Microcanonical Optimization을 이용한 BDD의 최소화 기법

이 민 나[†] · 조 상 영[††]

요 약

BDD를 이용하면, 부울 함수를 간결하고 유일하게 나타낼 수 있기 때문에, 논리 합성, 형식 검증 등 컴퓨터 지원 설계 분야에 널리 활용되고 있다. BDD의 크기는 입력 변수 순서에 따라 민감하게 변하므로, BDD의 크기를 최소화할 수 있는 변수 순서를 구하는 것은 매우 중요한 문제이다. 그러나 최적의 변수 순서를 찾는 것은 NP-complete 문제이므로, 근사 최적 변수 순서(near-optimal variable ordering)를 결정하기 위한 여러 가지 휴리스틱 알고리즘이 제안되었다. 본 논문에서는 μO(Microcanonical Optimization) 알고리즘에 기반한 BDD의 입력 변수 순서화 알고리즘을 제안한다. μO 알고리즘은 외부 환경과 독립된 시스템의 안정된 상태를 찾아내는 알고리즘으로 국부 탐색을 하는 초기화 단계와 국부 최적 해를 벗어나기 위한 샘플링 단계로 구성되어 반복 실행하는 알고리즘이다. 제안된 알고리즘은 많은 벤치마크 회로에 대하여 실험되었으며 SA(Simulated Annealing) 알고리즘 보다 더 좋은 성능을 보인다.

A Minimization Technique for BDD based on Microcanonical Optimization

Min-Na Lee[†] · Sang-Young Cho[††]

ABSTRACT

Using BDD, we can represent Boolean functions uniquely and compactly. Hence, BDD have become widely used for CAD applications, such as logic synthesis, formal verification, and etc. The size of the BDD representation for a function is very sensitive to the choice of orderings on the input variables. Therefore, it is very important to find a good variable ordering which minimize the size of the BDD. Since finding an optimal ordering is NP-complete, several heuristic algorithms have been proposed to find good variable orderings. In this paper, we propose a variable ordering algorithm based on the μO (microcanonical optimization). μO consists of two distinct procedures that are alternately applied : Initialization and Sampling. The initialization phase is to executes a fast local search, the sampling phase leaves the local optimum obtained in the previous initialization while remaining close to that area of search space. The proposed algorithm has been experimented on well known benchmark circuits and shows superior performance compared to a algorithm based on simulated annealing.

키워드 : 이진결정도(BDD), 순서 이진결정도(OBDD), 축약된 순서 이진결정도(ROBDD), 시뮬레이티드 어닐링(simulated annealing), 작은 바른틀 최적화(microcanonical optimization), variable orderings

1. 서 론

이진결정도(BDD : Binary Decision Diagrams)는 부울 함수의 표현과 조작에 효과적인 자료구조로서, 비순환 방향 그래프(DAG : Directed Acyclic Graph)로 표현된다[1-4]. 일반적인 BDD의 변수 순서에 제약조건을 준 "순서 BDD(OBDD : Ordered Binary Decision)"와 이의 축약된 형태인 "축약된 순서 BDD(ROBDD : Reduced Ordered Binary Decision Diagrams)"가 소개되면서 ROBDD의 유일한 표현(canonical form)을 갖는 특성과 효과적인 부울 함수의 조작 기술로 인하여 BDD는 컴퓨터 지원 설계(CAD : Computer Aided Design) 분야에 널리 활용되기 시작했다[3]. BDD의 주요 응용분야로는 조합 회로 검증, 테스트 패턴 생성, 심볼릭 시뮬레이션, 논리 합성(logic synthesis), 형식 검증(formal verification), 그리고 순차 회로 동일 검사(equivalence test) 등으로 BDD는 CAD 전 분야에 걸쳐 매우 광범위하게 사용되고있다[18, 19].

이러한 BDD는 그 크기가 최악의 경우, 입력 변수 개수에 비례하여 지수적으로 증가한다. 따라서 입력 변수의 개수가 많은 경우에 BDD의 크기는 컴퓨터로 다룰 수 없을 만큼 매우 커질 수 있다. BDD의 효율적인 조작을 위해서는 BDD의 크기가 작아야 하는데 그 크기는 입력 변수 순서에 직접적

† 준 회 원 : 한국외국어대학교 대학원 컴퓨터공학과
†† 종신회원 : 한국외국어대학교 컴퓨터공학과 교수
논문접수 : 2000년 11월 13일, 심사완료 : 2001년 4월 18일

으로 의존하기 때문에 적절한 변수 순서 선택에 의한 BDD 크기 최적화는 논리 합성과 형식 검증 등의 분야뿐만 아니라 모든 BDD 응용 분야에서도 매우 기본적이고도 중요한 문제이다[6].

입력 변수의 개수가 n개인 BDD에서 변수 순서의 가지 수는 $n!$이며, 이중 BDD 크기가 최소인 최적의 변수 순서를 발견하는 문제는 NP-complete 문제이다[24]. 따라서 지금까지 입력 변수 순서를 결정하는 다양한 휴리스틱 알고리즘들이 제안되었다.

현재까지 제안된 알고리즘 중 가장 널리 알려진 BDD 변수 순서 알고리즘으로 시프팅(Sifting) 알고리즘[7]이 있다. 이 알고리즘은 각각의 변수가 자신의 최적 위치를 찾기 위해 인접 변수와 위치를 교환하며 모든 위치를 순회하는 방법으로 그중 크기가 가장 작은 BDD 변수 순서를 찾아내는 알고리즘이다. 시프팅 알고리즘을 개선시킨 알고리즘으로 확장 시프팅(extension sifting) 알고리즘[8, 9]이 있으며, 이 알고리즘은 각각의 변수를 하나씩 이동시키는 것이 아니고 논리적으로 인접해야만 최소의 크기를 갖는 대칭성 변수(symmetry variable)를 하나의 그룹으로 묶어 그룹형태로 이동시키는 알고리즘이다. 또한 생물의 진화과정과 유전 법칙을 기반으로 한 유전자(Genetic) 알고리즘[11, 13, 14]과, 고체 물리학에서 에너지 수준이 가장 낮은 상태의 결정을 얻기 위해 어닐링 과정을 기반으로 한 SA(Simulated Annealing) 알고리즘[10] 등과 같은 확률에 기반을 둔 휴리스틱 알고리즘을 이용하여 BDD 변수 순서를 결정하는 알고리즘들이 제시되었다.

현재까지 제안된 알고리즘 중에서 SA 알고리즘이 가장 좋은 성능을 보이고 있다. 그러나 SA 알고리즘은 실행 시간이 매우 오래 걸리는 단점을 가지고 있다. 본 논문에서는 SA 알고리즘에 비교하여 비슷한 BDD 크기의 성능을 가지면서도 실행 시간을 단축할 수 있는 μO(Microcanonical Optimization) 알고리즘[20]을 기반으로 한 입력 변수 순서 최적화 알고리즘을 제안한다.

μO 알고리즘은 초기화 단계와 샘플링 단계를 반복 실행하는 알고리즘으로, 초기화 단계는 국부 최적 해를 찾는 단계로 향상된 해만을 받아들이는 단계이며, 샘플링 단계는 전체 최적 해를 찾기 위하여 초기화 단계에서 찾은 국부 최적 해를 벗어나기 위한 탐색과정으로 총 에너지가 같은 일정한 탐색 범위 내에서 나쁜 결과의 해도 받아들이는 단계이다. 이와 같은 초기화 단계와 샘플링 단계를 종료 조건이 만족할 때까지 반복 실행한다. μO 알고리즘은 TSP(traveling Salesman Problem) 문제[23], Linear Gate Assignment 문제[21], Task Scheduling 문제[22] 등에 적용되어 그 효율성이 증명되었다. 본 논문은 잘 알려진 IWLS91 벤치마크 데이터의 실험을 통하여 제안된 알고리즘이 SA 알고리즘에 비해 우수함을 보여준다.

본 논문의 구성은 다음과 같다. 제2장에서는 BDD의 일반적인 개념에 대하여 설명한다. 제3장에서는 μO 알고리즘의 개념을 설명하고, 제4장에서는 μO 알고리즘에 기반을 둔 BDD의 변수 순서 최적화 알고리즘을 제안한다. 제5장에서는 실험결과를 통해 제안된 알고리즘의 성능 및 효율성을 보이고, 제6장에서 결론을 맺는다.

2. BDD의 기본개념

2.1 BDD의 정의

BDD는 부울 함수를 비순환 방향 그래프로 나타낸 것을 말한다. Akers는 디지털 함수를 정의, 분석, 테스트하고, 구현하는 방법을 위해 BDD를 정의하였다[1]. BDD는 형식적 표현기법으로 부울 함수를 if-then-else 형태로 표현하고 조작하는 자료구조이다. n개의 입력변수, $Xn = \{x_1, x_2, \cdots x_n\}$에 대한 BDD는 루트 노드(root node)가 존재하는 DAG 형태로서 노드(node)와 에지(edge)로 구성되는 그래프 $G = (V, E)$의 형태로 표현된다. 노드는 비단말 노드(nonterminal node)와 단말 노드(terminal node) 두 가지 형태로 구분된다. 입력변수를 나타내는 비단말 노드(v)는 두 개의 후손 노드인 $low(v)$, $high(v) \in V$를 가진다. 이것은 각각 수식 (1)처럼 샤논 확장(Shannon expansion) 형태로서 변수 x_i에 관한 함수 f로 표현하게 된다.

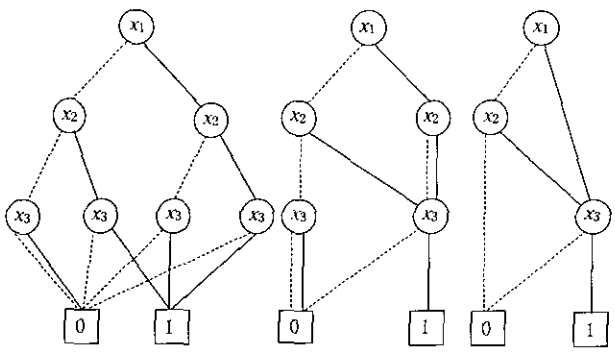
$$\begin{aligned} f &= \bar{x}_i \cdot f|_{x_i=0} + x_i \cdot f|_{x_i=1} \\ &= \bar{x}_i \cdot f(x_0, x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n) \\ &\quad + x_i \cdot f(x_0, x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) \end{aligned} \quad (1)$$

부울 함수의 논리 값을 나타내는 단말 노드는 0과 1을 나타내는 두 개의 노드로 구성되며 후손 노드를 가지지 않는다.

Bryant는 BDD의 변수 순서에 제약조건을 추가한 "순서 BDD(OBDD : Ordered BDD)"를 제안하였다[2, 3]. OBDD란 그래프의 모든 경로 상의 입력 변수 순서가 고정되어 있고, 변수는 각 경로(path)당 한번만 나타나는 BDD를 의미하는 것으로 부울 함수에 대한 연산은 OBDD상의 그래프 알고리즘으로 표현 할 수 있다. 또한 BDD의 유일한 표현 형태인 "축약된 순서 BDD(ROBDD : Reduced Ordered BDD)"는 동일한 형태의 부-그래프(isomorphic sub-graph)가 존재하지 않고, 두 개의 후손이 동일한 노드가 아닌 형태의 OBDD를 말한다[2, 3, 15]. ROBDD는 하나의 부울 함수에 대해 변수순서가 일정할 때 그 표현 형태가 유일하므로 형식 검증에 많이 사용된다. 본 논문에서의 BDD란 ROBDD를 의미한다.

(그림 1)은 $f = x_1x_3 + \bar{x}_1x_2x_3$ 함수의 OBDD를 ROBDD로 만들이 가는 과정을 나타낸 그림이다. (그림 1)의 (a)는 모든 경로에 각 변수의 순서가 일정하고, 각 경로에 각각의 변수가 한번씩만 나타나므로 OBDD이다. (그림 1)의 (b)는 (그림 1)의 (a)에서 동일한 부-그래프를 삭제하여 전체 그래프에 동일한 부-그래프가 존재하지 않도록 한 결과이며,

(그림 1)의 (c)는 (그림 1)의 (b)에서 하나의 동일한 후손 노드만을 갖는 노드를 삭제하여 ROBDD를 생성한다.

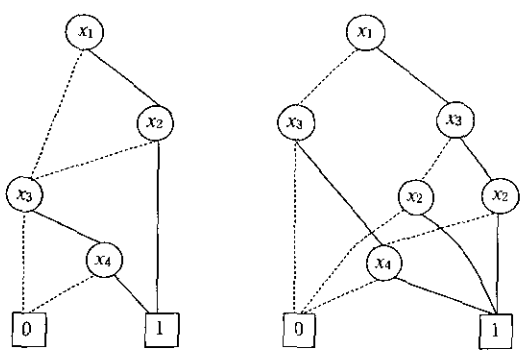


(a) OBDD (b) 동일 부-그래프 삭제 (c) 동일 후손 노드를 갖는 노드 삭제

(그림 1) $f = x_1x_3 + \bar{x}_1x_2x_3$ 함수의 OBDD의 축약과정

2.2 BDD의 입력 변수 순서

BDD의 크기는 입력 변수 순서에 직접적으로 의존하므로 적절한 입력 변수 순서의 선택은 매우 중요한 문제이다. (그림 2)는 4개의 입력 변수를 갖는 동일한 부울 함수 $f = x_1x_2 + x_3x_4$에 대해 서로 다른 변수 순서를 적용하였을 때의 BDD 크기를 나타낸다[4, 15]. (그림 2) (a)의 변수 순서는 (x_1, x_2, x_3, x_4)이며, (그림 2) (b)의 변수 순서는 (x_1, x_3, x_2, x_4)이다. 이 그림에서 볼 수 있듯이 입력 변수 순서에 따른 전체 BDD의 크기는 큰 차이가 남을 알 수 있다.



(a) 변수순서 (x_1, x_2, x_3, x_4) (b) 변수순서 (x_1, x_3, x_2, x_4)

(그림 2) 동일함수 $f = x_1x_2 + x_3x_4$에 대한 서로 다른 변수 순서의 BDD 표현

BDD의 최적 변수 순서를 결정하는 것은 NP-complete 문제이므로, 좋은 변수 순서를 결정하기 위하여 많은 변수 순서 최적화 기법들이 제안되었다[10, 11, 13, 14, 16, 17]. 가장 널리 알려진 알고리즘은 Rudell이 제안한 시프팅 알고리즘으로, 시프팅은 인접변수의 순서 교환 기법을 기반으로 하는 동적 변수 순서화 알고리즘이다[7]. 이 알고리즘은 각각의 변수가 가능한 모든 변수 위치를 순회하여 자신의 최적 위치를 찾는 방법으로 한번 자신의 최적 위치를 찾은 변수는 그 위치가 고정된다. 이러한 방법으로 전체 BDD 크기가 최소화 된 변수 순서를 찾는다. 그러나 이러한 방식은 국부 최적(local optimum)의 변수 순서를 찾는 방법으로 실행 시간은 빠르나 전체 최적(global optimum)에 가까운 변수 순서를 찾지 못하는 단점이 있다.

3. Microcanonical Optimization(μO)

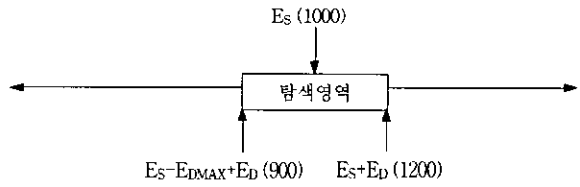
일반적으로 NP-complete 문제를 해결하기 위한 여러 방법 중에서 SA 알고리즘은 가장 최적 해에 근접한 해를 찾는 것으로 알려진 휴리스틱 알고리즘이다. 그러나 SA는 비교적 좋은 해를 찾는 것에 비해 매우 긴 실행 시간을 갖는 단점이 있다. 이러한 단점에 대처하기 위하여 더욱 빠르고 효과적인 대안 방법으로 μO 알고리즘이 Creutz에 의해 제시되었다[20].

μO는 통계 물리학을 기반으로 하며, SA가 외부온도를 조금씩 감소시키면서 시스템의 안정된 상태를 찾아내는 것과는 달리 자기장, 중력, 또는 온도와 같은 외부 환경과 상호작용이 없는 독립된 시스템을 고려한다. 그러므로 시스템의 상태는 외부 환경의 변화와 상관없이 일정한 에너지를 갖으며, 시스템의 모든 가능한 상태들은 동일한 확률로 분포되어 있다. 이러한 외부 환경과 독립되고 동일 확률 분포를 갖는 시스템의 어느 한 상태를 찾기 위한 시뮬레이션 알고리즘이 Creutz 알고리즘이다[20].

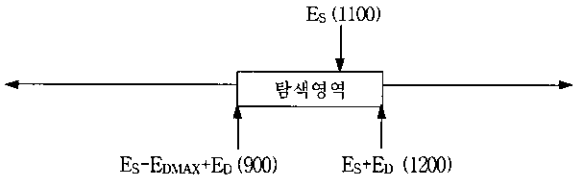
Creutz 알고리즘에는 여분의 자유도(extra degree of freedom)를 나타내는 데몬(demon)이 존재한다. 데몬은 항상 양수(positive) 값을 갖으며, 시스템과 에너지를 서로 교환하면서 시스템과 상호작용 한다. 데몬은 E_D($E_D < E_S$)로 표기하고, 데몬의 최대 용량을 E_{DMAX}로 표기하며, E_S는 시스템 에너지(E_S : System energy)를 의미한다. 시스템의 총 에너지는 데몬과 시스템 에너지의 합이며($E_{TOTAL} = E_D + E_S$), 알고리즘의 샘플링 단계에서 데몬과 시스템의 에너지가 변하더라도 같은 샘플링 단계에서는 시스템의 총 에너지가 항상 일정한 값을 유지한다.

현재 시스템 상태에서 새로운 상태로의 임의 변환(random transition)의 수락 여부는 현재 상태와 새로운 상태의 에너지 차이($\triangle E$ = new E_S - current E_S)에 따라 결정된다. 즉, $\triangle E$가 데몬에 의해 받아 들여 지거나, 또는 거절되어진다. 만약 $\triangle E > 0$일 경우는 $\triangle E \le E_D$일 때만 새로운 변환이 받아들여 지고, $\triangle E \le 0$일 경우는 $E_D - \triangle E \le E_{DMAX}$일 때만 새로운 변환이 받아 들여 진다. 새로운 임의 변환이 받아 들여 지면 데몬은 $E_D = E_D - \triangle E$으로 수정되어진다. 즉, 일정 크기의 데몬이 존재하여, 임의 변환으로 에너지가 남으면 데몬에 저장되며, 에너지가 부족하면 데몬에 저장된 에너지를 시스템에 전달한다. 데몬 크기이상으로 에너지가 남거나, 부족하면 임의 변환은 거절된다.

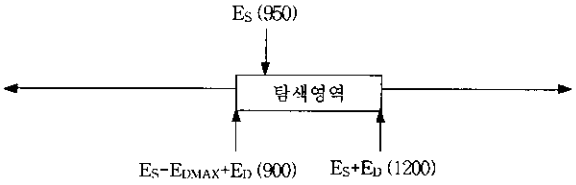
μO은 초기화 단계(Initialization Phase)와 샘플링 단계(Sampling Phase)의 2가지 단계로 구성되어 종료조건이 만족할 때까지 반복 실행한다. 초기화 단계의 목적은 빠른 국부 탐색(local search)을 구현하여 국부 최적 해를 찾는 것이며, 샘플링 단계의 목적은 초기화 단계에서 획득한 국부 최적 해를 벗어나기 위해 Creutz 알고리즘으로 같은 에너지 수준을 갖는 탐색 범위 안에서 대체 해(alternative solution)를 생성하는 것이다. 그러므로 탐색할 이웃 해의 영역은 [$E_S - E_{DMAX} + E_D$, $E_S + E_D$]이다. 데몬에 의해 새로 변환된 해가 수락되거나, 또는 거절된다. 해가 받아 들여 지면 데몬 값은 새로운 값으로 수정된다.



a) E_S = 1000, E_D = 200, E_{DMAX} = 300, E_{TOTAL} = 1000 + 200 = 1200



b) 임의 변환 후 E_S = 1100, E_D = 100, E_{DMAX} = 300, E_{TOTAL} = 1100 + 100 = 1200



c) 임의 변환 후 E_S = 950, E_D = 250, E_{DMAX} = 300, E_{TOTAL} = 950 + 250 = 1200

(그림 3) 탐색영역 안에서의 임의 변환에 따른 데몬, 시스템 에너지와 총 에너지의 변화

(그림 3)은 데몬, 시스템 에너지 그리고 총 에너지의 변화를 나타낸다. 임의 변환은 같은 에너지 수준의 일정한 탐색영역 안에서 실행되어야 받아들여지며, 임의 변환 후 데몬과 시스템 에너지 값의 변화에도 총 에너지는 임의 변환 전과 동일한 값을 갖는다. (그림 3)의 a)는 초기의 데몬과 시스템 에너지 값이다. (그림 3)의 b)는 임의 변환후의 데몬과 시스템 에너지, 그리고 총 에너지의 값이다. $\triangle E$ = 1100 - 1000 = 100 > 0이므로 $\triangle E(100) \leq E_D(200)$ 조건이 만족하여 임의 변환을 받아들이고, 데몬 값은 수정 ($\triangle E$ = 200 - 100 = 100) 된다. 그러나 총 에너지는 1100 + 100 = 1200으로 임의 변환 전과 동일하다. (그림 3)의 c)는 다시 임의 변환된 상태를 나타낸다. $\triangle E$ = 950 - 1100 = -150 < 0이므로 $E_D - \triangle E \leq E_{DMAX}$ (100 - (-150) = 250 ≤ 300) 조건을 만족하여 임의 변환은 받아들여진다. 새로운 데몬 값은 100 - (-150) = 250으로 수정되며, 총 에너지는 950 + 250 = 1200으로 임의 변환 전과 동일하다.

4. BDD 크기 최소화 문제에 적용된 μO 알고리즘

본 장에서는 BDD의 크기 최소화를 위한 μO 알고리즘을 제안한다. μO 알고리즘은 초기화 단계와 샘플링 단계의 2단계로 구성되며, 각 단계에서는 임의의 해가 제안된다. 임의의 해 제안 방법은 두 변수의 위치를 서로 교환하는 Exchange transition, 한 변수를 임의의 앞 위치로 옮겨놓는 Jump-up transition, 그리고 한 변수를 임의의 뒤 위치로 옮겨놓는 Jump-down transition 방법이 있다. 이러한 3가지 방법을 이용하는 것이 한가지 방법만을 이용하는 것보다 더 좋은 결과를 획득할 수 있다[10].

(그림 4)는 BDD의 크기 최소화를 위해 제안된 μO 알고리즘이다. μO 알고리즘은 초기화 단계와 샘플링 단계로 구분되며, 이 두 단계의 사이클이 종료조건을 만족할 때까지 반복 실행한다. μO 알고리즘의 파라메타들은 실행시간 향상을 위해 경험적으로 최적화된다. BDD의 초기 크기를 계산하기 위해 임의의 변수 순서를 발생하여 BDD를 생성하지 않는다. 왜냐하면 임의의 변수 순서는 최악의 경우 BDD의 생성이 불가능 할 수 있다. 그러므로 좀더 효율적인 알고리즘 실행을 위하여 초기 BDD는 시프팅 방법을 이용하여 생성하고 그 크기를 계산한다. 초기화 단계를 마치면 이전의 최적 해와 비교하여 더 좋은 해가 발견되었다면 최적 해에 저장한다.

```
Procedure Microcanonical Optimization
begin
  Sifting ;
  rejected = 0 ;
  while(rejected < STOP_NUM) do
  begin
      Initialization ;
      if (Cost(current_sol) < Cost(best_sol)) then
      begin
          best_sol = current_sol ;
          rejected = 0 ;
      end if
      else rejected++ ;
      Sampling ;
  end while
end
```

(그림 4) BDD 크기 최소화를 위한 μO 알고리즘

(그림 5)는 μO 알고리즘의 초기화 단계이다. 초기화 단계는 임의의 변환된 해를 제안하고, 제안된 임의의 해가 현재의 해보다 더 향상된 해이면 임의의 해를 받아들인다. 즉, 국부 탐색을 실행하는 단계이다. 초기화 단계에서 거절

된 해들은 샘플링 단계의 데몬을 결정하기 위해 편집한다.

```
procedure Initialization
begin
    iter = 0 ;
    while (iter < INIT_NUM) do
    begin
        iter++ ;
        Propse_Move ;
        if (ΔE < 0) then
        begin
            iter = 0 ;
            Accept_Move ;
        end if
    end while
end
```

(그림 5) 초기화 단계

(그림 6)은 μO 알고리즘의 샘플링 단계이다. 샘플링 단계는 이전 초기화 단계에서 얻어진 최소 해가 국부 최소일 경우가 있으므로 이를 벗어나기 위한 작업이다. 샘플링 단계를 시작하기 위해 데몬과 데몬의 최대 값을 결정한다. 데몬은 이전 초기화 단계에서 거절된 임의의 해들 중 그 크기가 가장 작은 해와 그 다음으로 작은 해를 구한 후 그 차이를 계산하여 초기 크기와 비례하는 상수 값을 곱하여 결정한다. 임의의 변환된 해를 제안하고, 임의 해와 현재 해의 크기 차이를 데몬과 비교한다. ΔE가 양수일 경우는 $E_D - \Delta E \geq 0$일 때 임의 해를 현재 해로 받아들이고, ΔE가 음수일 경우는 $E_D - \Delta E \leq E_{DMAX}$일 때 임의 해를 현재 해로 받아들인다. 임의 해를 현재 해로 받아들일 때는 데몬을 $E_D = E_D - \Delta E$으로 수정한다.

```
procedure Sampling
begin
  Demon_Decide ;
  E_DMAX = E_D * 2 ;
  iter = 0 ;
  while ( iter < SAMP_NUM) do
   begin
      iter++ ;
      Propse_Move ;
      if ( (ΔE ≤ E_D) and (E_D - ΔE ≤ E_DMAX) ) then
      begin
        Accept_Move ;
        E_D = E_D - ΔE ;
      end if
   end while
end
```

(그림 6) 샘플링 단계

5. 실험 결과

제안된 μO 알고리즘의 구현은 C언어를 사용하였으며, 콜로라도 주립대학의 CUDD 패키지[12]를 이용하였다. 제안된 알고리즘은 IWLS91 벤치마크[6] 회로에 적용되었으며 실험에 사용된 기종은 96MB 메모리를 갖은 167MHz의 SUN SPARC이다.

본 실험에서 적용된 알고리즘의 파라메타들은 모두 경험적으로 최적화되었다. 파라메타를 결정하기 위해서는 초기 BDD 크기의 측정이 필요하다. 초기 BDD 크기는 시프팅 방법을 이용하여 측정한다. 시프팅 방법을 이용하는 이유는 임의의 변수 순서를 발생하여 BDD의 크기를 측정하려할 때 최악의 경우 BDD를 생성하지 못할 수 있기 때문이다. 알고리즘 실행 반복 횟수인 STOP_NUM은 초기 BDD 크기에 비례하도록 설정하며, 초기화 단계의 반복 횟수인 INIT_NUM는 입력 개수에 비례하도록 설정한다. 샘플링 단계의 반복횟수인 SAMP_NUM은 초기 BDD 크기에 비례하도록 설정하며, 샘플링 단계는 국부 최적 해를 벗어나는 것이 목적이므로 반복 횟수가 많아도 결과에 큰 영향을 주지 않으므로 본 실험에서는 BDD 크기에 따라 5에서 20 이내의 값으로 설정하였다. 실험결과에 가장 큰 영향을 미치는 값은 데몬이다. 데몬이 작다고 항상 실행 시간이 단축되는 것이 아니며, 데몬이 크다고 항상 좋은 해를 찾는 것도 아니다. 각 회로마다 특성에 맞는 데몬을 결정하면 더욱 좋은 해를 구할 수 있으나 본 실험에서는 모든 회로에 동일한 계산식을 이용하여 데몬을 결정하였다. 이전의 초기화 단계에서 거절된 임의의 해들을 정렬하여 그 크기가 가장 작은 해와 그 다음으로 작은 해의 차이를 구한 후 초기 크기와 비례하는 상수 값을 곱하여 데몬을 설정한다.

<표 1>은 제안된 μO 알고리즘의 실험결과이다. circuit name열은 실험 회로의 이름을 나타내며, in과 out 열은 회로의 입출력 단자의 수를 각각 나타낸다. 수행된 알고리즘은 가장 널리 사용되는 Rudell의 시프팅 알고리즘[7], 근사 최적 변수 순서를 찾아내는 SA 알고리즘[10]과 본 논문에서 제안하는 μO 알고리즘이 수행되었다. 실험결과로 볼 때 μO 알고리즘은 SA 알고리즘과 비교하여 BDD 크기와 실행시간에서 월등한 성능 향상을 보였다.

먼저 규모가 크고 복잡한 회로를 비교하면, C1355와 C499, C432 회로는 SA와 동일한 노드 수의 BDD를 생성하여 크기 면에는 동일한 성능을 보였으며, 실행시간은 SA보다 각각 26.33%, 24.06%, 46.01%가 감소되어 성능향상의 결과를 보였다. 또한 C880 회로는 BDD 크기 성능이 0.03% 향상되고, 실행시간은 20.5% 감소되었다. 그러나 C1908 회로는 SA보다 실행시간은 감소되었으나 BDD 크기가 6.18% 더 커져 성능이 저하되었다.

비교적 크기가 적고, 덜 복잡한 회로들의 실험결과를 보면 SA와 비교하여 BDD 크기와 실행시간이 대체로 향상된 결과를 보여준다. apex6, apex7, b9, c8, cm85a, cordic, des, example2, i8, k2, too_large, vda, x1, x2, x3 회로는 BDD 크기 면에서 15.42%~0.40%의 성능 향상을 보였으며, 실행시간은 73.10%~4.80%의 성능 향상의 결과를 보여준다. 그러나 term1 회로는 10.72%의 BDD 크기 성능 향상을 위해 실행 시간은 SA보다 16.72% 오래 걸리는 현상을 보인다.

〈표 1〉 μ O 알고리즘 실험결과

circuit			Sifting		SA		μO	
Name	In	Out	Node	Time	Node	Time	Node	Time
C1355	41	32	30775	26.99	25866	12912.05	25866	9512.45
C3540	50	22	39780	62.85	23831	8574.40	23828	4382.83
C1908	33	25	7153	21.52	5652	1304.13	6024	774.05
C432	36	7	1210	1.36	1209	146.99	1209	79.36
C499	41	32	30775	24.47	25866	12960.47	25866	9843.09
C880	60	26	7064	41.14	4054	1576.37	4053	1253.24
alu2	10	6	162	0.80	154	6.13	154	5.09
alu4	14	8	603	1.27	350	19.25	350	11.03
apex6	135	99	641	1.81	555	179.66	502	125.72
apex7	49	37	304	1.21	253	63.30	214	44.72
b1	3	4	7	0.56	7	0.75	7	0.77
b9	41	21	110	0.75	110	18.45	98	15.57
c8	28	18	83	0.77	81	7.89	80	2.28
cc	21	20	60	0.76	46	5.31	46	1.75
cht	47	36	90	0.75	90	9.59	90	1.68
cm138a	6	8	18	0.71	18	1.31	18	0.96
cm150a	21	1	33	11.00	33	19.21	33	11.84
cm151a	12	2	17	0.71	17	9.08	17	1.12
cm162a	14	5	31	0.65	30	4.17	30	1.41
cm163a	16	5	27	0.66	26	3.60	26	1.14
cm82a	5	2	16	0.67	12	1.05	12	0.71
cm85a	11	3	36	0.69	31	3.32	28	2.06
cordic	23	2	43	0.68	43	12.23	42	3.29
count	35	16	81	0.81	81	24.27	81	3.09
des	256	245	3054	26.13	3037	1026.16	3025	404.16
example2	85	66	295	1.05	270	63.32	266	25.80
i8	133	81	2182	4.18	1300	324.45	1276	300.43
k2	45	45	1394	9.24	1264	203.90	1246	153.02
lal	26	19	86	0.74	67	9.35	86	2.09
mux	21	1	33	10.24	33	18.84	33	11.13
my_adder	33	17	82	13.68	82	39.70	82	15.10
parity	16	1	17	0.70	17	4.25	17	1.17
rot	135	107	8678	65.20	2825	3353.05	3159	1631.17
t481	16	1	21	1.15	21	5.08	21	1.83
term1	34	10	163	1.07	84	26.71	75	32.07
too_large	38	3	652	6.29	315	102.94	303	84.27
vda	17	39	507	2.50	481	24.44	478	21.53
x1	51	35	479	1.39	409	112.09	407	106.72
x2	10	7	37	0.71	32	3.06	31	1.59
x3	135	99	641	2.00	555	178.23	502	136.22
x4	94	71	532	1.37	363	98.09	493	49.15
z4ml	7	4	17	0.62	17	1.53	17	0.86

또한 C1908, lal, rot, x4 회로는 SA보다 BDD 크기면에서 성능이 각각 6.18%, 22.1%, 10.58%, 26.37% 저하되었다. 이와 같은 회로는 데몬을 더욱 크게 하면 좋은 결과를 얻을 수 있다.

<표 2>는 실험결과를 백분율로 비교한 것이다. ▼ 표시는 성능이 SA보다 저하된 것을 의미하며, - 표시는 μO 결과가 SA와 동일한 성능을 의미한다. 먼저 BDD 크기 면에서 41개의 벤치마크 회로 중 17개의 회로가 BDD 크기 감소의 향상된 성능을 보이고, 20개의 회로는 SA와 동일한 BDD 크기의 결과를 보이며, 4개의 회로만이 BDD 크기가 증가하여 성능저하의 결과를 보였다. 실행시간 면에서는 41개의 회로 중 39개의 회로가 실행시간이 감소되었으며, 단지 2개의 회로만이 실행시간이 증가했다. 이러한 결과로 인하여 μO 알고리즘은 SA보다 실행시간 면에서 월등히 우수한 성능을 갖는다는 것이 입증되었다.

전체 실험결과로 볼 때 제안된 알고리즘은 규모가 크고 복잡한 회로는 실행시간을 대폭 감소시키며, 규모가 작고 덜 복잡한 회로는 BDD 크기와 실행시간의 향상을 보여주어 그 효율성이 증명되었다.

6. 결 론

BDD는 회로 합성과 형식 검증에 널리 사용되는 회로 표현 기법으로서 BDD 크기의 최소화는 매우 중요한 문제이

〈표 2〉 SA와 μO의 성능 비교(%)

circuit	SA		μO			
Name	Node	Time	Node		Time	
C1355	25866	12912.05	25866	-	9512.45	26.33%
C3540	23831	8574.40	23828	0.02%	4382.83	48.89%
C1908	5652	1304.13	6024	▼ 6.18%	774.05	40.65%
C432	1209	146.99	1209	-	79.36	46.01%
C499	25866	12960.47	25866	-	9843.09	24.06%
C880	4054	1576.37	4053	0.03%	1253.24	20.5%
alu2	154	6.13	154	-	5.09	16.97%
alu4	350	19.25	350	-	11.03	42.71%
apex6	555	179.66	502	9.55%	125.72	30.03%
apex7	253	63.30	214	15.42%	44.72	29.36%
b1	7	0.75	7	-	0.77	▼ 2.60%
b9	110	18.45	98	10.91%	15.57	15.61%
c8	81	7.89	80	1.24%	2.28	71.11%
cc	46	5.31	46	-	1.75	67.05%
cht	90	9.59	90	-	1.68	82.49%
cm138a	18	1.31	18	-	0.96	26.72%
cm150a	33	19.21	33	-	11.84	38.37%
cm151a	17	9.08	17	-	1.12	87.67%
cm162a	30	4.17	30	-	1.41	66.19%
cm163a	26	3.60	26	-	1.14	68.34%
cm82a	12	1.05	12	-	0.71	32.39%
cm85a	31	3.32	28	9.68%	2.06	37.96%
cordic	43	12.23	42	2.33%	3.29	73.10%
count	81	24.27	81	-	3.09	87.27%
des	3037	1026.16	3025	0.40%	404.16	60.62%
example2	270	63.32	266	1.49%	25.80	59.26%
i8	1300	324.45	1276	1.85%	300.43	7.41%
k2	1264	203.90	1246	1.43%	153.02	24.96%
lal	67	9.35	86	▼ 22.10%	2.09	77.65%
mux	33	18.84	33	-	11.13	40.93%
my_adder	82	39.70	82	-	15.10	61.97%
parity	17	4.25	17	-	1.17	72.48%
rot	2825	3353.05	3159	▼ 10.58%	1631.17	51.36%
t481	21	5.08	21	-	1.83	63.98%
term1	84	26.71	75	10.72%	32.07	▼ 16.72%
too_large	315	102.94	303	3.81%	84.27	18.14%
vda	481	24.44	478	0.63%	21.53	11.91%
x1	409	112.09	407	0.49%	106.72	4.80%
x2	32	3.06	31	3.13%	1.59	48.04%
x3	555	178.23	502	9.55%	136.22	23.58%
x4	363	98.09	493	▼ 26.37%	49.15	49.90%
z4ml	17	1.53	17	-	0.86	43.80%

다. 본 논문에서는 μO 알고리즘을 이용하여 새로운 변수 순서화 알고리즘을 제안하고, IWLS91 벤치마크 데이터 실험을 통해 그 효율성이 검증되었다.

제안된 μO 알고리즘은 기존의 알고리즘들 중에서 가장 근사 최적 값을 찾는 휴리스틱 알고리즘인 SA의 실행 시간이 오래 걸리는 단점을 해소시킨 알고리즘으로, 보다 빠른 시간 내에 더욱 근사 최적 변수 순서를 찾아낸다. 제안된 알고리즘은 기존의 휴리스틱 알고리즘에 비하여 BDD 크기와 실행 시간 면에서 향상된 성능을 나타내어 그 효율성이 증명되었다.

향후 연구과제로는 실행 시간이 감소하면서 BDD 크기의 성능도 향상되는 μO 알고리즘의 개선 방안의 연구가 필요하다.

참 고 문 헌

[1] S. B. Akers, "Binary Decision Diagrams," IEEE Transaction on Computers, Vol.C-27, No.6, pp.509-516, August 1978.

[2] Randal E. Bryant "Symbolic Manipulation of Boolean Functions Using a Graphical Representation," 22nd Design Automation Conference, pp.688-694, June 1985.

[3] Randal E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transaction on Computers, Vol.C-35, pp.667-691, August 1986.

[4] Randal E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," ACM Computing Surveys, Vol.24, No.3, September 1992.

[5] Randal E. Bryant, "Binary Decision Diagrams and Beyond : Enabling Technologies for Formal Verification," International Conference on Computer Aided Design, pp.236-243, November 1995.

[6] Saeyang Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Distributed as part of the IWLS91 benchmark distribution, January 15, 1991.

[7] Richard Rudell, "Dynamic Variable Ordering of Ordered Binary Decision Diagrams," International Conference on Computer-Aided Design, pp.43-47, November 1993.

[8] D. Moller, P. Molitor, R. Orechsler, "Symmetry based variable ordering for Obdds," IFIP Workshop on Logic and Architecture Synthesis, Grenoble, December, 1994.

[9] S. Panda, F. Somenzi, "Who are the variables in your neighborhood," In Proceedings of the International Conference on Computer-Aided Design, pp.74-77, San Jose, CA. November 1995.

[10] B. Bolling, M. Lobbing, I. Wegener, "Simulated Annealing to improve variable orderings for OBDDs," International Workshop on Logic Synthesis, pp.5b:5.1-5.10, 1995.

[11] Rolf Drechsler, Bernd Beckern, Nicole Gockel, "A Genetic Algorithm for Variable Ordering of OBDDs," International Workshop on Logic Synthesis, pp.5C:5.55-5.64, 1995.

[12] Fabio Somenzi, "CUDD : CU Decision Diagram Package Release 2.3.0," University of Colorado, April 12, 1997.

[13] Rolf Drechsler, Nicole Gockel, "Simulation Based Minimization of large OBDDs," Preprint at the Institute of Computer Science, Albert-Ludwigs-University Freiburg im Breisgau, germany, January 1997.

[14] Rolf Drechsler, Nicole Gockel, "Minimization of BDDs by Evolutionary Algorithms," International Workshop on Logic Synthesis (IWLS'97), Lake Tahoe, 1997.

[15] Rolf Drechsler, Bernd Becker, "Binary Decision Diagrams : Theory and Implementation," Kluwer Academic Publishers, 1998.

[16] S. J. Fredman, K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," IEEE Transaction on Computers, Vol.C-39, No.5, pp.710-713, May 1990.

[17] H. Fujii, G. Otomo, G. Hori, "Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams : International Conference on Computer-Aided Design," pp.622-627, Nov 1993.

[18] Alan J. Hu, "Formmal Hardware Verification with BDDs : introduction," IEEE pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'97), 1997.

[19] Christoph Scholl, Rolf Drechsler, Bernd Becker, " Functional Simulation using Binary Decision Diagrams," ACM/IEEE International Conference on Computer Aided Design (ICCAD '97), pp.8-12, San Jose, 1997.

[20] M. Creutz, "Microcanonical Monte Carlo simulation," Phys, Rev, Lett, Vol.50, pp.1411-1413, 1983.

[21] Alexandre Linhares, Horacio H. Yanasse, Jose R. A. Torreao, "Linear Gate Assignment : A Fast Statistical Mechanics Approach," IEEE Transaction on Computer-Aided Design of Integrated circuits and system, Vol.18, No.12, December 1999.

[22] S. C S. Porto, A. M. Barrose, J. R. A. Torreao, "A Parallel Microcanonical Optimization Algorithm for the Task Scheduling Problem," Technical Report, Applied Computing & Automation Universidado Federal Fluminense, September 1999.

[23] A. Linhares, J. A. Torreao, "Microcanonical optimization applied to the traveling salesman problem," Int. J. Modern Phys. C, Vol.9, pp.133-146, 1998.

이 민 나

e-mail : mnlee@san.hufs.ac.kr

1988년 성신여자대학교 수학과 졸업(학사)

1991년 한국외국어대학교 경영정보대학원 응용전산학과(이학석사)

2000년 한국외국어대학교 대학원 컴퓨터 공학과 박사과정 수료

1991년~1995년 내무부 전산지도과 근무

관심분야 : 컴퓨터아키텍쳐, 병렬처리, CAD

조 상 영

e-mail : sycho@hufs.ac.kr

1988년 서울대학교 제어계측학과(공학사)

1990년 한국과학기술원 전기 및 전자공학과 (공학석사)

1994년 한국과학기술원 전기 및 전자공학과 (공학박사)

1994년~1995년 KAIST 정보전자 연구소 연구원

1996년~1997년 삼성전자 선임연구원

1997년~현재 한국외국어대학교 컴퓨터공학과 조교수

관심분야 : 컴퓨터구조, 내장형시스템, 병렬처리