (1−P)M +P form
14: L_E ← OR-NOR-XOR-XNOR-AND-GATE(L_S)
15: else if (both Q_0 and Q_1 exist): then E has (1−P)M +PN form
16: L_E ← MUX-AND-GATE(L_S)
17: else: this means that E is not an SC logic network
18: L_E ← DIRECT-SYNTHESIS(L_S)
19: return L_E

Fig. 37. Pseudo-code for Kernel-Based-Synthesis function.

$$\alpha\cdot((1-\beta)Var(x) + \beta\, Corr(x)) \cdot \{Scale(x)\}^2, \tag{19}$$

$$C_{total\_iso}(x) = (1-\alpha)\, N_{literal}(x) + \alpha\cdot Var(x)\cdot\{Scale(x)\}^2. \tag{20}$$

3.3.11 SC Synthesis Algorithm

```
CONTAINING-SEARCH (P, M, N )
    // P, M, and N represent common SC form, (1−P)M+P'N.
1:  L_E ←NIL, L_E is a set of solutions for (P, M, N)
2:  (L_P , L_M , L_N)←NIL, a set of solutions for P, M, and N, respectively
3:  NF←set of (1− P) and successors of (1− P) in the iSC kernel
            relationship graph
4:  for each F ∈ NF
5:      if ((1− P) is P-Containing F)
6:          L_P +← KERNEL-BASED-SYNTHESIS (P, F)
7:      if ((1− P) is MO-Containing F) and (M exists)
8:          L_M +← KERNEL-BASED-SYNTHESIS (M, F)
9:      else if ((1− P) is NO-Containing F) and (N exists)
10:         L_N +← KERNEL-BASED-SYNTHESIS (N, F)
11:     else if ((1− P) is M-Containing F) and (M exists)
12:         L_M +← KERNEL-BASED-SYNTHESIS (M, F)
13:         if (N exists)
14:             L_N +← KERNEL-BASED-SYNTHESIS (N, F)
15:     else: this means that F is not an SC logic network
16:         L_E +← DIRECT-SYNTHESIS(P, M, N)
17: L_E +← GET-ALL-COMBINATION(L_P , L_M , L_N)
18: return L_E
```

Fig. 38. Pseudo-code for Containing-Search function.

The overall procedure for synthesizing SC logic is illustrated from Fig. 36 to Fig. 38. Given a multivariate expression *E*, the SC-LOGIC-SYNTHESIS function in Fig. 36 first computes iSC kernels and their relations (Line 1). Then for each start node, it calls the KERNEL-BASED-SYNTHESIS function to search for a solution (Line 4). A *start node* is an iSC kernel that has at least one outgoing edge in the iSC relationship graph (Fig. 33 (b)). The KERNEL-BASED-SYNTHESIS function in Fig. 37 first divides the input expression by the iSC kernel passed as the 2nd

parameter (Line 4) and its 1′s complement (Line 5). Then it calls the CONTAINING-SEARCH function with P_0, Q_0, and Q_1 (Line 6), which represent a common SC form $E = (1-P_0)Q_0+P_0Q_1$. The CONTAINING-SEARCH function returns a set L_S of all combinations of possible implementations of P_0, Q_0, and Q_1. Then from Line 7 to Line 18 in Fig. 37, depending on the result of the divisions, a partial solution is formed for each combination in L_S (Lines 12, 14, and 16) or defaults to direct synthesis (Lines 8, 10, and 18). For example, in Line 13, if the first quotient Q_0 exists and the second quotient Q_1 is 1, the given sub-expression can be decomposed into simpler expressions including *M* and *N* using OR, NOR, XOR and AND gates.

The CONTAINING-SEARCH function of Fig. 38 first gathers the current node and all the successors (iSC kernels) of the current node in the relationship graph (e.g., Fig. 33 (b)) (Line 3). For each of the successor nodes, KERNEL-BASED-SYNTHESIS is recursively called according to their containing types, and the search continues.

For example, Fig. 39 shows the process for SC logic synthesis generating one candidate of the SC logic network for Fig. 30. Given the expression *t*, the SC-LOGIC-SYNTHESIS function calls KERNEL-BASED-SYNTHESIS with iSC kernel $A1=(1-a)$ among the start nodes (Fig. 33 (b)). *S*1 in Fig. 39 (a) illustrates that the expression *t* is divided by $(1-a)$ iSC kernel in Line 16 of KERNEL-BASED-SYNTHESIS, which is matched to a MUX, because the divided expression has the MUX form $(1-P)M+PN$ in Table VII. CONTAINING-SEARCH in *S*2 selects the next iSC kernel $(1-bc)$ which $(1-a)$ is *M*-containing. Thus, both *M*- and *N*-part of the expression are divided by $(1-bc)$ in *S*3 and *S*4, respectively. Because *S*3 has the form of $(1-bc)k$, it can be synthesized as AND(INV(AND(*b*,*c*)), *k*). The form generated in *S*4 is matched to an OR gate because it has the $(1-P)M+P$ form of Table VII. Meanwhile, since $(1-bc)$ is *MO*-containing $(1-hlm-hi-lm)$, we divide the *M*-part by $(1-hlm-hi-lm)$ in *S*6, which is synthesized as a MUX. Finally, utilizing that $(1-hlm-hi-lm)$ is *P*-containing $(1-h)$ in *S*7, we

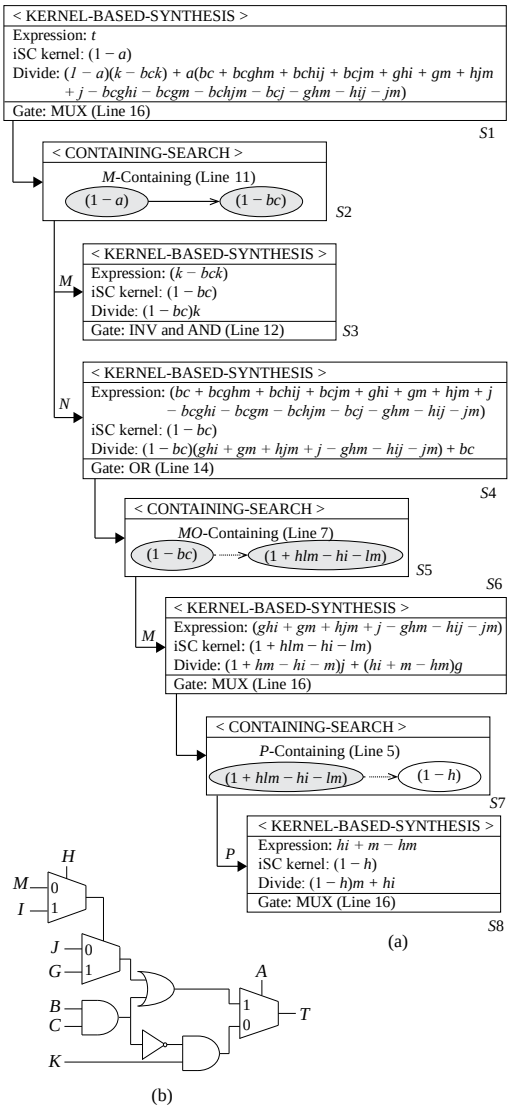


Fig. 39. SC logic synthesis example for Fig. 30. The result is a different candidate compared with Fig. 30b. (a) Synthesis steps according to each iSC kernel. (b) Schematic diagram for the result.

obtain the last result in *S*8. This example shows only one candidate solution.

Note that the result of Fig. 39 (b) is different from the result of Fig. 30 (b)

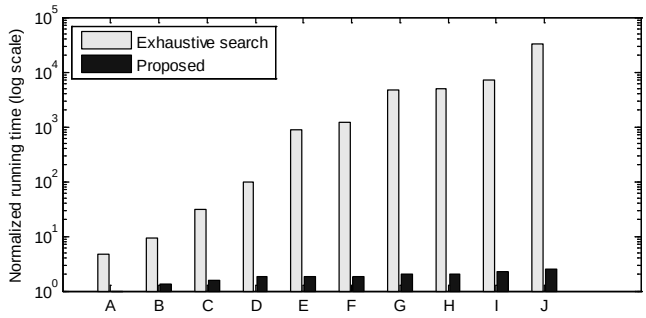


Fig. 40. Comparison of the proposed algorithm with exhaustive search.

TABLE X

INPUT PARAMETER FOR FIG. 40

Parameter	A	B	C	D	E	F	G	H	I	J
# of cubes	5	6	7	8	9	9	9	9	9	9
# of kernels	5	6	8	8	10	11	12	13	15	15
# of cubes in max kernel	5	6	7	8	9	9	9	9	9	9
# of variables	5	5	5	5	7	8	8	9	9	10

although both the input expression *t* and iSC kernels are identical. Different usage of iSC kernels during SC logic synthesis generates different SC logic networks; many candidates are actually generated. Using the cost function stated in Section 3.3.10, we finally select the best one among the candidates.

3.4 Experimental Results

In this section, we present experimental results for the performance of the proposed algorithm itself. We also present experimental results for the quality of the synthesis results including area, critical path delay, power consumption, as well as accuracy.[20]

[20] As mentioned in Section 3.2, because all previsous approaches focus on

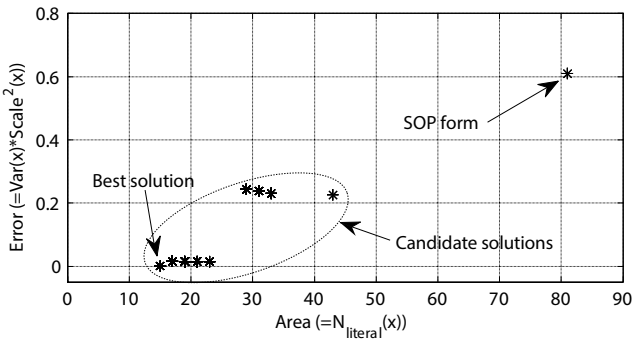


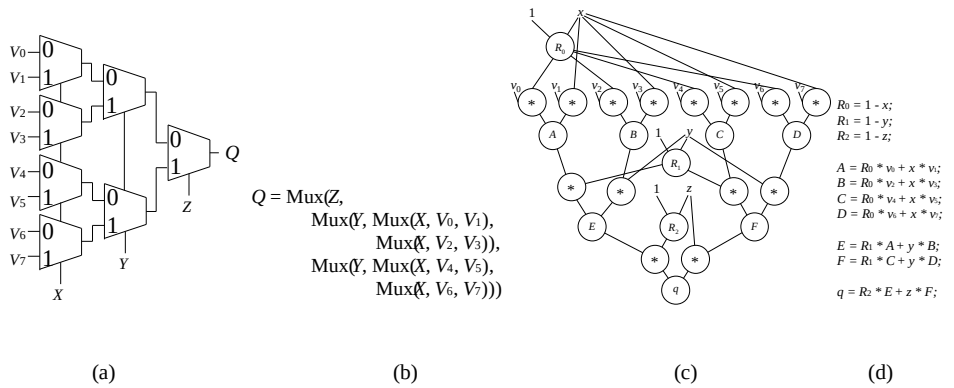
Fig. 41. Area and error values of candidate solutions for TI used to calculate the cost function (19).

3.4.1 Performance of SC Logic Synthesis Algorithm

To evaluate the performance of our algorithm, we use a synthetic input expression. The synthesis algorithm is written in C++ language and compiled with Microsoft visual studio compiler. The host machine is Intel Core i7-2600K operating at 3.40GHz clock frequency. Fig. 40 compares the runtime of the proposed algorithm with that of exhaustive search. We vary the complexity of the synthetic input expression by modifying parameters including the number of cubes, kernels, and variables as shown in Table X. As the parameter values get bigger as shown in Table X, the runtime of the exhaustive search increases exponentially, while our proposed algorithm finds all possible candidates for SC logic networks much faster. The reason is that our approach can efficiently explore the design space by using iSC kernels and their relationship.

---

polynomials including a small number of variables, the efficiency dramatically decreases in case of many variables. Thus, we do not compare them directly.



(a) (b) (c) (d)

Fig. 42. Comparison of TI implementations. (a) SC logic generated by the proposed algorithm. (b) SC logic expression of (a). (c) Data flow of conventional arithmetic operations optimized with CSE. (d) Arithmetic expressions for the nodes in (c).

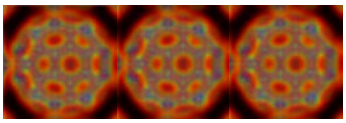
3.4.2 Quality of Synthesis Results

We experiment with four different applications. The first one is trilinear interpolation (TI) in volume rendering [38]. Volume rendering is commonly used in 3D data visualization and TI is a function mainly executed in volume rendering. It is found by using Intel Parallel Studio that the function occupies 72.3% of the entire execution time of volume rendering. It is fully mapped to an SC logic network that can be synthesized with only kernel-based synthesis method. The expression q for TI in SOP form is as follows,

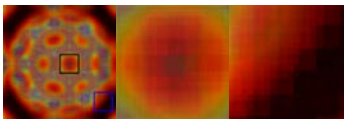
$$q = xyzv_1 + xyzv_2 + xyzv_4 + xyzv_7 + xyv_0 + xyv_3 + xzv_0 + xzv_5 + xv_1 + yzv_0 + yzv_6 + yv_2 + zv_4 + v_0 - xyzv_0 - xyzv_3 - xyzv_5 - xyzv_6 - xyv_1 - xyv_2 - xzv_1 - xzv_4 - xv_0 - yzv_2 - yzv_4 - yv_0 - zv_0, \quad (21)$$

where x, y, and z are fractional values for current coordination and v_0~$_7$ are voxel values; it has 11 variables, 27 terms, and 81 literals.

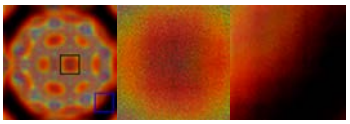
Fig. 41 depicts the cost value $C_{total}(x)$ of each TI candidate x as defined in (19).



(a) Reference image **(b) FIX (8 bit)** **(c) SC (256 bit)**



(d) FIX (5 bit) (e) Expansion of (d) (f) Expansion of (d)



(g) SC (32 bit) (h) Expansion of (g) (i) Expansion of (g)

Fig. 43. Result images for TI in volume rendering.

The horizontal axis represents the area cost modeled by the number of literals, $N_{literal}(x)$, while the vertical axis represents the error cost modeled by $Var(x)\cdot\{Scale(x)\}^2$.

Fig. 42 (a) and (b) present the best implementation in SC logic candidates generated by the proposed algorithm for TI, which uses only seven MUXes. We compare the result with a conventional binary implementation. In order to optimize the binary logic, we take the common sub-expression elimination (CSE) technique used in compilation and high level synthesis. The CSE is a method to enhance synthesis efficiency by eliminating common sub-expressions in the original expression [74]. Fig. 42 (d) shows the expressions optimized by CSE, where common sub-expressions such as R_0~R_2 and A~F are extracted. The data flow of conventional arithmetic operations optimized using CSE for TI is illustrated in Fig. 42 (c), where three subtractions, 20 multiplications, and seven additions should be used for binary logic implementation. SC logic is obviously more efficient than the

conventional binary logic in case of TI. Fig. 43 (b)~(i) shows the resulting images of volume rendering using the above implementations for different precision levels. Compared with the reference image (a) generated by 32 bit floating point implementation, both 5 bit fixed point implementation (FIX) and 32 bit SC logic implementation (SC) have precision loss as shown in (d) and (g); in particular, grid noise is seen in 5 bit FIX in (e) and (f), while dot noise is seen in 32 bit SC in (h) and (i). In case of 8 bit FIX and 256 bit SC the quality is almost same as that of the reference image as shown in (b) and (c) (also, refer to the SNR values in Fig. 46 (b)).

Since direct synthesis has lowest efficiency as explained in Section 3.3.4, dot product application can be used to see the lower bound of the gain obtained by our approach. It has no algebraic divisor and thus has no SC kernel. In the experiment, the dot product application has 32 terms and 64 variables as follows,

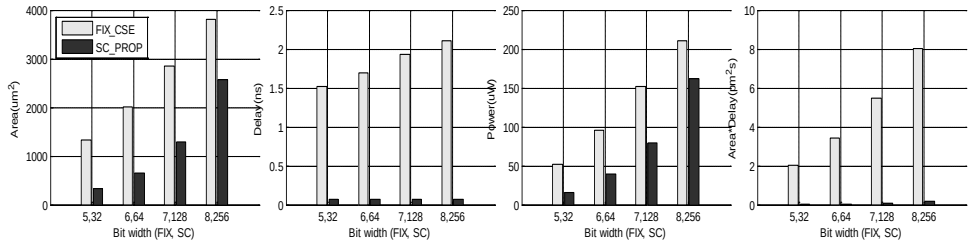
$$z = \sum_{i=1}^{32} x_i y_i, \tag{22}$$

where x_i's and y_i's are all different variables.

The runtimes of our proposed algorithm for TI and dot product are 25.06 and 5.35 seconds respectively.

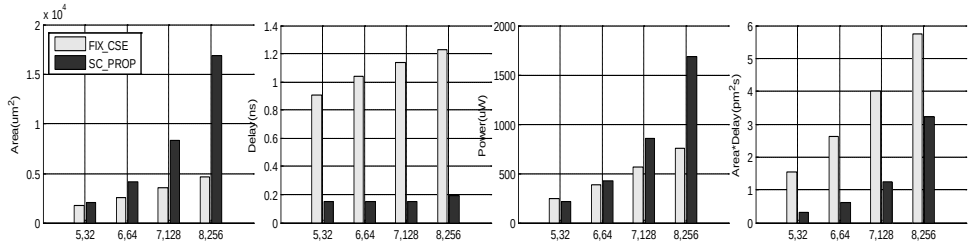
To compare the SC logic network generated by our algorithm with conventional binary logic implementation, we measure the gate area, critical path delay, and power dissipation of the four applications. They are implemented as combinational logic in TSMC 45nm technology library with Synopsys Design Compiler using Verilog HDL. We choose fixed point implementation (FIX) as the counterpart of SC logic implementation (SC), and for a fair comparison, we compare 5 bits of FIX with 32 bits of SC and 8 bits of FIX with 256 bits of SC.

As mentioned in Section I, SC logic can be implemented in a serial or parallel manner; it is also possible to mix them. If an SC logic network needs *N* bit streams, the serial implementation spends *N* clock cycles, while the fully parallel version with *N* duplications takes only one clock cycle. There is a trade-off between area



(a) (b) (c) (d)

Fig. 44. Performance comparisons for TI in volume rendering between SC and conventional binary representation. (a) Area (b) Critical path delay (c) Power (d) Area and delay product



(a) (b) (c) (d)

Fig. 45. Performance comparisons for Dot product. (a)Area (b) Critical path delay (c) Power (d) Area and delay product.

and delay. Thus, in order to measure the quality of serial, parallel, and mixed implementations, we use the area-delay product, which always gives us same metric regardless of parallelism for SC logic networks.

The performance results for the applications are shown in Fig. 44 and Fig. 45, where SC logic networks synthesized by the proposed algorithm (SC_PROP) are compared to FIX with CSE (FIX_CSE). SC_PROP outperforms FIX_CSE in every metric in TI. More specifically, our design is 25.92X faster on average compared to FIX_CSE, while taking only 42.23% of area and 50.20% of power. Speed gain of the SC logic network is due to the fact that critical path delay of one bit-line is

TABLE XI

ENERGY DISSIPATION (FJ)

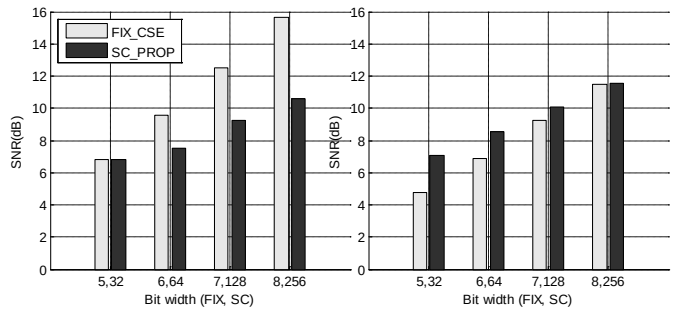
Application	Case	5,32 bit	6,64bit	7,128bit	8,256bit	SNR (dB) 8,256 bit
Volume rendering (Best case)	FIX_CSE	78.13	163.03	293.36	445.21	11.46
	SC_PROP	1.11	2.73	5.60	11.34	11.55
	Ratio	70.64	59.72	52.39	39.26	
Dot product (Worst case)	FIX_CSE	223.86	398.32	640.68	927.42	15.68
	SC_PROP	31.65	63.45	128.85	319.39	10.57
	Ratio	7.07	6.28	4.97	2.90	

significantly less (the path delay in TI is only for three MUXes) and all the bit-lanes of SC logic are completely independent of each other, thus enabling fully parallel operations. Serial implementation would increase the latency due to multiple clock cycles, but decrease the area and power instead.

In case of dot product, which allows direct synthesis method only, although SC_PROP is 6.77X faster on average compared with FIX_CSE, the area and power are increased by 1.89X and 1.26X respectively. This shows that direct synthesis version has no advantage compared to kernel-based synthesis and hybrid version. In terms of area-delay product, SC logic outperforms conventional binary logic in all the cases.

Energy dissipation is shown in Table XI. We obtain 55.50X energy reduction on average in volume rendering, while energy consumption in dot product is reduced by 5.31X on average. [21]

[21] In order for SC to be connected to conventional binary representation, randomizer and de-randomizer circuits are needed. However, it was not included in this work. If input data are generated from analog logic such as sensors [2], or a system is entirely operated in SC, the needs for randomizers are diminished. Also, there are researches on reducing the overhead of randomizers [40].



(a) Dot product (b) TI in volume rendering

Fig. 46. Accuracy comparison between FIX and SC logic. (a)SNR for dot product. (b)SNR for TI in volume rendering.

3.4.3 Comparison of Accuracy

As stated in Section I, SC logic with 2^K bit stream and conventional binary logic with K bit-width represent precision of 2^{-K}. However, due to probabilistic characteristics, SC inevitably has random errors for arithmetic operations, which degrades system accuracy. The result for dot product in Fig. 46 (a) shows that the SNR of FIX_CSE is better than that of SC_PROP as we have expected. In case of TI, however, we can make a more efficient structure than conventional binary logic as shown in Fig. 42 (a) and (c), and thus the SNR of SC logic can be higher than that of conventional binary logic as shown in Fig. 46 (b). For example, a MUX performs one subtraction, two multiplications, and one addition as mentioned in Section I (i.e., $y=(1-c)a+cb$), and thus four operations are simultaneously executed in SC, whereas the fixed point circuit has precision loss in each operation.

In both of (a) and (b), as the bit-width increases, the SNR growth of SC logic is slower than that of conventional binary logic. The reason is as follows. The bit stream of length N at the output of SC logic can be modeled as a Bernoulli sequence [1] [11], and the standard deviation of the Bernoulli sequence is known to be

inversely proportional to $\sqrt{N}$, which means that the accuracy increases slowly as N increases.

3.5 Conclusion

Stochastic computing (SC) is a promising design technology in terms of gate area, power, and error tolerance. However, synthesizing an SC logic network needs to search very large design space and requires a considerable amount of computation. In this paper, we proposed an approach to efficiently synthesizing SC logic for general arithmetic expressions even containing many variables. For this, we first define the concept of common form that can be implemented with logic gates and find basic building blocks called iSC kernels used for constructing SC logic networks. The design space can be pruned by using relationships between the iSC kernels. We applied the approach to real applications and demonstrated its effectiveness by generating SC logic that outperforms conventional binary logic.

4. An Energy-Efficient Random Number Generator for Stochastic Circuits

4.1 Introduction

The emerging classes of applications such as machine learning, computer vision, and computer graphics, have inherent resilience to errors and/or inaccuracy [75]. Nevertheless, the conventional computing performs accurate computations even for the applications that allow some level of inaccuracy. In such applications, stochastic computing (SC) achieves high efficiency in terms of silicon area and power consumption while having high error tolerance and massive parallelism, which is mainly due to its probabilistic nature implemented with conventional CMOS digital logic [5]. It is an attractive approach these days since the conventional binary approach is tardy in enhancing the efficiency. Due to the good characteristics compared with convention approach, SC has been studied for applications such as neural networks [76], low-density parity-check (LDPC) [77], median filters [78], image processing [79], and other applications allowing some errors.

In contrast to the conventional binary representation, SC uses random bit streams called *stochastic numbers* (SNs). For this reason, given conventional *binary numbers* (BNs), SC requires a circuit that converts the BNs to SNs; the circuit is called a *stochastic number generator* (SNG). However, an SNG spends a large amount of resource compared with the pure SC circuit in terms of area and power [80] [81] [1]. This degrades the advantage of SC in hardware cost and restricts the usability of SC.

Although there have been some studies for generating random bit streams in different domains [82] [83] [84], most of them focus on improving accuracy by

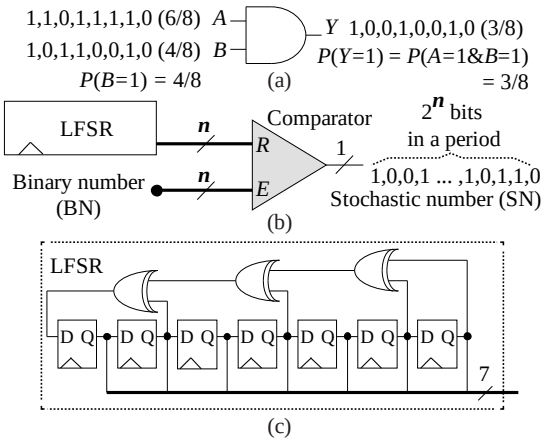


Fig. 47. Stochastic arithmetic operation and conventional stochastic number generator (SNG). (a) Multiplication of two stochastic numbers (SNs) and the output stream. (b) SNG with an LFSR. (c) An example of 7-bits LFSR.

enhancing randomness, while ignoring the hardware overhead. There has been a recent effort [1] in reducing hardware area by sharing a *linear feedback shift register* (LFSR) between two SNGs. However, simply sharing an LFSR among SNGs leads to accuracy degradation due to correlation of stochastic bits. This degradation rapidly increases as an LFSR is shared by more SNGs. In this paper, we suggest a much more efficient way of implementing SNGs, which generates asymptotically one stochastic bit per store unit, i.e., D-type flip-flop (D-F/F), in LFSR at a time. Moreover, it also enhances the accuracy of SC logic compared with previous approaches.

4.2 II. Background

4.2.1 Preliminaries

In the range of [0 1] (or [-1 1] in bipolar form), SC takes the signal probability of a bit stream as its real value; it is represented by the number of 1s over the length of

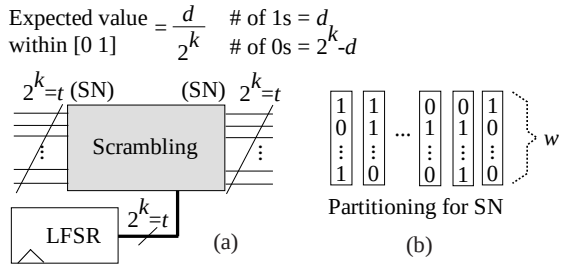
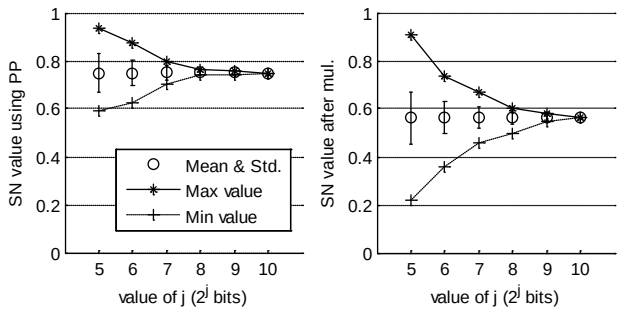


Fig. 48. Strategy for the proposed SNG. (a) Example of ideal SNG case, where one stochastic bit is generated by using one store unit (i.e., D-flip/flop). (b) Partitioning of stochastic bit-stream.

the bit stream. In the example of Fig. 47 (a), the eight-bit input stream *B* contains four 1s and thus represents a real number 4/8. The figure also shows arithmetic multiplication with an AND gate in which the output probability is derived by $P(Y=1) = P(A=1\&B=1) = 3/8$. In order to calculate arithmetic operations such as (a), SC requires random bit streams generated by SNGs, which can be implemented with a comparator and an LFSR consisting of D-F/Fs and XOR gates as shown in (b) and (c).

4.2.2 Shortcomings of Conventional Approaches

Conventional approaches activate the entire SNG circuit including the LFSR and the comparator in order to generate only a single stochastic bit. This means that the SNG circuit should be activated *n* times for one *n*-bit SN. The overhead increases as the required precision increases. We define the efficiency of an SNG as the number of bits it generates at a time, since it is related with the performance and energy consumption. Fig. 48 (a) shows a case that 2^k-bit SN stream is generated at a time by an SNG with a 2^k-bit LFSR, where the expected SN value is $d/2^k$, i.e., *d* among 2^k bits are 1s. It is obtained by scrambling 2^k input bits without the comparator shown in Fig. 47 (b). However, simultaneously scrambling 2^k (=*t*) bits may lead to high overhead in area and power consumption. Thus, we suggest to partition the bits



(a) (b)

Fig. 49. SN generation for 0.75 with progressive precision (PP) in 2^{10} bits SC circuit. (a) Distribution of SN values. (b) Multiplication of two SNs generated with PP using an AND gate.

into groups as shown in Fig. 48 (b), where each group has w bits ($w < t$) and the SNG circuit processes one group at a time.

The second shortcoming of the conventional SNG in Fig. 1 (b) is in progressive precision (PP) which is known to be one of advantages of SC compared with conventional binary representation.[22] Suppose that the SNG circuit can generate a bit stream with $N=2^{10}$-1 bits (i.e., the bit width of LFSR is 10), thus has a precision of 1/N. Consider using the circuit to generate an SN with a value 0.75. It can accurately generate an SN that has the expected value 0.75 because each of the N different numbers generated by the LFSR appears once during the period of generating the N numbers. However, if we take only 2^5 bits with PP, for example, the value may not accurately represent 0.75. A substring of the original N bit stream cannot exactly represent the statistics of the entire period of the original stream. Fig. 49 (a) shows the distribution of SN values generated as substrings of an N bit stream, where the precision varies between $1/2^5$ and 1/N and the expected value is 0.75. Fig.

[22] Precision in SC can be easily changed by adjusting the length of the bit streams without hardware modification, whereas the precision of conventional binary logic circuit cannot be changed with fixed hardware.

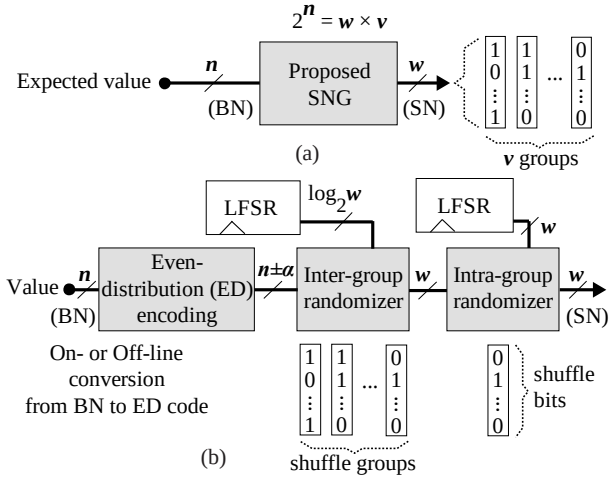


Fig. 50. Overview of the proposed SNG. (a) SNG with BN input and SN output, where *v* groups are generated and each group has *w* bit-width. (b) The proposed SNG consists of three parts such as even-distribution (ED) encoding, inter-group, and intra-group randomizer with LFSR input.

49 (b) depicts the distribution of SN value after multiplication of two SNs by using an AND gate; the error becomes bigger as the bit width becomes smaller.

Instead of using a substring of a long bit stream, the shorter bit streams generated in groups as shown in Fig. 2 (b) can be used for PP (e.g., one group of w bits can be used for precision of $1/w$ or two groups can be used for precision of $1/2w$). When we partition the entire bit stream into multiple groups as shown in Fig. 2 (b), it is important to evenly distribute 1s over the groups to reduce the variance. If we can do this, then the partitioning technique in Fig. 48 (b) can be a solution to the aforementioned problem of PP.

Another problem is that an SNG for higher precision (e.g., 10 bits) is activated for lower precision (e.g., 5 bits) when we use PP, which leads to overhead in terms of power and delay. In our approach, since the circuit is not constructed for the entire precision, but for a group precision, the problem of overhead is also solved.

4.3 III. Proposed Stochastic Number Generator

4.3.1 Overview of the Proposed SNG

Instead of generating new 0s and 1s, the proposed SNG shuffles 1s in the existing bit stream by using a random source. The basic idea of the proposed SNG is to evenly distribute 1s over the entire bit stream, which is named as *low-discrepancy* (LD); the usability of LD is mentioned in [6].

Fig. 49. shows the outline of the proposed SNG in (a), which consists of three parts as shown in (b): even-distribution (ED) encoding, inter-group randomizer, and intra-group randomizer. ED encoding makes 1s evenly distributed within all groups; inter-group randomizer shuffles the order of groups; intra-group randomizer scrambles bits within a group. All steps are implemented by combinational logic (except for two LFSRs), which makes the proposed SNG more efficient in terms of area and power.

4.3.2 Even-distribution Encoding

The objective of even-distribution (ED) encoding is to construct v groups, where the difference of the number of 1s between groups should be less than or equal to b. In other words,

$$\max_{i \in v} g_i - \min_{i \in v} g_i \leq b \qquad (1)$$

where g_i is the number of 1s for the i-th group. Fig. 51 (a) and (b) respectively shows two examples of ED encoding when b is set to 3 and 1, where each column represents a group and each row represents a digit in a group. Each digit has a weight of $2^j b$, where j is the index of the digit within the group with j=0 for the least significant digit (note that it is different from the conventional radix number having weight of r^j, where r is the radix and j is the index of the digit). Thus, the

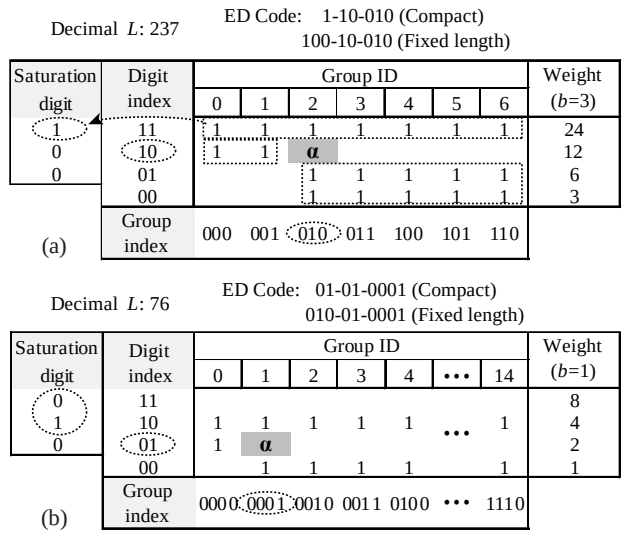


Fig. 51. Even-distribution (ED) encoding, where white space means zero. (a) ED code represents decimal number 204 with seven groups (column) and four digits (row) per group when *b* is 3. (b) Example of ED code with 15 groups when *b* is 1.

total weight of each group becomes $\sum_{j=0}^{k} 2^j b$, where k is the index of the most significant digit in a group. For example, Group 1 in (a) has 1100 (36 = 24x1 + 12x1 + 6x0 + 3x0) while Group 3 is has 1011 (33 = 24x1 + 12x0 + 6x1 + 3x1); the difference of the two groups is 3, which is bounded by *b*.

Given a decimal number L that is to be generated by the SNG, ED code is generated by finding the *pivot* position $'\alpha'$[23] from L as shown in Fig. 51. The upper side of α should be all zeros or all ones; we call the digits as *saturation digits*. The left side of α should be all ones whereas the right side of α should be all zeros

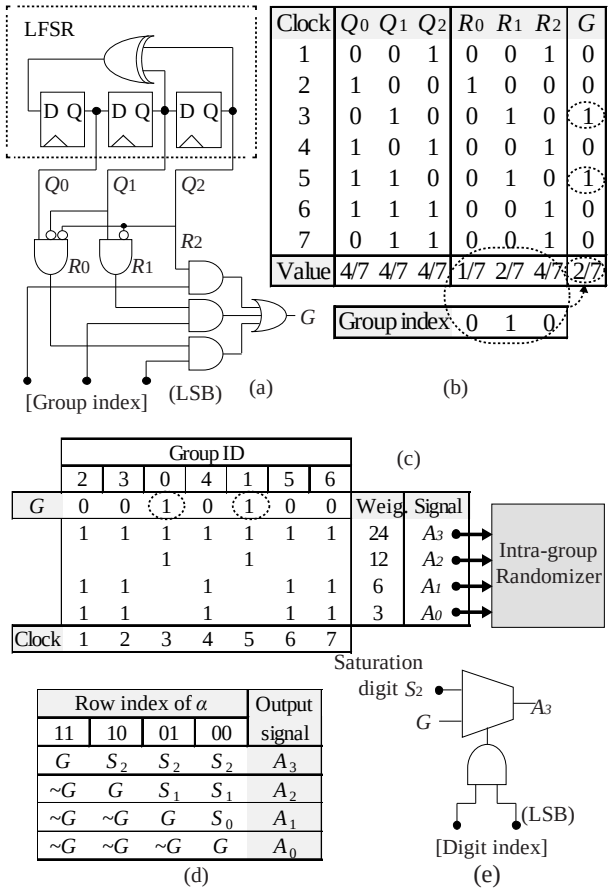
[23] α is adopted for an easy explanation on how to generate ED code.

(white space means zero in the illustration). The lower left side should be all zeros, whereas the remaining lower space should be all ones. Following the rule, the difference of all groups is limited within b because $2^k b - \sum_{j=0}^{k-1} 2^j b = b$, where k is the digit (*row*) index of α (the rows above α can be ignored because they have the same values). ED code is a tuple of the saturation digits, the digit index, and the group (*column*) index of α in order.

For example, ED codes of the decimal number 237 and 76 are 1-10-010 and 01-01-0001 in (a) and (b), respectively. The length of saturation digits in ED code is various according to the row index of α; for example, it becomes zero, one, two, and three when the row index is 11, 10, 01, and 00, respectively. It is possible to use a fixed length encoding for the saturation digit by taking $R - 1$ bits, where R is the number of rows.

The granularity of ED encoding is b, which means that the error due to encoding of a decimal number is at most $\lfloor b/2 \rfloor$. Note that the encoding error is zero when b is one such as Fig. 5 (b). Since using larger b reduces the number of ED code digits, we can achieve efficiency in terms of digit storage and other logic area while sacrificing accuracy. The conversion from a decimal number to ED code can be conducted on- or off-line. The circuit to convert a conventional binary number into an ED code is implemented by using combinational logic with a very small number of gates (it is omitted due to space limitation).

4.3.3 Inter-group Randomization



Clock	Q_0	Q_1	Q_2	R_0	R_1	R_2	G
1	0	0	1	0	0	1	0
2	1	0	0	1	0	0	0
3	0	1	0	0	1	0	1
4	1	0	1	0	0	1	0
5	1	1	0	0	1	0	1
6	1	1	1	0	0	1	0
7	0	1	1	0	0	1	0
Value	4/7	4/7	4/7	1/7	2/7	4/7	2/7

Group index	0	1	0

	Group ID						
	2	3	0	4	1	5	6
G	0	0	1	0	1	0	0
	1	1	1	1	1	1	1
			1		1		
	1	1		1		1	1
	1	1		1		1	1
Clock	1	2	3	4	5	6	7

Weig.	Signal
24	A_3
12	A_2
6	A_1
3	A_0

Row index of α				Output signal
11	10	01	00	
G	S_2	S_2	S_2	A_3
$\sim G$	G	S_1	S_1	A_2
$\sim G$	$\sim G$	G	S_0	A_1
$\sim G$	$\sim G$	$\sim G$	G	A_0

Fig. 52. Inter-group randomization, where the group index is from Fig. 51a. (a) Shuffling circuit for inter-group randomization. (b) Final signal *G* is a scrambled one and the value 2/7 matches with the number of BGs. (c) Shuffling of groups according to the value of G. (d) Method to actually generate the output signal, where '~*G*' means negation of *G*. (e) Example of logic for output signal A_3.

After allocating 1s to the groups through ED encoding, the sequence of the groups is scrambled in inter-group randomization step. For example, in Fig. 51 (a), the order of seven groups is shuffled to increase randomness, by using the column index. We adopt the technique used in [85] to scramble bits by using LFSR as follows.

Although we try to keep balance among the groups, some groups have a bigger value than the others (e.g., groups 1 and 2 in Fig. 51 (a) are bigger than other groups). Let us define the groups having bigger value as BGs. The objective of inter-group randomization is to shuffle the BGs with other groups. Fig. 52 (a) shows a circuit of inter-group randomization for the example in Fig. 51 (a), where LFSR is used as a random source. In the example, the values of bit streams R_0, R_1, and R_2 become 1/7, 2/7, and 4/7, respectively. We combine the input column index of α with R_0, R_1, and R_2 using AND and OR gates to generate a new placement of the columns as illustrated in column *G* of Fig. 52 (b). BGs are placed where *G* is one while the other groups are placed where *G* is zero as shown in Fig. 52 (c). Signals A_0, A_1, A_2, and A_3 representing the digits in each group are passed into intra-group randomizer in order to scramble all the bits within a group. The output signals from inter-group randomization are actually generated by using saturation bit, row index of α, and *G*; Fig. 52 (d) shows how each signal is constructed. It can be simply implemented by using a multiplexer with row digit as control input; the logic for signal A_3 is shown in Fig. 52 (e). Note that the entire circuit for inter-group randomization is also implemented by using combinational logic except for the LFSR.

4.3.4 Proposed Building Block for Bit Shuffling

The signals from inter-group randomization such as A_0, A_1, A_2, and A_3 in Fig. 52 are scrambled by using a randomizing network as shown in Fig. 53 (a), where W' is a set of shuffled signals obtained from *W*. We suggest a building block consisting of two multiplexers as shown in Fig. 53 (b). It swaps two input signals (*M* and *N*) into outputs (*X* and *Y*) according to the selecting signal *S* having signal probability of '0.5'; we call it a *swapper* and use the symbol in (d) instead of (b).

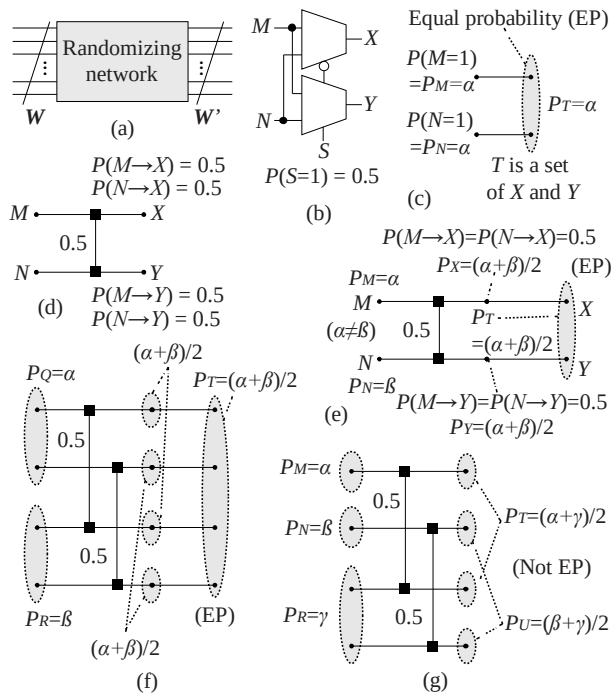


Fig. 53. Analysis of randomness after swapping signals. (a) Overview of randomizing network. (b) Swapping logic. (c) Definition of equal probability (EP) set. (d) Probability for input signals to be passed onto output signals. (e) Constructing an EP set by using a swapper. (f) Constructing an EP set of output signals from two different EP sets of input signals. (g) A case that output signals do not belong to an EP set.

Definition 1. (Equal probability set) An equal probability (EP) set is a set of signals that have the same signal probability. □

Let PM denote P(M=1), i.e., the signal probability of M. As shown in Fig. 53 (c), suppose two input signals M and N have the same signal probability α (i.e., PM = PN = α). Then the set T containing M and N is an EP set. In this case, we use PT to denote the signal probability of a signal in T. Thus PT = α.

Lemma 1. (Two input signals and swapper) Even though the signal probabilities of two input signals are different, the output signals of the swapper belong to an EP set.

Proof. If two signals M and N with different signal probabilities (i.e., PM=α and PN=β) pass through a swapper, the output signals X and Y have the same signal probability PX = PY = (α+β)/2 and thus, belong to an EP set. □

Lemma 2. (Two EP sets and swapper) Two EP sets having f signals can be merged into one EP set having 2f signals by using f swappers, where each swapper takes two input signals, one from each input EP set..

Proof. As shown in Fig. 53 (f), if two sets Q and R with PQ=α and PR=β pass through a swapper, the output set T becomes an EP set with PT=(α+β)/2 because of Lemma 1. Since f pairs are processed at the same time, f swappers are used. □

Fig. 53 (g) shows that output signals do not belong to an EP set because one input set is not an EP set (i.e., $P_M \neq P_N$).

4.3.5 Intra-group Randomization

The signals generated by the inter-group randomization are scrambled by using the intra-group randomization, where the randomizing network mentioned in the previous section is used. Fig. 54 shows the entire intra-group randomization circuit consisting of three-step randomizing network and an LFSR. In Step 3, A_2 is swapped with the result from Step 2; in Step 2, A_1 is swapped with the result from Step 1. In Step 1, however, there is no previous step and thus A_0 is swapped with a dummy signal. Because the size of A_0 is b ($=2^0 b$), b-bits are provided for the dummy signal. Since we are adding these dummy bits in this stage, the number L used in the even-distribution encoding stage (see Section III.B) should be adjusted as follows.

$$\begin{cases} \text{if } L \geq (b \cdot v), \text{ then } L \leftarrow (L - b \cdot v),\ S \leftarrow 1 \\ \text{otherwise, } L \leftarrow L,\ S \leftarrow 0 \end{cases} \quad (2)$$

where v is the number of groups and S is the *global saturation bit* to be used to set the dummy bits. If S is 1, then all the dummy bits are set to 1. Otherwise, they

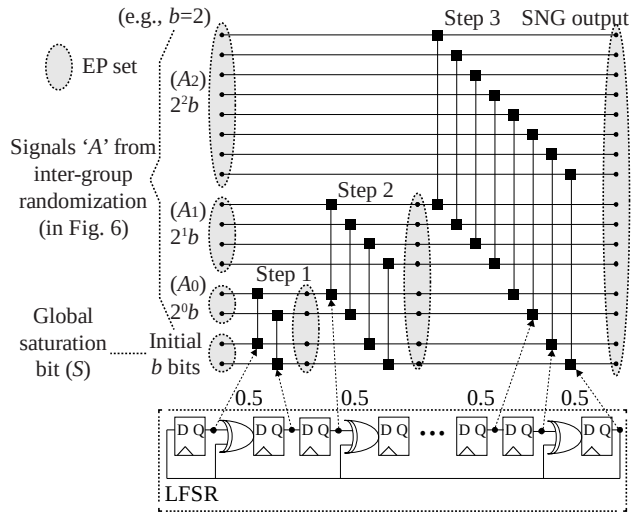


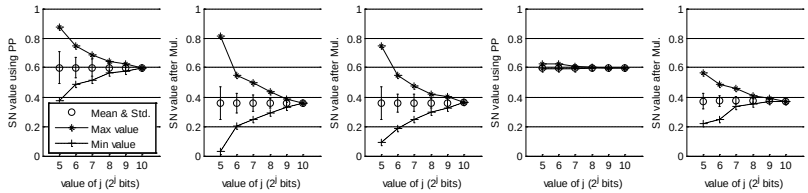
Fig. 54. Logic for intra-group randomization consisting of randomizing network and an LFSR. The input signals is from inter-group randomization.

are set to 0.

In case of Step 1, since two input signal sets are EP sets, the output signal set is also an EP set from Lemma 2; In Step 2, because two four-bit signal sets are EP sets, the output eight-bit signal set is also an EP set; the same applies to the following steps. Thus, the final output signals of the entire circuit belong to an EP set. Note that the gate delay is proportional to the number of steps because the swapper within a step operates in a parallel manner, which means that the critical path delay is relatively short.

We use a D-F/F of an LFSR as the random source with 0.5 probability as used in [86]. Galois LFSR [87] is adopted in order to reduce dependency between two adjacent D-F/Fs. Regarding the scalability, the number of swappers, i.e., the number of D-F/Fs in an LFSR is identical to the number of bits generated from a group, which means that one store unit (i.e., D-F/F) effectively generates one random bit at a time; it is different from conventional approaches as mentioned in Section II.B.

4.4 Experimental Results



(a) Conventional SNG (b) Conv. SNG (Mul.) (c) Shared SNG (Mul.) (d) Proposed SNG (e) Prop. SNG (Mul.)

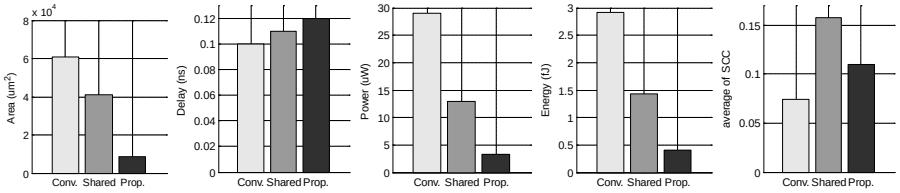
Fig. 55. Accuracy after generating SN value 0.6 and multiplying them while using PP.

In this section, we compare three SNGs: the conventional SNG as shown in Fig. 47 (b), the previous work [1] sharing LFSRs, and the proposed SNG. The length of LFSRs in the conventional SNG and the shared SNG [1] is 10; in the proposed SNG, a group has 32 bits. They are implemented in TSMC 45nm technology library with Synopsys Design Compiler using Verilog HDL.

4.4.1 Accuracy of Generated Stochastic Bit Stream

Fig. 55 (a) and (d) shows the accuracy of bit streams generated from the conventional SNG and the proposed SNG, respectively, in terms of mean, standard deviation, max, and min value; In the case, the expected value is 0.6. The accuracy of the shared SNG [1] is identical to (a) because it simply shares an LFSR between two SNGs. In particular, when exploiting PP (e.g., using 2^5 bits in 2^{10} period), the proposed SNG generates almost exact values whereas the values from the conventional SNG severely deviate. In case of multiplication of the stochastic bits by using an AND gate, the proposed SNG ((e)) shows better result than the others ((b) and (c)).

4.4.2 Area, Delay, Power, Energy and SCC Average



(a) Area (b) Critical path delay (c) Power (d) Energy (e) SCC average

Fig. 56. Comparison of area, critical path delay, power, energy, and SCC average value to generate a 2^{10}-bit stream for conventional SNG, SNG sharing two LFSRs [1], and the proposed SNG. For fair comparison, all cases are implemented in parallel manner. The length of LFSRs in two previous approaches is 10 while the number of bits in a group of the proposed SNG is 32.

As shown in Fig. 56, compared with the conventional SNG, the proposed SNG decreases area ((a)), power ((c)), and energy ((d)) by 85.7%, 88.6%, and 86.3%, respectively, where all SNGs are implemented in parallel manner for fair comparison (i.e., generating 2^{10} bits every clock cycle). Compared to the shared SNG [1], the proposed SNG decreases the same metrics by 78.9%, 74.4%, and 72.1%, respectively. However, the critical path delay ((b)) of the proposed SNG increases by 20% and 9% compared to the conventional and shared SNG, respectively. In terms of correlation, we use average of SCC [1] [88] which has smaller value when two SN bit streams are less correlated. The values for the conventional, shared, and proposed SNG are 0.074, 0.157, and 0.110, respectively, which means that the proposed SNG is between the two previous approaches in terms of correlation.

4.4.3 Energy Efficiency When Operated under Maximal Precision

Considering that the maximal precision is fixed and that each application has a desired precision in many cases, a system is operated in lower precision compared to maximal precision. Fig. 57 shows the energy consumption when generating bit

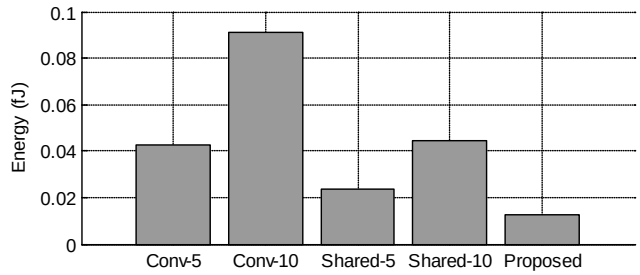


Fig. 57. Energy to generate 2^5 bits. Conv-5 and Conv-10 represent that the length of LFSR in the conventional SNG are 5 and 10, respectively. Shared-5 and Shared-10 mean the shared SNGs.

streams with desired precision of $1/2^5$, while the maximum precisions are $1/2^5$ and $1/2^{10}$, respectively. Conv-5 and Conv-10 mean that the length of the LFSR in the conventional SNG is 5 and 10, respectively. The figure shows that the proposed SNG is more energy-efficient. It is because the entire LFSR is activated for one stochastic bit in the previous approaches. However, in the proposed SNG, 2^5 bits are simultaneously generated as shown in Fig. 54, while maintaining the accuracy regardless of the maximal precision.

4.5 Conclusion

Stochastic computing is one of promising approaches since it requires small hardware footprint and low energy, and provides high error tolerance. However, it requires random bit streams generated by stochastic number generators (SNGs), which incurs area and energy overhead. In this paper, we proposed an area- and energy-efficient SNG even enhancing accuracy in progressive precision (PP). Experimental results show that our SNG outperforms the existing approaches in terms of area, power, energy, and accuracy.

5. Approximate De-randomizer for Stochastic Circuits

5.1 Introduction

Stochastic computing (SC) is an alternative paradigm to conventional binary arithmetic computing [5]. SC can boost efficiency in terms of area, power, and error tolerance while relaxing the accuracy of computation in emerging applications such as machine learning, computer vision, and computer graphics. Numbers in SC are represented by the probability of 1′s occurrence in a random bit stream. For example, in Fig. 47 (a), since the occurrences of 1′s in the three 8-bit streams at *A*, *B*, and *Y* are 6, 4, and 3, respectively, the corresponding *stochastic numbers* (SNs) are $P(A=1)=6/8$, $P(B=1)=4/8$, and $P(C=1)=3/8$. The single AND gate performs multiplication (i.e., 6/8 x 4/8 = 3/8).

For an efficient conversion between *binary numbers* (BNs) and SNs, there have been researches on converting from BNs to SNs (aka randomization) and vice versa (aka de-randomization). In this paper, we focus on the conversion from SNs to BNs. Converting an SN into a BN requires counting the number of 1s in the random bit stream. Fig. 47 (b) and (c) show a serial counter and an *accumulative parallel counter* (APC) [89] [90], respectively, where flip-flops are used with some other logic for accumulation. [24] The *parallel counter* (PC) in an APC uses full adders (FA) in order to generate 2^0~2^{m-1} weighted bits (i.e., a BN) from a stream of 2^0 weighted

[24] In this paper, we consider only APC because the serial counter consumes large amounts of energy.

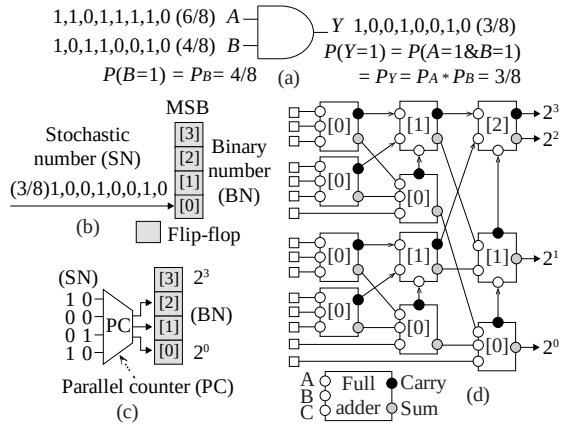


Fig. 58. Stochastic numbers (SNs) and conventional counters, where the numbers in brackets represent the bit index of binary numbers (BNs). (a) Multiplication of two SNs. (b) Accumulative serial counter. (c) Accumulative parallel counter (APC). (d) Example of a parallel counter (PC) converting 15 bits SN into 4 bits BN.

input bits (i.e., an SN), where *m* is the number of bits of the BN. Fig. 47 (d) shows an example of PC generating 4 bits BN from 15 bits SN.

Considering that SNs are based on the probabilistic nature of random bit streams and thus the accuracy is already compromised during SC, there is no reason to stick to accurate conversion using the conventional PC. Note that using the conventional PC may lead to lots of inefficiency in the aspect of entire workload, since the length of bit streams in SN is much longer than that in BN; only *k* bits in BN becomes 2^k bits in SN.

In this paper, we suggest an approximate PC exploiting two properties of SN: 1) inaccuracy due to randomness and 2) long bit stream. The proposed PC can be implemented with a smaller circuit compared with the conventional accurate PC, where the inaccuracy problem can be alleviated due to the long bit stream.

5.2 Proposed Approximate Parallel Counter

The proposed approximate PC is shown in Fig. 59 (a), which consists of two parts: an approximation unit (AU) and a conventional accurate PC. The former is implemented by simple gates such as AND and/or OR gate; Fig. 59 (b) shows a 2-layer AU. The approximate PC exploiting a 1-layer AU is shown in (c). The input weight of AU is 2^0 while the output weight becomes 2^l, where l is the number of layers.[25] (e) shows errors for 1-layer AU using AND or OR gate. Note that AND gates generate negative errors while OR gates generate positive errors.

5.2.1 Analysis for Gate Count in 1-layer Approximate PC

In order to see how much area reduction can be obtained by the proposed approach, we calculate the number of FAs in Fig. 47 (d) and Fig. 59 (c). Suppose that N-input bits become v-output binary bits ($N=2^v-1$) and $f(v)$ is the number of FAs. In case of the conventional PC (i.e., Fig. 47 (d)), when v increases by one, the number of FAs increases by two times plus $v-1$. Thus, $f(v) = 2 \cdot f(v-1)+v-1 = 2^v-v-1$, where $f(2)=1$ (i.e., one FA is needed to generate two-digit binary numbers). The number of gates in the conventional PC is given by

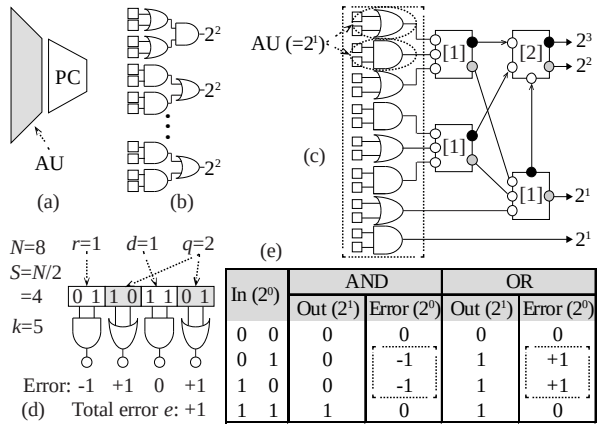
$$G_{conv}(N) = \{(N+1) - \log_2^{N+1} - 1\} \cdot 5, \qquad (1)$$

considering that $v = \log_2^{N+1}$ and FA consists of five gates. The proposed PC with 1-layer AU, as shown in Fig. 59 (c), uses $f(v-1)$ FAs for v-output bits and $N/2$ additional AND or OR gates, where $N=2^v$.[26] Thus, the number of gates in the proposed PC is,

---

[25] Due to space limitation, we explain only 1-layer AU in this paper.

[26] In this case, we use $N=2^v$ instead of $N=2^v-1$, since it fits better with the use of two input gates. The generated BN cannot represent the maximum value of SN, but the error is small and thus can be ignored.



In (2^0)		AND		OR	
		Out (2^1)	Error (2^0)	Out (2^1)	Error (2^0)
0	0	0	0	0	0
0	1	0	-1	1	+1
1	0	0	-1	1	+1
1	1	1	0	1	0

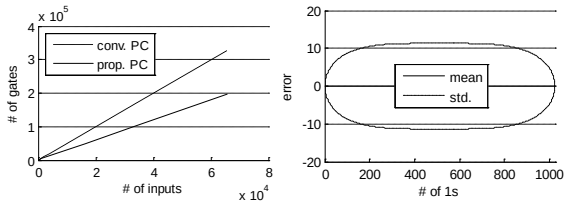
Fig. 59. The proposed parallel counter (PC). (a) Overview of the PC. (b) 2-layer approximate unit (AU). (c) The proposed PC using 1-layer AU, converting 16-bit SN into 4-bit BN. (d) Example of 1s distribution in 1-layer AU. (e) Output and error for all inputs in AND and OR gate.

$$G_{prop\text{-}1}(N) = \{N/2 \; - \; \log_2{}^N\} \cdot 5 + N/2, \qquad (2)$$

Fig. 60 (a) shows the number of gates for (1) and (2), where the proposed PC using 1-layer AU reduces gate count by about 40% compared to the conventional PC.

5.2.2 Analysis for Error in 1-layer Approximate PC

In order to analyze the effect of approximation and error, we define e as the output error in number of 1s and $T1(N,k,e)$ as the probability mass function (PMF), where N is the length of the given bit stream and k is the number of 1s in the input. In other words, given k 1s in the N-bit stream, $T1(\cdot)$ shows the probability of error e generated by the proposed approximate PC in Fig. 59 (c). The PMF is given by



(a) (b)

Fig. 60. Theoretical analysis of the proposed scheme. (a) The number of gates for the conventional PC and the proposed PC with 1-layer AU. (b) The mean and standard deviation of number of errors for 1-layer PC with 1024 input bits.

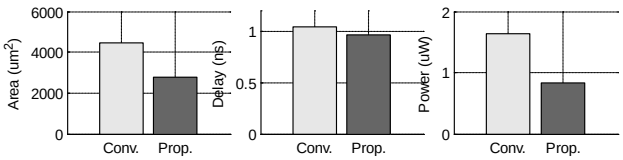
$$T1(N,k,e)=\sum_{e=q-r}\frac{\binom{S/2}{q}\binom{S/2}{r}\binom{S-q-r}{d}\cdot 2^q 2^r}{\binom{N}{k}}, (3)$$

where S is the number of output bits (slots) of the AU (S=N/2), and d is the number of slots containing two 1s at the inputs (d=(k−e)/2), and q is the positive error of OR gates while r is the negative error of AND gates. Fig. 59 (d) shows five 1s in the 8-bit input stream. Note that q+r is the number of errors and is the same as the number of slots containing one 1-bit and that the cumulative error e is q−r.

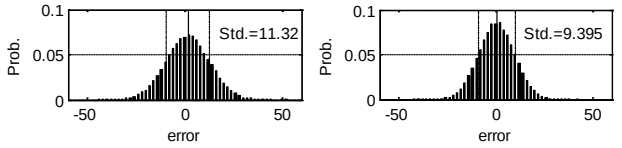
Fig. 60 (b) shows a theoretical result for the error PMF of $T1$(.), given 1024-bit streams. The mean of error values of $T1$(.) is zero and the maximum standard deviation of the errors is only 11.3 (about 1.1%). This means that there is no bias, and that the error is within only 1.1% for about 70% among all trials (within 2.2% for 95%).

5.3 Experimental Results

We compare the conventional PC and the proposed PC with heterogeneous 1-layer AU, given 1024-bit streams. They are implemented in TSMC 45nm technology library with Synopsys Design Compiler using Verilog HDL. As shown in Fig. 61, the proposed PC decreases area ((a)), critical path delay ((b)), and power



(a) Area (b) Critical path delay (c) Power



(d) PMF (512 1s) (e) PMF (128 1s)

Fig. 61. Experimental results of the proposed approximate PC compared with the conventional PC in 1024-bit stream. (a) Area. (b) Critical path delay. (c) Power. (d) PMF when 512 1s among 1024 bits. (e) PMF when 128 1s.

((c)) by 38.3%, 7.6%, and 49.4%, respectively. (d) and (e) show the errors of 1024-bit streams containing 512 and 128 1s, where the means of errors are almost zero and the standard deviations are 11.32 and 9.40, respectively. The results match well with the theoretical analysis results.

5.4 Conclusion

Although de-randomizer is a very important component for stochastic circuits, it has not been paid attention in the literature of stochastic computing (SC). Considering that SC is based on inaccurate computation, using a conventional accurate parallel counter (PC) in SC leads to inefficiency. We have proposed an approximate PC, which outperforms the conventional PC in terms of area, delay, and power, with no bias and small standard deviation of errors.

6. Dynamic Energy-Accuracy Trade-off Using Stochastic Computing in Deep Neural Networks

6.1 Introduction

Deep neural networks (DNNs) dramatically improve the accuracy of machine learning applications such as object detection [91] and speech recognition [92] that need the intelligence of human. However, compared with other machine learning techniques such as support vector machine (SVM), decision tree, and k-nearest neighbor (KNN), DNNs typically require a lot more computations due to many layers and many neurons comprising the network. Moreover, the industrial and academic needs tend to increase the size and complicate the topology of DNNs [93]. Because of this, using high performance computers with accelerators such as GPUs and/or clustering a bunch of machines is regarded as a practical solution to implementing DNNs [94]. Considering, however, that machine learning has also been rapidly adopted in mobile and embedded systems such as self-driving car [95] and patient data analysis [96] with limited resources, researchers have paid great attention to finding possible ways of efficiently executing DNNs including minimizing the required precision [97] and reducing the size of network [98].

In contrast to those studies based on conventional binary arithmetic computing, a different type of computing such as stochastic computing (SC) can be an attractive solution. SC uses the probability of 1s in a random bit stream to represent a number. For example, six 1s in an eight-bit stream in unipolar encoding represent 6/8 as shown in Figure 62 (a). SC can implement a circuit with smaller hardware footprint, lower power, and shorter critical path delay compared with conventional binary

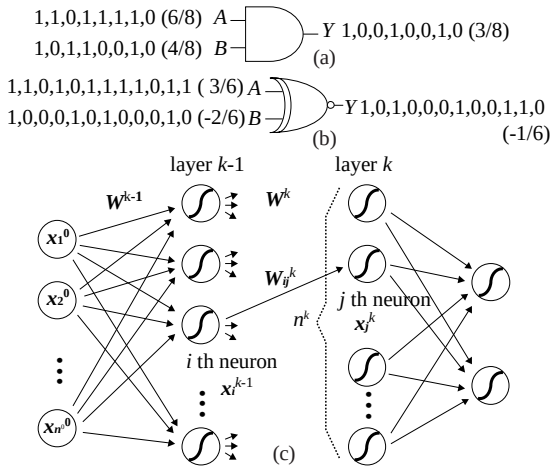


Figure 62. Deep neural network (DNN) using stochastic computing. (a) Stochastic multiplication in unipolar encoding with the range [0 1]. (b) Bipolar stochastic multiplication with [−1 1] range. (c) DNN layers with weight vector W^k in layer k.

logic. It also has advantages in error tolerance and bit-level parallelism. Considering that the majority operations of s DNN is multiplication, SC has great advantage in implementing a DNN because a single AND gate in SC can execute multiplication as shown in Figure 62 (a) with the unipolar encoding having range [0 1].

However, compared with conventional binary arithmetic, SC has some limitations such as small operational range [−1 1] in bipolar encoding (or [0 1] in unipolar encoding), random error fluctuation, and inefficiency of accumulation. In this paper, we present how we can alleviate the problems that we encounter with when designing a DNN using SC.

Neural network exploiting SC was first introduced in the noticeable research of [99], where a state-machine based approach was used for implementing an activation function. However, it was not a DNN but a two-layer autoencoder. Although a research for hardware implementation of SC for deep belief network was studied recently [100], only the multiplication part was implemented with SC.

6.2 Background

Because DNN commonly requires negative numbers as well as positive numbers, we use bipolar encoding in this paper. Bipolar stochastic number can be calculated from unipolar number as $p_{bipolar} = 2*p_{unipolar} - 1$. For example, as shown in Figure 62 (b), because nine 1s in a 12-bit stream is 9/12 in unipolar, it is 3/6 in bipolar encoding. An XNOR gate can be used in bipolar encoding to perform multiplication such as $(3/6) \times (-2/6) = (-1/6)$.

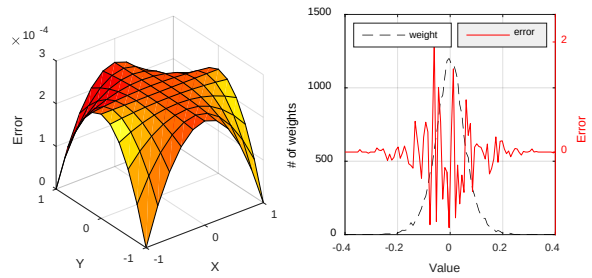
A DNN consists of neurons executing multiplication, accumulation, and activation function. Sigmoid or hyperbolic tangent is a typical form of the activation function. In Figure 62 (c), the operation of the j^{th} neuron in layer k (i.e., $\boldsymbol{x}_j^k$) can be defined as follows,

$$\boldsymbol{x}_j^k = af\left(\sum_{i=0}^{n^k-1} \boldsymbol{W}_{ij}^k \boldsymbol{x}_i^{k-1}\right), \quad (1)$$

where $\boldsymbol{W}_{ij}^k$ is a synaptic weight between $\boldsymbol{x}_i^{k-1}$ and $\boldsymbol{x}_j^k$ in layer k; n^k is the number of neurons in layer k; $af(\cdot)$ is the activate function.

6.3 Challenges to Apply SC to DNN

When adopting SC to DNNs, some obstacles should be solved in order to reach the accuracy level achieved by conventional floating-point or fixed point arithmetic. We found that directly applying SC to DNN lead to severe accuracy degradation which is not acceptable in common cases. It happens when synaptic weights are initialized to random numbers with a normal distribution around zero (mean is zero and standard deviation is given by $m^{-1/2}$ where m is the number of inputs to a neuron) as recommended in [101]. In addition, the weights are close to zero due to L1-, L2-regularization which gives penalties to non-zero parameters in order to prevent overfitting [102]. As a result, synaptic weights are aggregated near zero. However, as shown in Figure 63 (a), the XNOR(X, Y) operation representing



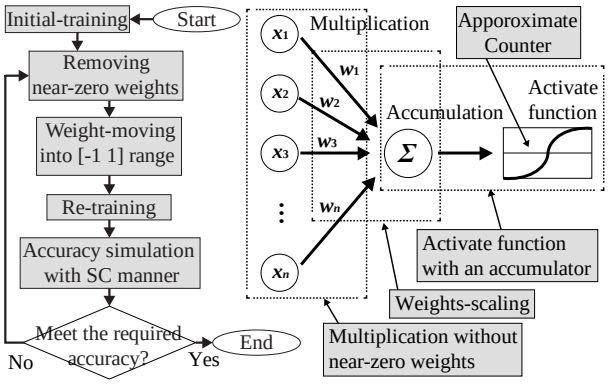
(a) Error of XNOR gate (b) Error of near-zero weights

Figure 63. Random error problem occurs when applying SC to DNN. (a) Random error of XNOR gate in 1024-bit stream as absolute value. (b) 20000 weights distribution in 200x100 networks (left Y-axis) and error multiplying by zero (right Y-axis).

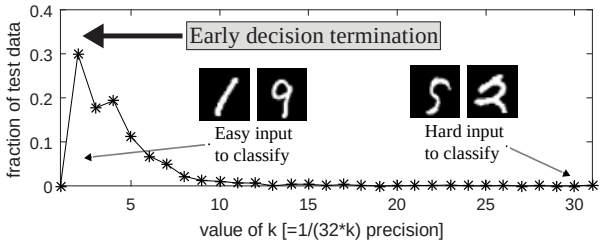
multiplication in bipolar encoding generates large random errors near the zero values of X and Y. Unfortunately, because many synaptic weights exist near zero, accumulating the products of synaptic weights and inputs increases the error to an acceptable level in our investigation. Figure 63 (b) shows the distribution of 20000 synaptic weights concentrated near zero (dashed line and left Y-axis) and the error of the sum of the products[27] (solid line and right Y-axis). In terms of signal-to-noise ratio (SNR), the results get worse, because the signal values (i.e., the products) tend to be small for near-zero weights.

The second problem in applying SC to DNN is the accumulation of many products. In SC, accumulation can be implemented with a multiplexer (MUX) known as scaled addition or an OR gate known as saturated addition. The former sacrifices the precision due to scaling down (or taking average of inputs) while the latter is too sensitive to the correlation of inputs. Another problem is that SC has

[27] To calculate the error of the sum of products at each weight value, the products were obtained by multiplying all the weights for the same value with 0 input. This is to show the tendency of errors appearing at the output of the accumulator.



(a) Training procedures (b) Testing with the SC neuron



(c) Early decision termination

Figure 64. Overview of the proposed procedures and main idea. (a) Training procedure for DNN using SC with 32-bit floating-point computation. (b) SC neurons are operated with SC exploiting the suggested solutions in testing phase. (c) Early decision termination by using progressive precision of SC.

range limitation from −1 to 1 in bipolar encoding. Thus arithmetic operations in SC can be easily saturated if special care is not taken.

6.4 DNN Using Stochastic Circuit

6.4.1 Overview of the Proposed DNN using SC

To use a DNN, we first need to train it (training phase) using training data and then test it with new test inputs (testing phase). We apply SC to testing phase

because DNN is mostly operated in testing phase once it is trained in a high-performance system such as a super-computer. Figure 64 shows the proposed training procedure that supports our approach of using SC for DNN. The training performed in a 32-bit floating-point system consists of initial-training, removing near-zero weights, incremental weight-moving into [−1 1] range, and re-training as shown in Figure 3 (a); accuracy simulation is conducted in SC after the re-training. The two training steps, initial- and re-training, are identical to the conventional DNN training method using back-propagation algorithm [103]. There are typically many near-zero weights because weights have a tendency to become zero due to regularization as mentioned in Section 6.3. And we find that the technique to remove such near-zero weights is very effective.

In the testing phase, as shown in Figure 64, four methods are applied: 1) multiplications are done without near-zero weights to minimize random errors, 2) weights are scaled to improve signal intensity (i.e., SNR), 3) activation functions are implemented with an accumulator, and 4) approximate counters are used to reduce the size of hardware circuit.

SC can adjust the computation precision without hardware modification. This cannot be realized in other computing systems. For example, an SC system can use 32 and 1024 bits (or any bit-length) for 1/32 and 1/1024 precision, respectively, whereas 10-bit fixed-point system is fixed to 1/1024 precision only. By using this property of SC for the operation at low-precision, we can reduce energy-consumption. Fortunately, in most time, a large fraction of input data can be easily classified because many classification problems are far from decision boundaries [104]. We investigate MNIST dataset for handwritten digit images [105], as shown in Figure 64 (c), where classifying 78.1% of input data needs 1/128 precision while classifying 93.3% needs 1/256 precision. Thus, we suggest early decision termination (EDT) to finish the computation for easily classified inputs in an early stage with low precision (i.e., a relatively small number of bits).

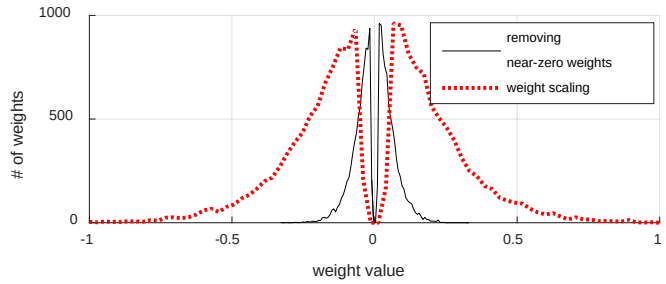


Figure 65. The distribution of weights after removing near-zero weights and

6.4.2 Removing Near-Zero Weights

Because the existence of near-zero weights is a main source of generating errors when using SC for a DNN as mentioned in Section 6.3, we propose to remove near-zero weights in the training phase. In the literature of machine learning, discarding zero weights is recently suggested in order to reduce the size of network [102]. On the other hand, we remove near-zero weights to reduce random fluctuation errors. Re-training is necessarily required because we find that the accuracy after pruning near-zero weights severely decreases and recovers after re-training. We remove near-zero weights under the threshold proportional to standard deviation of weights in a layer as follows,

$$Th_{near-zero} = std(\boldsymbol{W}^k) \cdot (\alpha + \beta \cdot i), \quad (2)$$

where $\boldsymbol{W}^k$ is all synaptic weights in a layer *k*; α and β are parameters decided empirically; *i* represents the number of iterations in Figure 64 (a). Thus, as *i* increases, more near-zero weights are removed. In our case, α and β are set to 0.2 and 0.01, respectively; re-training epoch is ten. The distribution of weights after removing near-zero weights is shown as black solid line in Figure 65.

6.4.3 Applying Weight Scaling

In order to minimize random fluctuation error and maximize SNR, we propose weight scaling technique. As shown in Figure 63 (a), error increases as the weight becomes close to zero and decreases as it becomes close to 1 or -1 while signals are changed in the opposite way. Thus, in (1), if we scale up the weights $\boldsymbol{W}^k$ before the multiplication and then scale back down the result after accumulation, the SNR can be improved. Suppose that the weights $\boldsymbol{W}^k$ are limited to a range $[-\frac{1}{s} \quad \frac{1}{s}]$ where $s > 1$, (1) can be rewritten as follows,

$$\boldsymbol{x}_j^k = af\left(\frac{1}{s} \cdot \sum_{i=0}^{n^{k-1}} s \cdot \boldsymbol{W}_{ij}^k \boldsymbol{x}_i^{k-1}\right) \tag{3}$$

Because $s \cdot \boldsymbol{W}^k > \boldsymbol{W}^k$, the signal level increases whereas the error decreases. For example, if $\boldsymbol{W}^k$ are (0.10, -0.15, 0.20), i.e., limited to [-0.2 0.2], then the weights can be scaled up five times (s=5) to become (0.50, -0.75, 1.00). The red dotted line in Figure 65 shows the advantages of the proposed weight-scaling technique. Note that the number of near-zero weights also decreases because they become far from zero center. The overhead of scaling is negligible because the scaling operation can be applied to synaptic weights in binary format only once after finishing the training phase.

The problem of this method is that it needs a scaled activation function as follows,

$$\boldsymbol{x}_j^k = af\left(\frac{1}{s}t\right), \text{ where } t = \sum_{i=0}^{n^{k-1}} s \cdot \boldsymbol{W}_{ij}^k \boldsymbol{x}_i^{k-1}. \tag{4}$$

In the next section, we suggest a scaled active function.

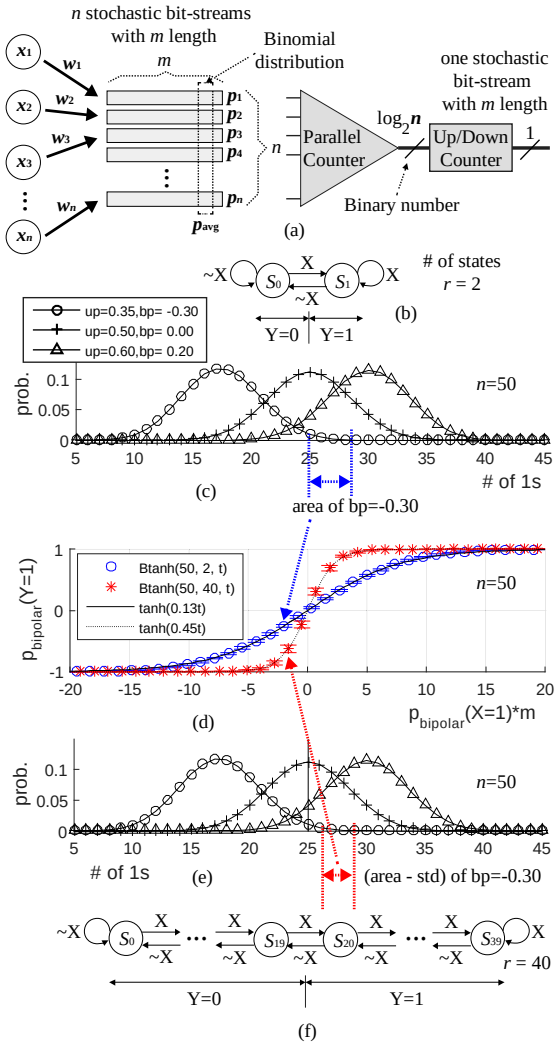


Figure 66. A stochastic neuron and the mechanism of state-machine based activate function. (a) A single neuron using SC. (b) state-machine having two states in an up/down counter. (c) Using binomial distribution for the logistic function. (d) The proposed activate function. (e) Binomial distribution with many states. (f) state-machine having 40 states.

6.4.4 Activation Function with Accumulation

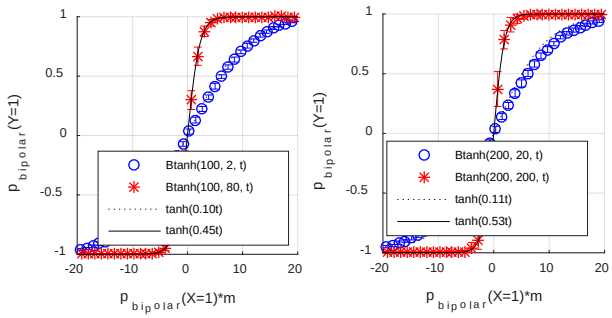
We present a state-machine based hyperbolic tangent activation function (i.e., *tanh*(·)) to solve the accumulation problem mentioned in Section 6.3 and provide the scaled functionality $tanh\left(\frac{1}{s}t\right)$ for weight-scaling. State-machine based hyperbolic tangent was introduced in [76] and [106] for a single bit stream and multiple bit streams, respectively. The hyperbolic tangent activation functions proposed in the previous work only support *tanh*(*vt*), where $v \geq 2$ and a natural number; whereas, our $tanh\left(\frac{1}{s}t\right)$ supports a small coefficient (i.e., $1/s < 1$) as well as multiple bit streams. As shown in Figure 66 (a), given *n* bit-streams with *m* bit length generated by the *n* multiplications of inputs (x_i's) and weights (w_i's), we count the number of 1s in each column by using a parallel counter. The counted value is used by the following saturated up/down counter as the amount of increase or decrease. The resulting binary value of the up/down counter is regarded as its state. Given *r* states in the up/down counter, one half of the *r* states generates 0 bit while the other half generates 1 bit; the generated bits approximate outputs of *tanh*(·) in the form of stochastic number.

6.4.4.1 Proposed Stochastic Hyperbolic Tangent

Given *n* input bit-streams with average probability p_{avg} for any bit to be 1, the probability P_{one} of having *b* 1s in a column of the input bit-streams becomes binomial distribution as follows,

$$P_{one} = \binom{n}{b}\left(p_{avg}\right)^{b}(1-p_{avg})^{n-b}. \quad (5)$$

Because cumulative distribution function (CDF) of binomial distribution mimic the logistic function, we construct *tanh*(·) by using CDF of (5). In case of two state up/down counter shown in Figure 66 (b), CDF of binomial distribution shown in Figure 5 (c) for p_{avg} = 0.00 follows *tanh*(0.13t) shown in Figure 5 (d). Note that state S_1 and S0 in Figure 5 (b) generates 1s and 0s, respectively. As the number of states



(a) (b)

Figure 67. The result comparison between the proposed hyperbolic tangent *Btanh*(·) and the original *tanh*(·). (a) The number of states is two and 80 for 100 bit-streams. (b) 20 and 200 states for 200 bit-streams.

increases as shown in (f), it becomes a bounded random walk problem [107] and the results are affected by the variation of walking. From this property, we find the relationship between $tanh\left(\frac{1}{s}t\right)$ and the proposed *Btanh*(*n*, *r*, *t*) as follows,

$$Btanh(n,r,t) \cong tanh\left(\frac{1}{s}t\right), \tag{6}$$

$$\frac{1}{s} = \frac{1-q}{2(n-1)}(r'-2n)+1, \text{ and } q = 1.835(2n)^{-0.5552} \tag{7}$$

$$r' = \frac{2(1-s)(n-1)}{s(1-q)} + 2n \tag{8}$$

$$r = nearest\_multiple\_of\_two(r') \tag{9}$$

where *n* is the number of bit-streams and *r* is the number of states and multiple of two. Figure 67 shows that both the proposed *Btanh*(·) and the corresponding *tanh*(·) are almost identical to each other. Figure 68 shows the algorithm for *Btanh*(·). In Line 1, the maximum state is set to $R-1$; the state index starts from 0 and the

BTANH (N, R, T)

// N is the number of bit-stream as shown in Figure 66 (a).

// M is the length of bit-stream.

// R is the number of states in up/down counter.

// T is the bipolar SC input, MxN size, where $-N \leq T \leq N$.

// T_i is the number of 1s bits in ith column of input streams.

Output: Y_i is the ith stochastic output bit for $Btan(\cdot)$

1: $S_{max} = R - 1$ //max state

2: $S_{half} = R/2$ //half state

3: $S \leftarrow S_{half}$ //current state

4: for i = 1 to M

5: V = Count(T_i) * 2 − N //bipolar 1s counting

6: $S \leftarrow S$ + V

7: if $S > S_{max}$ then $S \leftarrow S_{max}$

8: if $S < 0$ then $S \leftarrow 0$

9: if $S > S_{half}$

10: $Y_i \leftarrow 1$

11: else

12: $Y_i \leftarrow 0$

Figure 68. Pseudo-code for the proposed *Btanh*(·).

number of states is R. The offset V jumping between states is calculated as bipolar encoding in Line 5; it is zero if the number of 1s in a column is a half of N. Note that the SC simulation step uses $Btanh(n, r, t)$ representing $tanh\left(\frac{1}{s}t\right)$ while intial- and re-training steps in Figure 64 (a) use $tanh(a \cdot x)$, where $t = s \cdot a \cdot x$; s and a are real numbers.

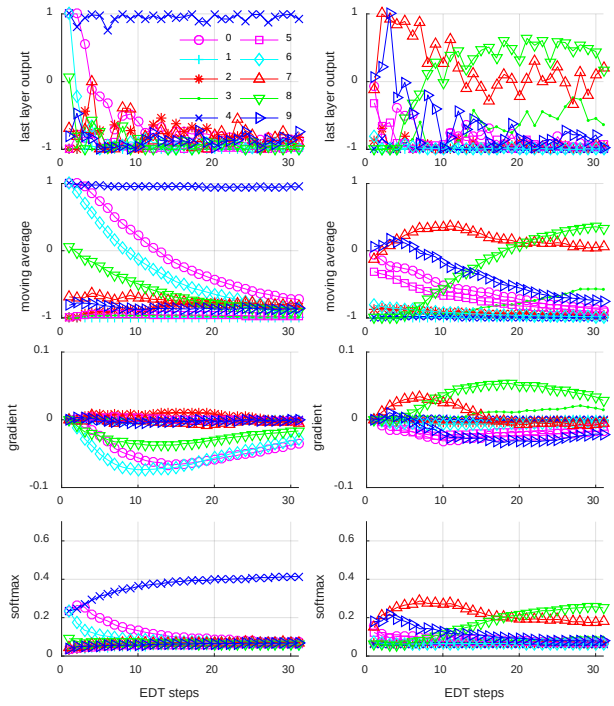
6.4.4.2 Using Approximate Counter

In the stochastic neuron shown in Figure 66 (a), the biggest component in terms

of area and power is the parallel counter. Since SC calculates in an inaccurate manner anyway, we do not need to stick to a conventional accurate parallel. [108] presents an approximate parallel counter with very small error and no bias. We reduce the area and power by using the same approach.

6.5 Early Decision Termination

Because the bits fed in at different times are independent of each other in stochastic bit-streams, the precision can be adjusted without hardware modification; it is known as progressive precision [5]. It is a salient advantage of SC over conventional logic using binary arithmetic. As mentioned in Section 6.4.1, early decision termination (EDT) is useful in terms of energy consumption and decision speed. Our current implementation has 32 bits as the precision granularity. That is, it processes 32 bits to make a decision, and if it fails, it continues processing the next 32 bits.



(a) (b)

Figure 69. The intermediate procedures of early decision termination. Ground truths are 4 and 8, respectively.

6.5.1 Moving Average Tracking Output Trends

As shown in the first row of Figure 69[28], we find that the outputs of the last layer from SC circuit severely fluctuate as EDT steps proceed, where every EDT step processes 32 stochastic bits. In order to monitor the trend of decisions as a time series, the following moving average is used as a low-pass filter,

[28] The example uses MNIST handwritten image dataset, where the number of classes is ten from zero to nine.

$$MV_{c,i} = \alpha \cdot Y_{c,i} + (1 - \alpha) \cdot MV_{c,(i-1)} \quad (10)$$

where c is the class; $Y_{c,i}$ is the output of the last layer for class c in EDT step i and $MV_{c,1}=Y_{c,1}$ and α is empirically set to 0.1. As shown in the second row of Figure 69, the result of moving average better shows the trend. We find two important components for EDT: 1) output value of the last layer and 2) the gradient of current step. Using the former is natural because the largest output value is selected in general classification domain. The latter can be an indicator to notify the possibility of changing the current decision in the future. For example, class 7 and 8 are swapped in the moving average of Figure 69 (b) (the second row), which can be informed early by examining the gradient (the third row). Thus, the objective value of individual class c in ith EDT step is defined as follows,

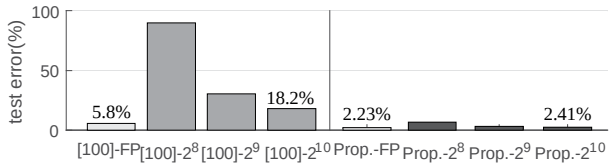
$$O_{c,i} = MV_{c,i} + \beta \cdot Grad_{c,i} \quad (11)$$

where $Grad_{c,i}$ is the gradient value of class c in ith step; β is a weight factor. Finally, by using softmax functions, the objective values of individual classes are normalized depending on the categorical probability, which is commonly used in multiclass classification problems.

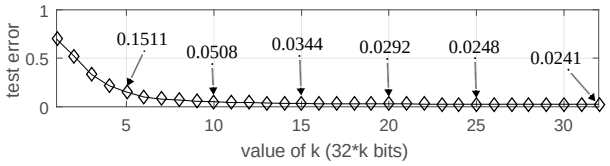
$$\mathrm{SM}_{c,i} = \frac{e^{O_{c,i}}}{\sum_k e^{O_{k,i}}} \quad (12)$$

As shown in the fourth row of Figure 69, the normalized value represents the current status of a class candidate with absolute value. For example, because class 4 dominates other classes in case of Figure 69 (a), it can be selected as the final decision in early EDT step.

6.6 Experimental Results



(a)



(b)

Figure 70. Comparison misclassification error. MNIST test data error in 32-bit floating point is 2.23%. The proposed method is 2.41% while the previous work [11] is 18.2% in 2^{10}-bit stream.

For the experiment, we use MNIST handwritten digit image dataset [105] consisting of 60000 training data and 10000 testing data with 28x28 grayscale image and 10 classes. The networks in this experiments have two hidden layers with a 784-100-200-10 configuration.

6.6.1 Accuracy of DNN Using SC

The accuracy of DNN is a very important metric, because usability of DNN totally depends on it. We compare our proposed method with the previous methods using SC [100], and a 32-bit floating-point system. As shown in Figure 70 (a), misclassification error of test data in MNIST dataset in 32-bits floating-point is 2.23%, while test error for our proposed DNN using SC is 2.41% with 2^{10}-bit stream. Considering that the previous work [100] using SC and floating-point are 18.2% and 5.8%, respectively, Our work dramatically improves in terms of accuracy. Because fixed-point arithmetic generally has bigger error compared with

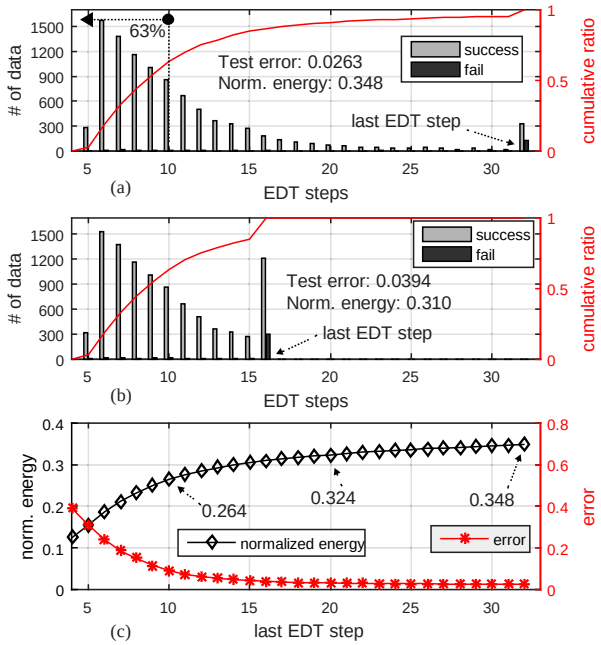


Figure 71. Experimental result for early decision termination (EDT) where one EDT step use 32 stochastic bits. (a) Applying EDT to 1024 bits. (b) The last step of EDT is set to the 16th step (i.e., 16x32=512 bits). (c) Normalized energy reduction between using and not using EDT and test error according to the last EDP step.

floating-point arithmetic and the proposed approach is almost similar to floating-point result, we conclude that the improvement is significant. Figure 70 (b) shows test error in the proposed method when using progressive precision mentioned in Section 6.5, where each step uses 32-bit parallel stochastic circuit. Note that the difference of test error between the 15th step and the 32th step is about 1% error, which means that reducing energy as well as improving decision speed is possible with sacrificing only 1% error rate.

6.6.2 Effectiveness of Early Decision Termination

Early decision termination (EDT) exploits progressive precision of SC, which can

adjust the required precision without hardware modification. In our experiment for EDT, the baseline SC circuit uses 1024 stochastic bit-stream; since one EDT step uses 32 bits, 32 EDT steps are identical to the baseline SC circuit (i.e., 32x32=1024). EDT can reduce energy and improve decision speed. Figure 71 (a) shows that the tests are finished earlier compared to the baseline SC circuit. For example, 63% among 10000 tests are finished before the 10^{th} step (i.e., 320 bits); compared with the baseline SC with 2.41% error, EDT decreases energy by 65.2% with 2.63% error as shown in (a). If we move the last EDT step to earlier ones, we can save more energy as shown in (b). By setting the last EDT step to 16, energy decreases by 69% while sacrificing accuracy by 1.53%, compared with the baseline SC circuit. (c) shows this trade-off relationship between the last EDT step, test error, and normalized energy reduction using EDT compared with not using EDT.

6.6.3 Comparison of Synthesis Results

We synthesize one SC neuron with 200 inputs, which is compared with 9-bit fixed point (FIX) because SC circuit using 512(=2^9) bits shows reasonable test error rate 0.031. Stochastic bits for each synaptic weight are generated with stochastic number generators (SNG) proposed in [109], where linear feedback shift registers (LFSRs) are shared among parallel SC circuits without generating correlation. They are implemented as combinational logic in TSMC 45nm technology library with Synopsys Design Compiler using Verilog HDL. The fixed-point is implemented with 3-stage pipelines. The recent work [2] reports that spintronic SNG using magnetic tunneling junctions (MTJs) improves energy efficiency about seven times compared with CMOS SNG. Thus, we also add the estimated value for MTJ-SNG.

Figure 72 shows the synthesis results in terms of area, critical path delay, power, and energy, where the delay of the fixed-point circuit is multiplied by three due to 3-stage pipeline. SC circuit can be implemented with a serial unit up to 512 parallel units, and we select 32- and 64- parallel SC circuit for area, critical path delay and power investigation. Note that the area and power increase in proportion to the

parallelism while critical path delay does not vary. As a result, we find that SNG overhead is very significant; it takes 41.50%, 59.58%, and 75.76% of SC circuit (SC w/ SNG) in terms of area, power, and critical path delay. However, regardless of the parallelism of SC circuit, energy consumption is identical in all cases. Compared with 9-bit width fixed-point, SC with SNG increases energy by 3.0 times while SC without SNG decreases energy by 70.0%; we also estimate SNG with MTJ-SNG decreasing energy by 30.0% from the result of [2]. Due to EDT, energy decrease by about 34% compared to basic SC in all cases. Figure 73 shows iso-area performance where all circuits are set to the area of 9-bit fixed-point which is 72104 um^2. In case of latency with EDT[29], SC without SNG is 4.61 times faster while SC with SNG is 1.53 times slower compared with 9-bit fixed-point. It is because the two cases have 120x and 70x parallelisms, respectively, under iso-area and the critical path delay of the former is 4.125 shorter than that of the latter.

[29] The latency is the time to calculate a 9-bit binary number in a fixed-point arithmetic and to calculate a 512-bit stream in SC.

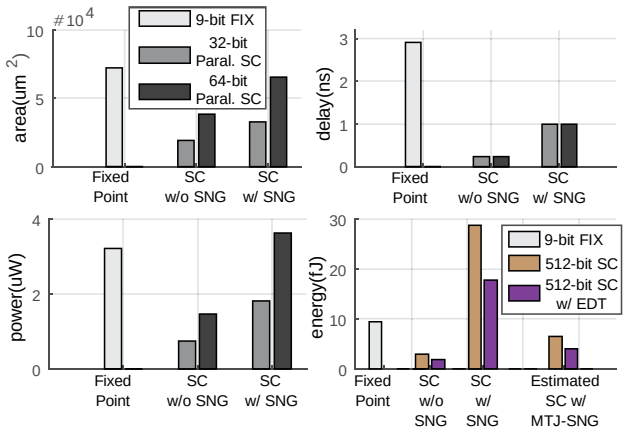


Figure 72. Synthesis results. All cases are compared with 9-bit fixed-point (9-bit FIX). In case of area, critical path delay, and power, 32- and 64-bit parallel SC circuits are used. In case of energy, SC circuit executes 2^9(=512) bits.

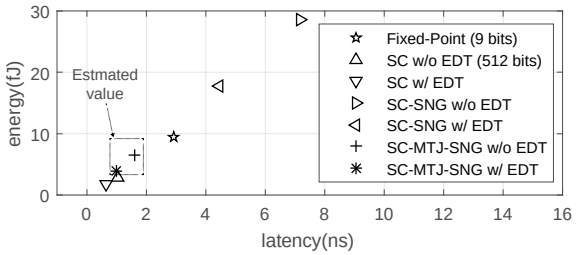


Figure 73. Iso-area performance comparison. Energy and latency for each case are compared under same area. The values for SC circuits using MTJ-SNG are estimated according to [2]. Fixed-point computes with 9-bit width while all SC circuits compute

6.7 Conclusion

In this paper, we address the problems in directly adopting stochastic computing to DNN by removing near-zero weights, applying weight-scaling, and using state machine based activation function integrated with the accumulator. We also suggest the early decision termination which is very useful in terms of energy and decision speed. The experimental results demonstrate that the accuracy of DNN using SC is

close to that of the conventional floating-point system while reducing the area, power, critical path delay, and energy.

7. Conclusion

Since machine learning is a very promising domain to assist humans with intelligence, it has been paid attention in many various application areas. We present schemes for machine learning from high-level algorithms down to low-level hardware building blocks, which include hierarchical ensemble learning framework and stochastic computing logic synthesis. They are combined in the domain of deep learning. The result shows that the synergistic effect of them will lead to a very efficient machine learning system.

It is well known that ensemble of classifiers can achieve higher accuracy compared to a single classifier system. This paper pays attention to ensemble systems consisting of multiple feature extractors and multiple classifiers (MFMC). However, MFMC increases the system complexity dramatically, leading to a highly complex combinatorial optimization problem. In order to overcome the complexity while exploiting the diversity of MFMC, we suggest in this paper a hierarchical ensemble of MFMC and its optimizing framework. By constructing local groups of feature extractors and classifiers and then combining them as a global group, the approach achieves a better scalability. Both reinforcement machine learning and Bayesian networks are adopted to enhance the accuracy. We apply the proposed method to vision based pedestrian detection and recognition of handwritten numerals. Experimental results show that the proposed framework outperforms the previous ensemble methods in terms of accuracy.

Stochastic Computing (SC) is a very promising technique to boost logic efficiency in terms of area, power, and error tolerance when the accuracy of computation can be relaxed. One of the challenges with SC, however, is how to find optimal SC operations from conventional expressions based on binary arithmetic. This work presents a novel approach to automatically synthesizing a network of SC

operations from a set of given conventional arithmetic expressions possibly including many variables. It first generates building blocks called iSC kernels from the given conventional expression and then synthesizes an SC logic network by using the relationship between iSC kernels. Experimental results obtained by applying our proposed algorithm to a set of applications demonstrate that our technique generates SC logic that outperforms the conventional binary logic in terms of area, critical path delay, and power consumption.

Deep neural networks (DNNs) have been recently paid great attention because they achieve a noticeable performance improvement in term of accuracy in supervised learning over other machine learning techniques. However, DNNs consisting of many neurons require lots of computations leading to considerable power consumption, which is an obstacle to the wide usage of DNNs in embedded systems or mobile devices. This paper presents a method of implementing a DNN using stochastic computing, where multiplications-the majority operations-can be implemented with a single XNOR gate in bipolar format. Based on the observation that directly adopting stochastic computing to DNN has some challenges such as random error fluctuation, range limitation from -1 to 1, and overhead in accumulating many products of inputs and synaptic weights, we address these problems by removing near-zero weights, applying weight-scaling, and using state machine based activation function integrated with the accumulator. The approach allows an easy implementation of early decision termination without hardware modification for a given classification problem by efficiently exploiting the progressive precision characteristics of stochastic computing, which was not easy with existing approaches. We find that the early decision termination is very useful in terms of energy and decision speed because most of test inputs are far from decision boundary. Our experimental results demonstrate that our technique outperforms the conventional binary fixed logic in terms of gate area, latency, and power consumption.

Bibliography

[1] H. Ichihara, *et al.*, "Compact and accurate stochastic circuits with shared random number sources," Proc. ICCD, pp. 361-366, 2014.

[2] R. Venkatesan, *et al.*, "Spintastic: spin-based stochastic logic for energy-efficient computing," Proc. Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 1575-1578, 2015.

[3] R. Polikar, "Ensemble based systems in decision making," Circuits and Systems Magazine, IEEE, vol. 6, pp. 21-45, 2006.

[4] L. Rokach, "Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography," Computational Statistics & Data Analysis, vol. 53, pp. 4046-4072, 2009.

[5] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," ACM Trans. Embed. Comput. Syst., vol. 12, p. 92, 2013.

[6] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," Proc. DATE, p. 76, 2014.

[7] D. H. Wolpert, "The supervised learning no-free-lunch theorems," in *Soft Computing and Industry*, ed: Springer, 2002, pp. 25-42.

[8] N. C. Oza and K. Tumer, "Classifier ensembles: Select real-world applications," Information Fusion, vol. 9, pp. 4-20, 2008.

[9] L. I. Kuncheva, *Combining Pattern Classifiers, Methods and Algorithms*: Wiley-Interscience, 2004.

[10] L. Rokach, *Pattern classification using ensemble methods* vol. 75: World Scientific, 2010.

[11] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning

and an application to boosting," in *Computational learning theory*. vol. 904, ed: Springer Berlin Heidelberg, 1995, pp. 23-37.

[12] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," IEEE Transactions on Neural Networks, vol. 22, pp. 1517-1531, 2011.

[13] H. Lin, K. Kim, and K. Choi, "Concept-Aware Ensemble System for Pedestrian Detection," Proc. In Proceedings of IEEE Symposium on Intelligent Vehicles, 2014.

[14] O. Ludwig, D. Delgado, V. Goncalves, and U. Nunes, "Trainable classifier-fusion schemes: An application to pedestrian detection," Proc. In Proceedings of International IEEE Conference on Intelligent Transportation Systems, pp. 1-6, 2009.

[15] L. Oliveira, U. Nunes, and P. Peixoto, "On Exploration of Classifier Ensemble Synergism in Pedestrian Detection," IEEE Transactions on Intelligent Transportation Systems, vol. 11, pp. 16-27, 2010.

[16] S. Walk, K. Schindler, and B. Schiele, "Disparity statistics for pedestrian detection: Combining appearance, motion and stereo," Proc. In Proceedings of European Conference on Computer Vision, pp. 182-195, 2010.

[17] Y. Xu, X. Cao, and H. Qiao, "An efficient tree classifier ensemble-based approach for pedestrian detection," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 41, pp. 107-117, 2011.

[18] O. Tuzel, F. Porikli, and P. Meer, "Pedestrian detection via classification on riemannian manifolds," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 30, pp. 1713-1727, 2008.

[19] C. Wojek, S. Walk, and B. Schiele, "Multi-cue onboard pedestrian detection," Proc. Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pp. 794-801, 2009.

[20] H. Yoshida, *et al.*, "Integration of Generative Learning and Multiple Pose Classifiers for Pedestrian Detection," Proc. VISAPP (1), pp. 567-572, 2012.

[21] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, pp. 1239-1258, 2010.

[22] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: An Evaluation of the State of the Art," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, pp. 743-761, 2012.

[23] J. Yan, *et al.*, "Robust Multi-resolution Pedestrian Detection in Traffic Scenes," Proc. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3033-3040, 2013.

[24] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. Lecun, "Pedestrian Detection with Unsupervised Multi-stage Feature Learning," Proc. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3626-3633, 2013.

[25] C. G. Keller, *et al.*, "The Benefits of Dense Stereo for Pedestrian Detection," IEEE Transactions on Intelligent Transportation Systems, vol. 12, pp. 1096-1106, 2011.

[26] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New features and insights for pedestrian detection," Proc. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1030-1037, 2010.

[27] M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata, "Pedestrian detection with convolutional neural networks," Proc. In Proceedings of IEEE Intelligent Vehicles Symposium, pp. 224-229, 2005.

[28] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," Proc. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 886-893, 2005.

[29] R. Xia, C. Zong, X. Hu, and E. Cambria, "Feature ensemble plus sample selection: domain adaptation for sentiment classification," Intelligent Systems, IEEE, vol. 28, pp. 10-18, 2013.

[30] Y. S. Huang and C. Y. Suen, "A method of combining multiple experts for the recognition of unconstrained handwritten numerals," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 17, pp. 90-94, 1995.

[31] K.-D. Wernecke, "A coupling procedure for the discrimination of mixed data," Biometrics, pp. 497-506, 1992.

[32] C. J. Merz, "Using correspondence analysis to combine classifiers," Machine Learning, vol. 36, pp. 33-58, 1999.

[33] F. Moreno-Seco, J. M. Inesta, P. J. P. De León, and L. Micó, "Comparison of classifier fusion methods for classification in pattern recognition tasks," in *Structural, Syntactic, and Statistical Pattern Recognition*. vol. 4109, ed: Springer Berlin Heidelberg, 2006, pp. 705-713.

[34] M. Smętek and B. Trawiński, "Selection of heterogeneous fuzzy model ensembles using self-adaptive genetic algorithms," New Generation Computing, vol. 29, pp. 309-327, 2011.

[35] M. Wozniak, "Evolutionary approach to produce classifier ensemble based on weighted voting," Proc. Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, pp. 648-653, 2009.

[36] M. Wozniak and M. Zmyslony, "Combining classifiers using trained fuser—Analytical and experimental results," Neural Network World, vol. 20, p. 925,

2010.

[37] T. Wilk and M. Wozniak, "Soft computing methods applied to combination of one-class classifiers," Neurocomputing, vol. 75, pp. 185-193, 2012.

[38] L. A. Alexandre, A. C. Campilho, and M. Kamel, "Combining independent and unbiased classifiers using weighted average," Proc. Pattern Recognition, 2000. Proceedings. 15th International Conference on, pp. 495-498, 2000.

[39] B. Biggio, G. Fumera, and F. Roli, "Bayesian analysis of linear combiners," in *Multiple Classifier Systems*, ed: Springer, 2007, pp. 292-301.

[40] K. Chen, L. Wang, and H. Chi, "Methods of combining multiple classifiers with different features and their applications to text-independent speaker identification," International Journal of Pattern Recognition and Artificial Intelligence, vol. 11, pp. 417-445, 1997.

[41] G. Zenobi and P. Cunningham, "Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error," in *Machine Learning: ECML 2001*, ed: Springer, 2001, pp. 576-587.

[42] L. Breiman, "Bagging predictors," Machine learning, vol. 24, pp. 123-140, 1996.

[43] R. E. Schapire, "The strength of weak learnability," Machine learning, vol. 5, pp. 197-227, 1990.

[44] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," Journal of artificial intelligence research, pp. 263-286, 1995.

[45] G. Zhong and C.-L. Liu, "Error-correcting output codes based ensemble feature extraction," Pattern Recognition, vol. 46, pp. 1091-1100, 4// 2013.

[46] G. Zhong and M. Cheriet, "Adaptive error-correcting output codes," Proc. Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, pp. 1932-1938, 2013.

[47] J. Wu and J. M. Rehg, "CENTRIST: A visual descriptor for scene categorization," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, pp. 1489-1501, 2011.

[48] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proc. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. I-511-I-518 vol. 1, 2001.

[49] C. M. Bishop, *Pattern Recognition and Machine Learning*: Springer, 2007.

[50] R. S. Sutton and A. G. Barto, *Reinforcement Learning, An Introduction*: MIT Press,

1999.

[51] A. Frank and A. Asuncion. *UCI Machine Learning Repository, 2010*. Available: http://archive.ics.uci.edu/ml

[52] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *Abstraction, Reformulation and Approximation*. vol. 3607, ed: Springer Berlin Heidelberg, 2005, pp. 194-205.

[53] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," Machine learning, vol. 29, pp. 131-163, 1997.

[54] A. Mittal and A. A. Kassim, *Bayesian network technologies: applications and graphical models*: IGI Global, 2007.

[55] R. E. Neapolitan, *Learning bayesian networks* vol. 1: Prentice Hall, 2004.

[56] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," Foundations and Trends in Machine Learning, vol. 1, pp. 1-305, 2008.

[57] B. Lerner and R. Malka, "Investigation of the K2 algorithm in learning Bayesian network classifiers," Applied Artificial Intelligence, vol. 25, pp. 74-96, 2011.

[58] N. Friedman and D. Koller, "Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks," Machine learning, vol. 50, pp. 95-125, 2003.

[59] F. V. Jensen and T. D. Nielsen, *Bayesian networks and decision graphs*, second ed.: Springer, 2007.

[60] A. Marchetti-Spaccamela, U. Nanni, and H. Rohnert, "Maintaining a topological order under edge insertions," Information Processing Letters, vol. 59, pp. 53-58, 1996.

[61] B. Haeupler, *et al.*, "Faster Algorithms for Incremental Topological Ordering," in *Automata, Languages and Programming*. vol. 5125, ed: Springer Berlin Heidelberg, 2008, pp. 421-433.

[62] S. Munder and D. M. Gavrila, "An Experimental Study on Pedestrian Classification," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, pp. 1863-1868, 2006.

[63] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," Proc. Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, pp. 1491-1498, 2006.

[64] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *Computer Vision–ECCV 2006*, ed: Springer, 2006, pp.

428-441.

[65] X. Wang, T. X. Han, and S. Yan, "An HOG-LBP human detector with partial occlusion handling," Proc. Computer Vision, 2009 IEEE 12th International Conference on, pp. 32-39, 2009.

[66] J. Wu, C. Geyer, and J. M. Rehg, "Real-time human detection using contour cues," Proc. Robotics and Automation (ICRA), 2011 IEEE International Conference on, pp. 860-867, 2011.

[67] T. Yang, *et al.*, "Condensation-based multi-person tracking using an online SVM approach," Proc. Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on, pp. 1983-1988, 2013.

[68] Y.-S. Hwang, J.-C. Kwak, and K.-Y. Lee, "Implementation of a Pedestrian Detection Device based on CENTRIST for an Embedded Environment," 2014.

[69] X. Cui, *et al.*, "3D Haar-Like Features for Pedestrian Detection," Proc. ICME, pp. 1263-1266, 2007.

[70] V.-D. Hoang, A. Vavilin, and K.-H. Jo, "Fast human detection based on parallelogram haar-like features," Proc. IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society, pp. 4220-4225, 2012.

[71] P. Geismann and G. Schneider, "A two-staged approach to vision-based pedestrian recognition using Haar and HOG features," Proc. Intelligent Vehicles Symposium, 2008 IEEE, pp. 554-559, 2008.

[72] OpenCV. *Open source computer vision (OpenCV)*. Available: http://opencv.org

[73] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," Machine learning, vol. 37, pp. 297-336, 1999.

[74] S. Gupta, R. K. Gupta, N. D. Dutt, and A. Nicolau, *SPARK: a parallelizing approach to the high-level synthesis of digital circuits*: Springer, 2004.

[75] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," Proc. DAC, p. 113, 2013.

[76] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," IEEE Trans. Comput., vol. 50, pp. 891-905, 2001.

[77] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," Electron. Lett., vol. 39, pp. 299-301, 2003.

[78] P. Li, *et al.*, "Case Studies of Logical Computation on Stochastic Bit Streams," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and*

Simulation, ed: Springer, 2013, pp. 235-244.

[79] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," Proc. DAC, pp. 1-6, 2013.

[80] W. Qian, *et al.*, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," IEEE Trans. Comput., vol. 60, pp. 93-105, 2011.

[81] V. K. Chippa, S. Venkataramani, K. Roy, and A. Raghunathan, "StoRM: a stochastic recognition and mining processor," Proc. ISLPED, pp. 39-44, 2014.

[82] A. Singhee and R. A. Rutenbar, *Novel algorithms for fast statistical analysis of scaled circuits* vol. 46: Springer Science & Business Media, 2009.

[83] J. Hartmann and G. Kemnitz, "How to do weighted random testing for BIST?," Proc. ICCAD, pp. 568-568, 1993.

[84] F. Muradali, V. K. Agarwal, and B. Nadeau-Dostie, "A new procedure for weighted random built-in self-test," Proc. ITC, pp. 660-669, 1990.

[85] P. Gupta and R. Kumaresan, "Binary multiplication with PN sequences," IEEE T. Acoust. Speech., vol. 36, pp. 603-606, 1988.

[86] A. Alaghi and J. P. Hayes, "A spectral transform approach to stochastic circuits," Proc. ICCD, pp. 315-321, 2012.

[87] H. Beker and F. Piper, *Cipher systems: the protection of communications*: Wiley-Interscience, 1982.

[88] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," Proc. ICCD, pp. 39-46, 2013.

[89] B. Parhami and C.-H. Yeh, "Accumulative parallel counters," Proc. Asilomar Conf. Signals, Systems & Computers, pp. 966-970, 1995.

[90] T. Pai-Shun and J. P. Hayes, "Stochastic Logic Realization of Matrix Operations," Proc. Digital System Design, pp. 356-364, 2014.

[91] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Proc. NIPS, pp. 1097-1105, 2012.

[92] G. Hinton, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," IEEE Signal Process. Mag., vol. 29, pp. 82-97, 2012.

[93] C. Szegedy, *et al.*, "Going deeper with convolutions," arXiv preprint arXiv:1409.4842, 2014.

[94] A. Coates, *et al.*, "Deep learning with COTS HPC systems," Proc. ICML, pp.

1337-1345, 2013.

[95] M. Montemerlo, *et al.*, "Junior: The stanford entry in the urban challenge," J. Field Robot., vol. 25, pp. 569-597, 2008.

[96] K. H. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," IEEE J. Solid-State Circuit, vol. 48, pp. 1625-1637, 2013.

[97] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," Proc. workshop contribution at ICLR, 2015.

[98] H. Geoffrey, V. Oriol, and D. Jeff, "Distilling the knowledge in a neural network," Proc. NIPS workshop, 2014.

[99] B. D. Brown and H. C. Card, "Stochastic neural computation. II. Soft competitive learning," IEEE Trans. Comput., vol. 50, pp. 906-920, 2001.

[100] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "FPGA implementation of a Deep Belief Network architecture for character recognition using stochastic computation," Proc. CISS, pp. 1-5, 2015.

[101] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, ed: Springer, 2012, pp. 9-48.

[102] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," Proc. NIPS, 2015.

[103] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85-117, 2015.

[104] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," Proc. DAC, p. 67, 2015.

[105] L. Deng, "The MNIST database of handwritten digit images for machine learning research," IEEE Signal Process. Mag., vol. 29, pp. 141-142, 2012.

[106] A. Ardakani, *et al.*, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing," arXiv preprint arXiv:1509.08972, 2015.

[107] O. C. Ibe, *Elements of Random Walk and Diffusion Processes*: John Wiley & Sons, 2013.

[108] K. Kim, J. Lee, and K. Choi, "Approximate De-randomizer for Stochastic Circuits," Proc. ISOCC, 2015.

[109] K. Kim, J. Lee, and K. Choi, "An Energy-Efficient Random Number Generator for Stochastic Circuits," Proc. ASP-DAC, 2016.

요약(국문초록)

머신러닝(machine learning)은 인지컴퓨팅의 분야로 최근에 산업, 의학, 교통, 엔터테인먼트 등 사람과 상호작용이 필요한 많은 분야에서 뛰어난 성능을 입증 받아 주목을 받고 있다. 효율적인 머신러닝 시스템을 구현하기 위하여 본 논문은 상위레벨의 알고리즘으로 부터 하위레벨의 하드웨어 회로에 까지 다양한 계층의 연구를 포함하고 있다.

상위레벨의 알고리즘 연구는 혼성모델(ensemble model)을 활용한 학습방법에 대한 것이다. 혼성모델은 다수의 특징 추출기(feature extractor)와 분류기(classifier)를 동시에 사용하는 것으로, 이러한 요소들을 결합하여 단일 모델에서 발생하는 문제점과 파라미터 최적화 문제점을 해결하여 정확도 측면의 향상시킨다. 본 연구에서 다수의 특징 추출기와 분류기를 사용하는 계층적인 혼성 프레임웍(hierarchical ensemble learning framework)을 제안하여 정확도 측면에 향상을 보여 주었다.

머신러닝 시스템은 일반적으로 부정확한 계산을 허용하고 데이터의 중복성이 존재한다고 알려져 있다. 머신러닝 효율성을 극대화하기 위하여, 이러한 성질을 효율적으로 활용할 수 있는 확률 컴퓨팅(stochastic computing)을 도입하여, 회로 면적 및 에너지 효율을 높일 수 있는 방법을 제안하였다. 확률 컴퓨팅은 기존에 고정소수점 연산에 비하여 정확도를 희생하면서 면적, 속도, 에너지 등을 향상시킬 수 있는 컴퓨팅 방법이다. 이것을 활용하여 임의의 연산에 대한 회로를 합성할 수 있는 방법을 제안하였다.

최근 머신러닝 분야에서 훌륭한 성능으로 주목을 받고 있는 분야가 딥러닝 (deep learning)이다. 딥러닝은 많은 뉴런(neuron)과 레이어(layer)를 활용하여 기존의 머신러닝 시스템의 정확도 문제점을 해결하고 있다. 본 연구는 기존 딥러닝에 확률 컴퓨팅을 접목하여 향상을 꾀하였다. 하지만 기

존의 확률 컴퓨팅을 딥러닝에 그대로 적용하는 데는 랜덤 에러, 수치의 제한, 가산기의 문제 등이 존재한다. 본 연구에서는 이러한 문제를 해결할 수 있는 방법을 제안하여 딥러닝 효율성을 높였다. 또한 대부분의 데이터들이 결정경계(decision boundary)에서 멀리 떨어져 있는 것을 이용하여 이른 결정 종료(early decision termination) 방법을 제안하였다. 이를 기반으로 합성회로의 면적, 속도, 파워 등의 향상을 실험 결과로 보여주었다.

주요어 : 머신러닝, 기계학습, 확률컴퓨팅, 혼성 학습, 딥러닝, 딥신경망

학 번 : 2012-30193