종분화 기법을 이용한 진화 하드웨어의 다양성 향상

(Increasing Diversity of Evolvable Hardware with Speciation Technique)

황 금 성 † 조 성 배 ††
(Keum-Sung Hwang) (Sung-Bae Cho)

요 약 진화 하드웨어(evolvable hardware)는 재구성 가능한 디지털 회로에 진화연산이 적용되어 실시간으로 환경에 적응함으로써 필요한 기능을 자동적으로 구현하는 기술이다. 이는 하드웨어 회로의 자동 설계 가능성을 열어 주었지만, 아직 복잡한 회로를 얻기에는 한계가 있다. 본 논문에서는 진화 하드웨어의 적합도 공간을 분석하여, 다양한 개체가 동시에 진화되는 종분화 기법을 제안하고 그 효율성을 실험적으로 보인다. 또한 종분화 기법으로 얻은 다양한 회로를 분석하여 유용한 부가 기능이 창출될 수 있음을 보인다.

키워드 : 진화 하드웨어, 종분화 기법, 다양성, 부가 기능, 디지털 회로의 설계

Abstract Evolvable Hardware is the technique that obtains target function by adapting reconfigurable digital devices to environment in real time using evolutionary computation. It opens the possibility of automatic design of hardware circuits but still has the limitation to produce complex circuits. In this paper, we have analyzed the fitness landscape of evolvable hardware and proposed a speciation technique of evolving diverse individuals simultaneously, proving the efficiency empirically. Also, we show that useful extra functions can be obtained by analyzing diverse circuits from the speciation technique.

Key words : evolvable hardware, speciation technique, diversity, extra function, digital circuit design

1. 서 론

1975년 John Holland가 생물학적 학습 원리를 적용하여 창시한 유전자 알고리즘은 문제에 대한 해를 표현한 염색체 집단을 인공적으로 진화시켜서 문제를 해결한다[1]. 이러한 방법은 인간의 전문 지식을 통한 해결이 어려운 문제에 대해 탁월한 성능을 보이기 때문에, 많은 복잡한 문제의 해법으로 사용되어 왔고 1980년대부터는 하드웨어의 설계에도 이용되기 시작하였다[2]. 초기의 하드웨어 진화는 회로구조의 최적화 정도에 머물렀으나, 1980년대에 실시간으로 하드웨어 재구성이 가능한 PLD (Programmable Logic Devices)와 같은 하드웨어가 개발되면서 실시간 최적화가 가능하게 되어 주목 받기 시작하였고, 1990년대에 세밀한 조작 및 빠른 프로그래밍이 가능한 FPGA(Field Programmable Gate Array)가 등장하면서 진화 하드웨어에 대한 기대와 관심이 크게 증가하였다[3]. 1993년 Higuchi 등은 게이트 회로를 빌딩블록으로 하는 게이트수준의(gate-level) 진화 하드웨어를 구현하고 간단한 회로가 실시간으로 진화되는 실험을 하였다[4]. 그리고 Sipper 등[5]과 Yao 등[2]은 여러 연구를 통해 진화 회로 설계를 위한 다양한 방법을 소개하였으며, Thompson[6]은 디지털 회로의 효율적인 진화에 대한 연구들을 종합하여 소개하였다.

진화 하드웨어는 주어진 환경에서 들어오는 정보를 입력 받아 목표로 하는 기능에 대해 최적의 성능을 내는 하드웨어 구조를 찾아낸다. 이러한 기능은 변화가 심한 환경에서 적응력이 강하고 성능이 안정적인 하드웨어를 가능케 할 뿐만 아니라, 예기치 않은 문제 상황을

· 이 연구(논문)는 산업자원부 지원으로 수행하는 21세기 프론티어 연구개발사업(인간기능 생활지원 지능로봇 기술개발사업)의 일환으로 수행되었습니다.

† 학생회원 : 연세대학교 컴퓨터과학과
yellowg@candy.yonsei.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
sbcho@csai.yonsei.ac.kr

논문접수 : 2002년 5월 14일
심사완료 : 2004년 11월 2일

자동으로 극복할 수 있게 하며, 환경조건에 따라 다르게 기능하는 유한상태기계를 구현할 수 있다[7]. 그리고 진화에 의해 탐색되는 하드웨어 구조는 다양한 가능성을 고려하기 때문에, 보다 최적화된 우수한 성능의 하드웨어 개발이 가능하다.

그러나 목표 기능이 크고 복잡할수록 이를 표현하는데 필요한 염색체의 길이가 길어지고 진화에 요구되는 시간이 기하급수적으로 늘어나기 때문에 복잡한 하드웨어를 진화 하드웨어로 구현하기는 어렵다. 이러한 문제를 해결하기 위해서 여러 연구 및 방법들이 제시되었다. 좀더 높은 수준의 기능을 진화의 단위로 사용하는 기능 수준의(function-level) 진화 하드웨어 설계 방식이 Liu 등[8]에 의해 제안되었고, Kalganova[9]에 의해 기능 수준과 게이트 수준을 모두 사용하는 회로 설계 방법이 연구되었으며, 하드웨어를 한 번에 진화시키지 않고 여러 부분으로 나누어 진화한 뒤 결합하는 분할정복 하드웨어 설계 방법이 Torresen[10]에 의해 제시되었다. 또한 Vassilev[11]는 적합도 공간을 분석하여 진화 하드웨어의 확장성을 보임으로써 좀더 큰 하드웨어 단위를 가지고 진화시키는 방법의 유용성을 입증하였다.

본 논문에서는 생물의 진화 개념 중 하나인 종분화(speciation) 기법을 적용하여 효과적으로 다양한 하드웨어 회로를 진화시키고자 한다. 종분화 기법은 개체의 진화가 좁은 방향으로 진행되지 않고 다양한 종으로 분화하여 진화하도록 유도하는 진화연산 기법의 하나로써 이미 많은 분야에서 인정받고 있으며[12], 회로 기능을 가진 유전자 프로그램(genetic program)에 종분화를 적용하여 진화의 향상시킨 연구도 있었다[13]. 본 논문에서는 이 방법을 이용하여 종의 분화를 통해 환경에 더욱 잘 적응된 개체가 나타나는 자연계에서의 효과를 진화 하드웨어에 구현하고자 한다. 또한, 하드웨어의 종분화가 어떻게 일어나는지 알아보고 그 가능성과 유용성을 살펴본다. 제안하는 방법을 이용한 실험 내용은 이미 [14]를 통해 소개된 바 있으나, 본 논문에서는 제안하는 방법을 적용하기 위한 이론 및 추가 실험 내용, 새롭게 고안하여 사용된 분산 룰렛휠 선택 방법 등 기존에 지면상 소개하지 못했던 자세한 부분들을 수록하였다.

2. 배 경

2.1 디지털 회로의 진화

진화연산은 생명체의 진화를 모방하여 문제를 해결하는 것으로 크게 유전자 알고리즘(Genetic Algorithm: GA)과 진화 전략(Evolutionary Strategy) 및 진화 프로그래밍(Evolutionary Programming)이 축을 이루고 있다[15]. 이는 자연의 적자생존 과정을 모델링한 것으로, 한 세대를 형성하는 개체들 중에서 환경에 대한 적합도가 높은 개체가 살아남을 확률이 높다는 다윈의 학설에 기반을 둔다[1]. 즉, 한 세대의 여러 개체들 중에서 환경에 대한 적응력이 우수한 개체들을 다음 세대의 구성원으로 선택하고, 유전 연산자를 적용하여 새로운 개체를 생성하는 것이다. 이때 각 개체는 유전자들의 열인 염색체로 표현되며, 이 염색체의 변화가 진화과정에 해당된다. 유전 연산자 중에서 교차(crossover)는 두 개체 간의 유전자를 부분적으로 서로 바꿔서 새로운 유전자 열의 개체를 생성하는 방법이며, 돌연변이(mutation)는 임의의 유전자를 확률에 따라 대립형질의 유전자 값으로 바꾸는 것으로, 해당 개체에 근접한 새로운 개체를 생성하거나 집단에서 잃어버린 유전 형질을 복구하는 기능을 한다.

일반적으로 디지털 회로를 진화시키기 위해서 사용되는 하드웨어는 FPGA(Field Programmable Gate Array)와 같은 재구성가능 하드웨어이다. 본 논문에서는 프로그램 가능형 논리 장치(PLD: Programmable Logic Device)의 하나인 GAL16V8칩을 대상으로 하였다. PLD는 사용자가 실현하고자 하는 임의의 기능을 고속으로 간결하게 지원하며 개발의 용이성을 고려한 전기적인 기능교체가 여러 번 가능하기 때문에 실험에 유리하다. 그림 1의 GAL16V8 구조는 8개의 출력 논리 마크로셀(OLMC: Output Logic Macro Cell)과 그 입력값 사이의 연결 장치, 그리고 적항입력비트(product-term-disable-bit)로 구성되어 있다.

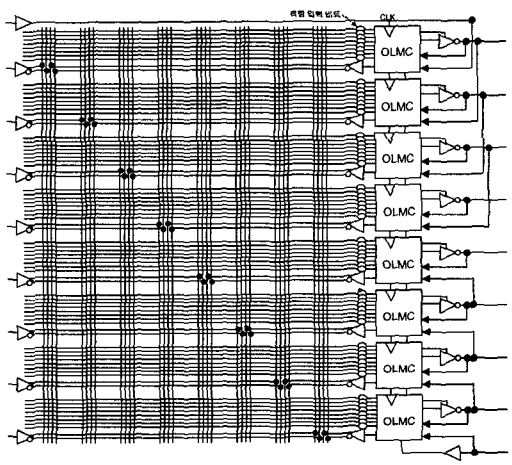


그림 1 PLD의 하나인 GAL16V8 회로

OLMC는 범용의 논리회로로, 기능지정 비트열에 따라 기능이 다양하게 바뀌는 하드웨어 장치이다. 구체적으로 그림 2와 같은 구조를 가지고 있으며 SYN, AC0, AC1에 들어가는 비트값을 다르게 하여 5가지의 기능을 구현할 수 있다.

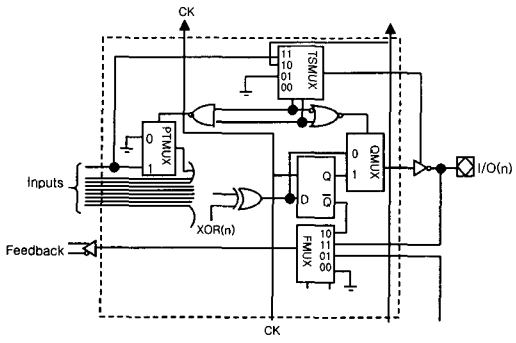


그림 2 GAL16V8 칩의 OLMC 구조

OLMC 입력값과 외부입력값 사이의 연결 부분은 2차원상의 퓨즈 배열로 구성되어 있어서 임의로 AND 조합하는 것이 가능하다. 적항입력비트는 OLMC에 들어가는 입력의 사용여부를 결정짓는 비트이다. 각 OLMC 입력부에 위치하여 해당비트값이 '1'인 경우 들어오는 입력의 사용을 허가하고, '0'인 경우 그 사용을 금지한다. 이처럼 논리마크로셀의 기능을 결정하는 비트열과 퓨즈 배열의 패턴을 지정하는 비트열, 그리고 적항입력 비트열이 GAL 회로의 "프로그램"을 결정짓게 되며 이를 합쳐 "아키텍처 비트"라고 한다. 진화 하드웨어에서는 이 아키텍처 비트를 염색체로 표현하여 진화시킨다[4].

본 논문에서는 성능 평가를 위해 앞서 소개한 GAL16V8 칩을 간소화시켜 사용하였다. 그림 3은 6개의 외부입력과 1개의 8입력 OLMC를 가지고 있고, OLMC의 기능이 OR 조합 기능으로 고정되어 있는 단순화된 GAL 회로를 보여준다. 이 구조는 실험에서 목표로 정한 4개의 외부입력과 2개의 선택입력을 갖는 6멀티플렉서를 진화시키기에 적당한 크기를 가지고 있다.

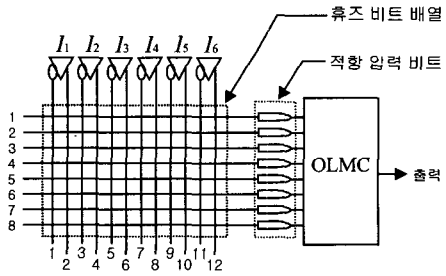


그림 3 단순화된 GAL 회로

2.2 종분화 기법

유전자 알고리즘은 탐색이 진행됨에 따라 해공간상의 한 지점으로 수렴하는 경향이 있기 때문에 단일해 문제에 대해서는 유용한 방법이지만, 다중해(multimodal) 최적화 문제에서 전역해를 찾는 데에는 효과적이지 않다[1]. 유전자 알고리즘의 이러한 문제를 해결하기 위해 고안된 것이 니칭 기법(niching technique)이다. 이것은 진화 알고리즘의 선택 연산자에 의한 유전적 해쏠림(genetic drift)을 감소시키기 위해 개발된 방법으로, 개체의 다양성을 유지시키고 동시에 여러 해영역을 탐색하도록 유도하며 지역해에 빠져서 조기수렴하는 것을 방지한다. 대표적인 방법으로 적합도 공유(fitness sharing), 크라우딩(crowding), 한정된 토너먼트 선택(restricted tournament selection) 방법 등이 있다[16].

종분화 기법은 이후에 제안된 다양성 향상 기법 개념의 하나로서 좀더 생명체의 진화 개념에 근접한다. 니칭 기법이 단순히 다양성 유지와 해쏠림 방지에 관심을 두고 있는 반면, 종분화 기법은 개체의 종별 분화를 염두에 두고 있다[12]. 즉, 종의 구분을 두어 같은 종 내에서만 교차를 허용하는 제한된 교배(restricted mating) 방법이나 개체 유사도를 평가하여 비슷한 개체들이 종 단위로 살아남도록 유도하는 방법 등 자연상의 종분화 현상을 모방하여 종의 분화를 유도한다. 기존에 제안된 니칭 기법 중에서 이런 특성을 가진 방법들은 종분화 기법이라고도 할 수 있으며 최근에는 종분화 기법으로 부르는 것이 선호되고 있다.

본 논문에서는 하드웨어에 종분화 기법을 적용하기 위해 명시적 적합도 공유(explicit fitness sharing) 방법을 사용한다[1,17]. 이것은 1989년 Goldberg와 Richardson에 의해 처음 소개되었는데, 직관적인 다양성 유지 방법을 사용하고 있어서 관찰이 용이하고 그 성능도 좋기 때문에 널리 사용되고 있다. 명시적 적합도 공유 기법은 선택연산자의 기준이 되는 적합도를 조정해 줌으로써 종분화의 효과를 유도하는 방법이다. 즉, 각 개체별로 공유범위(σ_s) 내에 속한 개체의 수량(니치 카운트 m_i)을 계산하고 적합도를 이것으로 나눠주어 생존확률을 낮춤으로써 자손이 한 종으로 몰리지 않도록 유도한다. 이는 자연계에서 특정 장소에 너무 많은 개체가 모이면 각 개체에게 돌아가는 자원의 양이 줄어드는 것과 같은 이치이다. 그림 4는 실제로 여러 개의 지역해를 가진 함수($y=x \cdot sin^2(x)$) 속에서 특정 위치에 개체가 많이 모여 있을 경우 적합도가 어떻게 변하는지를 보여준다.

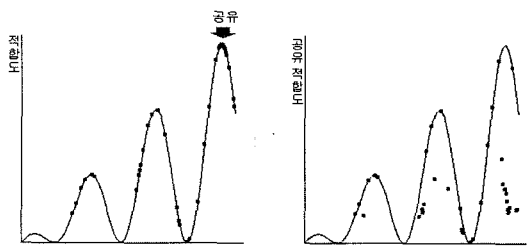


그림 4 적합도를 공유했을 때 적합도 공간상에서의 변화

3. 진화 하드웨어의 적합도 공간

본 장에서는 진화 하드웨어에서의 종분화 사용 타당성을 알아보기 위해 적합도 공간의 형태를 조사한다. 앞서 제시한 간소화된 GAL 회로는 퓨즈비트 배열과 적항입력비트를 모두 연결하여 염색체로 사용할 경우 표현 가능한 수가 $2.03\times10^{31}(=2^{104})$인데, 이는 전부 조사하기에는 너무나 큰 수이다. 따라서 그림 5와 같은 아주 간소화된 GAL 회로를 구현하여 해공간을 조사하기로 한다. 이 회로는 외부입력이 2개이고 OLMC에 들어가는 입력도 2개이며 적항입력비트를 쓰지 않았으며, OLMC의 기능은 항상 OR 조합을 하도록 하였고, 적항입력비트가 없는 대신 OLMC의 입력 기본값을 'ON'으로 하여 외부입력과의 AND 조합을 통해 값이 변할 수 있도록 하였다. 또한 4장에서 소개할 염색체 축약 방법을 여기에 적용하여 3진수 염색체를 사용하였다.

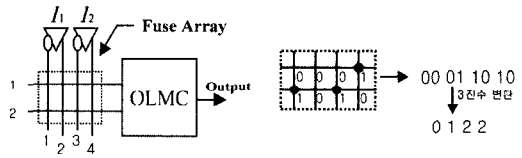


그림 5 반덧셈기를 진화시키기 위해 아주 간소화된 GAL 회로(왼쪽)와 염색체의 예(오른쪽)

목표하드웨어를 반덧셈기로 정하고 실험한 결과 그림 6과 같은 적합도 공간 그래프가 나왔다. 이것은 표현 가능한 염색체 81개에 대해서 1차원으로 배열한 뒤 각 염색체에 대한 적합도를 표시한 결과이다. 이때 적합도는 모든 입력 조합에 대한 출력값의 정답비율을 사용하였다. 그림 6을 보면 예상대로 상당히 많은 굴곡(pike)이 있음을 알 수 있다. 그리고 적합도가 0인 지점이 적합도

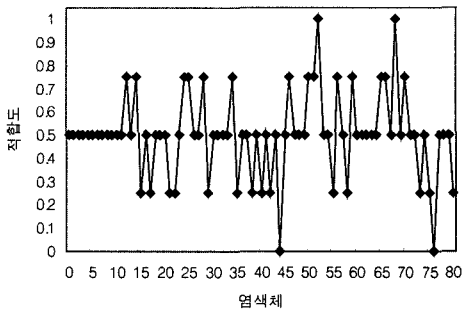


그림 6 염색체를 1차원에 배열한 적합도 공간 그래프

가 1인 2개의 전역해와 상당히 가깝게 있음을 볼 수 있다. 이는 해 근처의 함정으로 작동할 가능성이 크다. 즉, 근처까지 온 염색체에게 낮은 적합도를 보임으로써 부근에 대한 탐색 가능성을 낮추는 영향을 주는 것이다.

그림 7은 실험에 사용한 염색체를 절반으로 나누어 유전자1, 유전자2로 정의하고, 이를 2차원의 평면에 표시하여 적합도에 따른 굴곡을 3차원으로 나타낸 것이다. 오른쪽에 나타나 있는 그래프는 왼쪽 그래프의 평면도이다. 역시 2개의 전역해 주변이 경사가 상당히 심하고 많은 굴곡이 존재해서 진화를 통한 전역해 탐색이 쉽지 않은 문제임을 보여주고 있다. 예를 들어 오른쪽 그림에서 전역해가 위치한 (6, S8)의 이웃을 살펴보면 (5, S7)을 제외하고는 모두 0.6이하의 값을 보이고 있다.

적합도 공간의 표현은 염색체가 2차원보다 큰 경우 적절히 차원을 줄여서 관찰할 수밖에 없다. 하지만 이러한 변형은 정확한 해공간을 반영한다고 볼 수는 없다. 따라서 해간 상관성을 좀더 정확히 분석하기 위하여 해밍거리를 이용한 분포를 조사하였다. 그림 8은 그림 7에 보이는 두 개의 해 중 하나인 염색체 "2112"(위치: 8, S8)를 기준으로 한 해밍거리 분포인데, 두 개의 해가 4

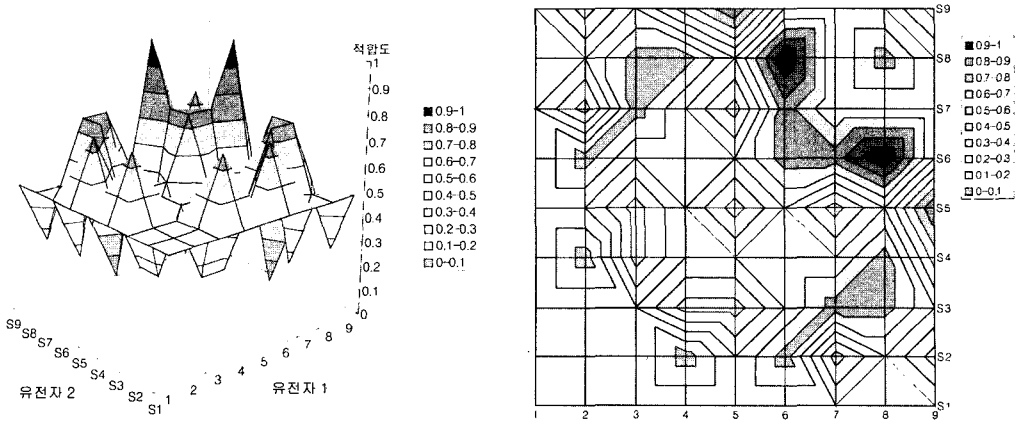


그림 7 염색체를 2차원에 배열한 적합도 공간의 그래프(왼쪽)와 평면도(오른쪽)

의 거리로 가장 멀리 떨어져 있다. 이는 그림 6과 7과는 좀 다른 결과인데 앞의 그래프들이 차원을 줄이는 과정에서 적합도 공간이 조금 왜곡되면서 해가 가깝게 되었기 때문이다. 그림 8에서 최저값 0을 보이는 함정은 두 해에서 모두 2의 거리만큼 떨어져 있는데 이는 앞서 살펴본 그래프에서의 결과와 동일하다.

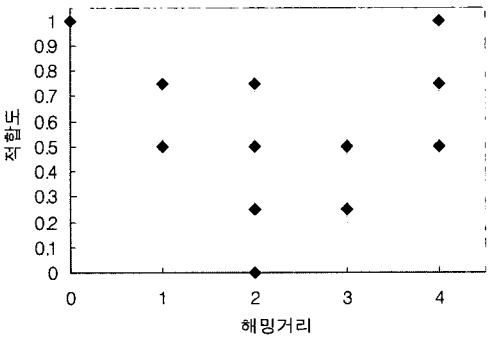


그림 8 해밍거리에 따른 염색체 분포(염색체 "2112"를 기준으로 측정)

이처럼 진화 하드웨어는 적합도 공간이 굴곡이 심하고 전역해 근처에 왜곡(deception) 혹은 함정이 많기 때문에 기본 유전자 알고리즘으로는 해를 찾기가 쉽지 않다. 적합도 함수가 하드웨어의 성능 평가로 이루어져 있고, 디지털 회로는 일부의 구조만 달라져도 기능이 많이 바뀔 수 있기 때문에, 해에 근접한 하드웨어를 찾기 어려운 것이다. 따라서 전역해를 효율적으로 찾기 위해서는 종분화와 같은 다양성 기법이 필요하다.

4. 진화 하드웨어의 종분화

종분화된 진화 하드웨어는 하드웨어 시뮬레이션을 통해서 적합도가 평가되고 공유적합도의 계산을 통해 종의 분화가 유도되어 여러 해를 한번에 얻을 수 있는 특징을 가지고 있다.

4.1 염색체 표현

디지털 회로의 구조 정보를 담을 염색체를 위해서 단순화된 GAL 칩의 퓨즈 비트 배열과 적항입력 비트열을 포함하는 아키텍처 비트를 하나의 배열로 연결하여 사용하였다. 6개의 외부입력이 NOT 게이트와의 조합을 하므로 각 OLMC 입력에 12비트의 퓨즈 비트가 사용되고, OLMC가 8개의 입력 조합을 받아들이므로 퓨즈 비트는 총 96비트가 사용된다. 그리고 각 OLMC 입력에 대한 적항입력비트가 8개이므로 염색체의 길이는 104비트가 된다. 그러나 이 하드웨어 구조에서 퓨즈비트를 통해 외부입력을 AND 조합할 때 NOT을 통과한 값과 그렇지 않은 값을 AND 조합하는 경우는 아래 수식과 같이 무의미한 값을 가지게 된다.

$$I_k \text{ AND (NOT } I_k) = 0, \text{ where } k=1,2,\cdots,n \quad (1)$$

따라서 본 논문에서는 각 입력별로 위 수식과 같은 AND 조합을 나타내는 비트쌍 '11'의 경우는 제외하고 사용하였다. 그림 5에서 "00 01 10 10"을 "0 1 2 2"로 줄인 것처럼 조합 입력 부분의 값을 3진수 문자열로 축약하여 염색체 표현에 의한 경우의 수를 줄였다. 이 결과, 염색체의 길이는 56으로 단축되었고 진화의 속도도 향상시킬 수 있었다.

4.2 적합도 계산

적합도 연산을 위한 목표 하드웨어는 6멀티플렉서를 사용하였다. 6멀티플렉서는 4개의 외부입력과 2개의 선택입력을 가지고 있는 회로로서, AND와 OR의 조합만으로 구성이 가능하고 적당한 세대 진화로 획득이 가능하기 때문에 진화 하드웨어의 성능을 테스트하기에 좋다. 본 논문에서는 편의상 선택입력 S_1과 S_2를 I_5와 I_6으로 표현하였다.

염색체로 구성되어 얻어지는 하드웨어에 대한 적합도 평가를 위해서 각 염색체가 표현하는 아키텍처 비트를 하드웨어로 구성한 다음 입력 가능한 모든 비트 조합에 대한 출력 패턴을 조사하여 목표 하드웨어의 입·출력 패턴과 비교하였다. 이때 개체 i에 대한 적합도 f_i는 다음과 같은 수식으로 정의될 수 있다.

$$f_i = \left(1 - \frac{\sum_{\mathbb{K}' \in \mathbb{K}} | T(\mathbb{K}') - O_i(\mathbb{K}') |}{|\mathbb{K}|}\right) \times 100 \quad (2)$$

여기에서 $T(\ )$는 입력 집합이 인 경우 나오는 목표 하드웨어의 출력값을, $O_i(\ )$는 입력 집합 에 대한 개체 i의 출력값을 나타내며, n_i는 입력의 수를, | | 는 집합 의 원소 수를 의미한다.

식 (2)에 사용된 는 입력의 집합을 나타내는 변수로서 다음과 같이 정의된다.

$$= \{ x_1, x_2, \cdots, x_{n_i} | x_k \in 0,1, k=0,1,\cdots,n_i \}$$

위의 수식은 입·출력패턴의 차이를 카운트하여 그 확률을 계산한 다음 적합도 함수로 쓰기 좋도록 차이가 적을수록 큰 값이 나오도록 정하고 있다. 즉, 개체 i의 적합도 f_i는 목표 하드웨어와의 입·출력패턴 일치회수를 전체 비교 회수로 나눈 백분율로 정의된다.

종분화를 유도하기 위해서는 적합도를 공유하여 공유적합도를 계산해야 한다. 공유 적합도 sf_i의 계산 과정을 수식으로 표현하면 다음과 같다.

$$sf_i = \frac{f_i}{m_i} \quad (3)$$

f_i는 개체 i에 대한 적합도를 의미하며 m_i는 공유해야 할 개체의 수를 나타내는 니치 카운트이다. 니치 카운트 m_i를 계산하는 방법은 다음과 같다.

$$m_i = \sum_{j=1}^{n} sh(d_{ij}) \quad (4)$$

여기서 n은 집단의 크기(전체 개체수)를 의미하고 $sh(d_{ij})$는 개체간 공유 상태를 계산해주는 함수이다. 즉, 개체 i와 j의 거리(차이)를 나타내는 d_{ij}를 이용해서 공유거리 내에 들어와 있을 경우 공유의 정도를 계산해 준다. 공유함수 $sh(d_{ij})$를 계산하는 방법은 다음과 같다.

$$sh(d_{ij}) = \begin{cases} 1-(\frac{d_{ij}}{\sigma_s})^{\alpha}, & for \ \ 0 \le d_{ij} < \sigma_s \\ 0 \ , & for \ \ d_{ij} \ge \sigma_s \end{cases} \quad (5)$$

이때 σ_s는 공유반경으로 개체간 공유 범위를 정하는 파라미터이며, α는 공유 함수의 모양을 결정짓는 상수이다. 일반적으로 α값은 1을 사용한다. 위의 수식은 개체 i와 j의 거리가 공유거리 이내인 경우 그 거리가 가까운 정도에 따라서 0에서 1 사이의 공유를 하도록 한다. 따라서, 만일 모든 개체가 공유 거리 밖에 있을 경우에는 자기 자신의 $sh()$값만 1이 되므로 m_i값은 1이 되고 공유적합도는 적합도와 같게 된다.

개체간 거리 d_{ij}는 별도의 유사도 측정 함수에 의해 정의되는데, 보통 유전자의 차이를 계산하는 유전자형(genotype) 거리와 염색체가 의미하는 특성의 차이를 계산하는 표현형(phenotype) 거리가 사용된다. 본 논문에서는 염색체의 특성을 반영하기 위해 두 가지 방식을 모두 사용한다. 즉, 염색체 구조에서 OLMC의 입력으로 들어가는 입력의 AND 조합식은 독립적인 성격을 띤 부분이라고 할 수 있기 때문에 이를 '블록'으로 정의하고, 블록단위의 거리 계산은 유전자형 거리의 하나인 해밍거리(Hamming distance) 방식을 이용하였으며, 블록의 OR조합이라 할 수 있는 염색체 전체의 거리 계산은 표현형 거리의 하나인 유클리드거리(Euclidean distance) 방식을 이용하였다[18]. 다음은 개체간 거리를 계산하는 수식이다.

$$d_{ij} = \sqrt[2]{\sum_{k=1}^{8} (h_{ij}(k))^2} \quad (6)$$

여기서 $h_{ij}(k)$는 k번째 블록에 대한 개체 i와 j의 해밍거리를 나타낸다. 블록간 해밍거리를 계산하는 방법은 다음과 같다.

$$h_{ij}(k) = \begin{pmatrix} \sum_{c=1}^{l_b} b_{ij}(k,c), & \text{for } p_{ik} = p_{jk} \\ l_b, & \text{for } p_{ik} \neq p_{jk} \end{pmatrix} \quad (7)$$

수식에서 l_b는 블록의 길이를, $b_{ij}(k,c)$는 개체 i와 j의 k번째 블록의 c번째 비트가 같으면 1을, 다르면 0을 출력하는 함수를 의미한다. 그리고 p_{ik}는 개체 i의 k번째 적항입력비트를 의미하는데, 적항입력 제어신호가 다른 경우에는 블록의 사용여부가 다른 것이므로 최대 거리를 적용하여 l_b값이 주어진다.

4.3 분산 룰렛휠 선택

유전자 알고리즘에서 사용되는 선택 연산자 중 가장 일반적인 알고리즘은 룰렛휠선택(roulette wheel selection) 방법이다. 이 방법은 적합도에 비례하는 넓이를 가지는 룰렛을 만들어서 선택 연산을 하는 방법으로, 룰렛상의 n개의 개체 중에서 1개의 개체를 선택하는 과정을 n'번 수행한다[1]. 이때 각 개체의 선택될 확률은 s_i이며 그 값은 다음 수식과 같다.

$$s_i = \frac{sf_i}{\sum_{k=1}^{n} sf_k}, \qquad n' = n \times p_s \quad (8)$$

여기서 p_s는 교체될 세대의 비율을 정하는 값이며, n'는 다음 세대의 부모로 선택된 개체의 수를 의미한다. 이 선택방법은 우수한 개체일수록 더 많이 선택되도록 하여 진화를 유도하는 중요한 역할을 한다. 그러나 룰렛휠 선택 방법은 좋은 개체만 선택되거나 좋지 않은 개체가 선택되는 등 해가 골고루 선택되지 않아서 좋은 유전자를 잃는 선택 오류(selection noise)를 유발할 수 있다. 선택 오류를 줄이기 위한 방법은 여러 가지가 있다[19]. 대표적으로 Stochastic universal sampling을 들 수 있는데, 룰렛을 조금씩 다른 n'개의 영역으로 나눈 다음 그 속에서만 룰렛 화살이 꽂히도록 한 방법이다. 즉, 쉽게 말해 둥글게 배열된 n'개의 화살을 동시에 쏘아서 룰렛에 쏘아 맞추는 방법이다. 이렇게 하면 특정 영역에 너무 많은 화살이 꽂이는 선택 오류를 줄일 수 있는 것이다.

본 논문에서는 이와 비슷한 효과를 얻으면서도 하드웨어 환경에서 사용하기 쉽도록 하기 위해서 분산 룰렛휠 방법을 고안하여 사용하였다. 제안하는 방법은 하나의 룰렛휠을 만들어서 뽑는 기존의 방법과는 달리 개체별로 다른 크기의 룰렛을 만들어서 선택의 기회를 골고루 주는 방법이다. 각 개체의 룰렛은 s_i만큼의 허가 영역과 $1-s_i$만큼의 불허 영역을 가지며, 이 룰렛에 의해 그 개체의 선택 여부가 결정된다. 즉, 임의의 개체를 뽑은 뒤, 룰렛 시험을 거쳐서 선택된 개체의 수가 n_i'가 될 때까지 이를 반복하는 방법이다. 이때 각 개체의 선택 허용 확률 s_i는 다음과 같다.

$$s_i = \frac{sf_i}{\max_{j=1,2,\cdots,n} sf_j} \quad (9)$$

수식을 보면 s_i 값이 기존 방법에 비해 상당히 크고, 최대적합도를 가진 개체의 경우에는 1이 되어 항상 선택됨을 알 수 있다. 기존의 룰렛휠 선택은 뽑히는 확률

이 적합도에 의해서만 좌우되는 방법이지만, 분산 룰렛휠 방법은 선택 시도 확률은 동등하되, 선택 허용 확률만 차이를 주어서 좀더 고른 개체가 안정적으로 선택되도록 한 방법이다. 아래 표 1은 실제 분산 룰렛휠 방법을 사용했을 때와 사용하지 않았을 때의 실험 결과이다.

표 1의 실험 결과를 보면 엘리트 유지전략을 사용했을 경우와 사용하지 않았을 경우 모두 분산 룰렛휠의 발견 세대가 앞서고 있다. 그리고 표준 편차도 작게 나왔는데 이는 그만큼 안정적인 성능이 보장됨을 의미한다. 특히 엘리트 유지전략을 사용하지 않았을 경우에는 발견세대와 표준 편차가 지나치게 크게 나와서 상당한 실험 시간을 요구했다. 따라서 본 논문에서는 모든 실험에서 분산 룰렛휠 선택 방법을 사용하여 좀더 효율적으로 실험을 수행하였다.

4.4 교차와 돌연변이

교차 방법으로는 일반적으로 많이 쓰이는 1점 교차, 일정 교차(uniform crossover) 방법과 함께 블록단위 일정교차를 사용하였다. 1점 교차는 1점을 기준으로 염색체를 교환하는 방법이고, 일정교차는 두 염색체에 대해 비트단위의 교차여부를 확률적으로 정하는 방법으로 유전자 교환이 빠르게 일어나는 특성이 있다. 블록단위 일정교차는 염색체 구조상 가지게 되는 블록 단위별 특성을 유지하면서 교차할 수 있도록 고안한 방법이다. 블록단위 일정교차의 실행 방법은 그림 9와 같다. 즉, 블록 별로 마스크를 정하고 일정 교차를 하며, 해당 블록의 적항입력비트도 똑같은 마스크를 적용하도록 한 것이다. 본 논문에서는 유전자 교환을 가속시키고 블록단위의 유전자 정보를 유지시키기 위해, 매번 임의로 선택되는 스위치에 따라 세 가지 교차 방식이 골고루 사용되도록 하였다.

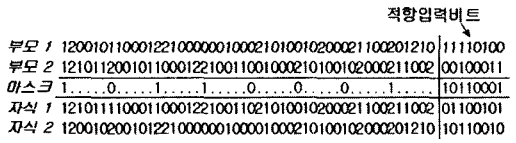


그림 9 블록 단위 일정교차의 예

돌연변이는 각 유전자에 대해 돌연변이 확률에 의해 대립형질로 값이 바뀌도록 하는 비트플립 돌연변이를 문제에 맞게 수정하여 사용하였다. 유전자 '0'과 '1', '2'가 모두 독립적인 대립형질로 간주되도록 하였으며, 비트 단위 돌연변이와 블록 단위 돌연변이가 임의로 선택되어 수행되도록 하였다. 블록단위 돌연변이는 적항비트 하나의 값을 바꾸어 바뀐 값이 '1'인 경우에만 그에 해당되는 블록의 비트열을 모두 임의의 대립형질로 바꾸도록 하는 방법이다.

5. 종분화된 진화 하드웨어 실험 결과

본 절에서는 제안하는 방법의 유용성을 알아보기 위해 단순화된 GAL 회로 시뮬레이션을 이용한 진화 실험 결과를 보인다. 이때 목표 하드웨어는 기존에 GAL 하드웨어의 실험대상으로 많이 이용된 그림 10과 같은 6멀티플렉서를 사용하였다[4]. 이 회로는 4개의 입력과 2개의 선택입력 1개의 출력비트를 가진 회로이며, 그림 10의 오른쪽 데이타와 같은 입출력 패턴을 가진다.

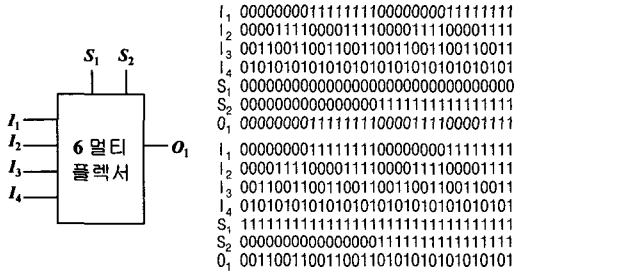


그림 10 6멀티플렉서(왼쪽)와 입출력 패턴(오른쪽)

5.1 종분화된 진화 하드웨어

모든 실험은 표 2에 나타나 있는 환경 변수를 사용하여 수행되었다. 사용된 값은 경험적으로 우수한 값을 선택하여 사용하였다. 환경 변수에서 '돌연변이 개체 발생률'은 돌연변이 될 개체를 뽑을 확률값으로 사용하였으며, '목표 해발견율'은 종분화를 통해 얻고자 하는 해의 비율을 의미한다. 여기서는 집단의 크기가 50이고 0.3을

표 2 환경 변수

파라미터	값
집단의 크기	50
돌연변이 개체 발생률	0.02
교차율	0.4
공유거리	3.0
목표 해발견율	0.3

표 1 룰렛휠과 분산 룰렛휠 선택 방법의 성능 비교 (15번 실험한 뒤 평균)

	엘리트 유지전략 사용 안 함		엘리트 유지전략 사용	
	룰렛휠	분산 룰렛휠	룰렛휠	분산 룰렛휠
발견 세대	11,111.5	381.3	122.5	114.6
표준 편차	6,636.8	235.1	61.6	54.9

사용하였으므로 15개의 해를 발견할 때까지 진화를 수행한다.

먼저 종분화를 통한 다양한 해의 탐색 기능을 조사하기 위하여, 염색체의 모양이 동일한 해는 1개의 해로 간주하여 해의 수를 조사하였다. 그림 11을 보면 종분화에 의해 해의 수가 점차적으로 증가하는 것을 알 수 있다.

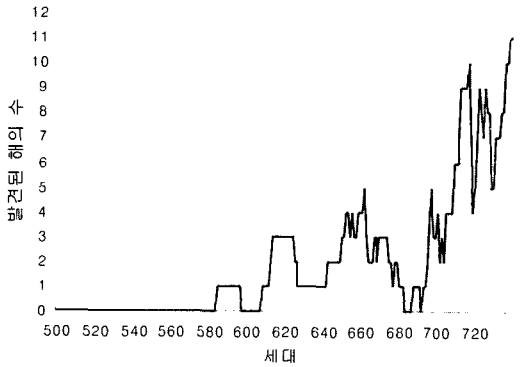


그림 11 종분화 GA의 세대에 따른 해의 수 변화 그래프

그림 12는 적합도 공유를 수행한 경우와 그렇지 않은 경우의 세대별 평균 적합도를 비교하기 위해서 일반적인 경우를 나타낸 그래프이다. 공유된 적합도가 순수한 적합도에 비해 기복이 크고 낮은 값의 분포를 보이고 있어서 적합도 공유가 활발히 이루어지고 있음을 알 수 있다.

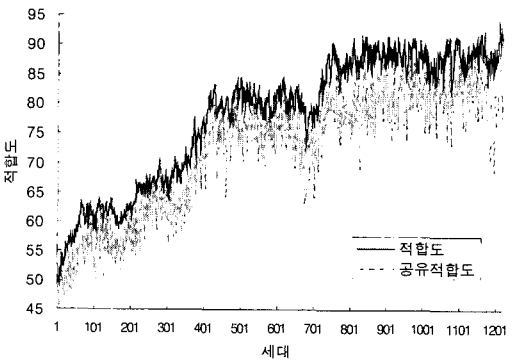


그림 12 적합도와 공유적합도의 세대별 집단 평균값 변화 그래프

그림 13은 세대별 최고 적합도를 나타낸 그래프이다. 첫 번째 해는 901세대에 발견되었고, 1,216세대에 이르러 목표 해발견율에 도달하여 진화가 끝났음을 알 수 있다.

이 실험에서 1,216세대에서 발견된 해의 수는 총 15개였고, 그 중 염색체의 모양이 동일한 해를 제거하여

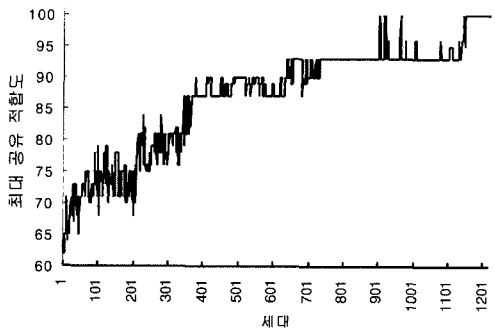
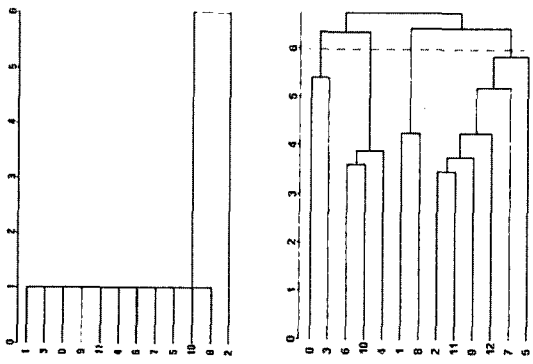


그림 13 공유 적합도의 세대별 변화 그래프

13개의 해를 얻을 수 있었다. 이 13개의 개체는 다른 형태를 가진 하드웨어 구조이지만 모두 6 멀티플렉서의 기능을 가지고 있는 다양한 종의 하드웨어이다. 해의 다양성을 알아보기 위해서 13개의 개체를 단일거리 클러스터링(single linkage clustering) 한 뒤 그림 14에 덴드로그램(dendrogram)으로 나타내었다. 그림 14에서 왼쪽 덴드로그램은 표준 GA를 통해 15개의 개체를 얻은 뒤 염색체 모양이 동일한 해를 제거하여 나타낸 결과이다. 개체 10을 제외하고는 모두 거리가 1인 유사한 개체임을 알 수 있다. 그에 비해 오른쪽에 나타낸 종분화된 개체들의 덴드로그램에서는 개체들이 최소거리 3.3에서 최대거리 6.7까지 멀리 떨어져 있음을 알 수 있다. 본 논문에서 사용된 길이 계산 방법에 따르면 블록 하나가 완전히 다를 경우 2.4의 거리를 가지고 최대 8.5의 거리를 가질 수 있음을 볼 때, 종분화된 개체들은 적어도 한 블록 이상이 전혀 다르면서 다양한 특성을 지닌 하드웨어들임을 알 수 있다.

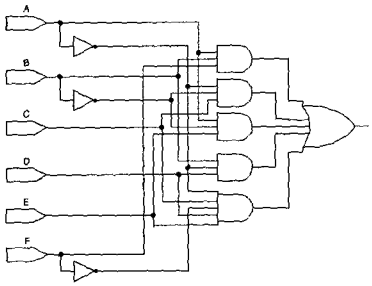


(a) 표준 GA (b) 종분화 GA

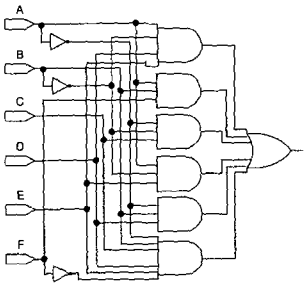
그림 14 얻어진 해에 단일거리 클러스터링을 적용한 결과 얻어진 덴드로그램

종분화된 개체들의 다양성을 좀더 자세히 알아보기 위해서 회로 구조도를 그려서 살펴보기로 한다. 그림 15

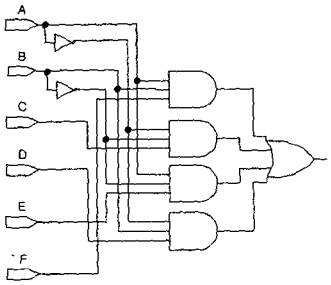
는 그림 14의 덴드로그램에서 거리 6인 지점의 점선을 기준으로 4그룹으로 나눈 뒤 그룹의 대표 개체를 하나씩 뽑아서 회로 구조를 나타낸 것이다. 각기 사용된 입력의 수나 게이트의 수, 구조가 다름을 알 수 있다.



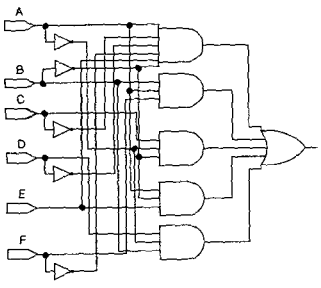
(a) 그룹 1의 개체 0



(b) 그룹 2의 개체 6



(c) 그룹 3의 개체 1



(d) 그룹 4의 개체 2

그림 15 각 종의 대표 개체의 회로 구조도

그림 15의 회로를 살펴보면 각 회로들은 특징적인 부분도 있지만 동일한 회로구조도 가지고 있다. 예를 들어 개체 6의 2번째에서 5번째까지의 AND 게이트만 놓고 보면 개체 1의 구조와 동일하다. 이처럼 모든 회로는 같은 부분 회로를 가지고 같은 기능을 내포하고 있다. 이는 각 개체의 AND 게이트 구조를 간단히 나타낸 그림 16를 보면 잘 알 수 있다.

개체 0	개체 6	개체 1	개체 2
ABF	$A\bar{B}DE$	ABF	$A\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}$
$\bar{A}\bar{B}C$	ABF	$\bar{A}\bar{B}C$	ABF
$A\bar{B}E$	$\bar{A}\bar{B}C$	$A\bar{B}E$	$\bar{A}\bar{B}C$
$\bar{A}BD$	$A\bar{B}E$	$\bar{A}BD$	$A\bar{B}E$
$\bar{A}CDE\bar{F}$	$\bar{A}BD$		$\bar{A}BD$
	$\bar{A}BDE\bar{F}$		

그림 16 각 개체별 AND 조합 부분 비교(회색박스로 칠해진 곳은 같은 구조를 가지고 있다)

결국, 각 개체에서 동일하게 가지고 있는 부분 회로를 뺀 나머지 회로들은 그 회로만의 부가 기능을 가진 회로라고 볼 수 있다. 예를 들어 개체 0의 부가 회로는 $\bar{A}CDE\bar{F}$이다. 종분화를 통해 얻어진 다양한 해 속에서 찾아진 이러한 회로들은 미처 예상하지 못했던 유용한 부가 기능일 가능성이 있다. 다양한 하드웨어 구조를 얻는 것은 원하는 기준에 따라 가장 좋은 하드웨어를 선택할 수 있다는 또 다른 장점도 있다. 이미 다양한 해를 얻은 상황에서 필요한 것은 어떤 기준으로 뽑을 것인지 정하는 것뿐이다. 예를 들어 개체 1은 가장 적은 하드웨어 자원을 사용하고 있는데, 최적화된 구조를 기준으로 뽑는다면 개체 1이 선택될 것이다.

그 외에 다양한 하드웨어를 얻음으로써 얻을 수 있는 이점으로 고장극복(fault-tolerance) 기능이 있다. 예를 들어 OLMC의 첫 번째 입력부의 기능이 고장이 났을 경우 개체 1은 정상적인 동작을 할 수 없다. 그러나 고장 부위를 사용하지 않게 한 뒤 개체 2를 사용하면 추가적인 하드웨어의 진화 없이 고장상황을 극복할 수 있을 것이다.

5.2 종분화의 탐색 효율성

종분화 알고리즘의 탐색 능력을 알아보기 위해서 진화 하드웨어의 해 발견 세대를 표준 GA와 비교해 보았다. 앞서 말했듯이 종분화 GA는 탐색 영역이 넓게 유지되기 때문에 진화 하드웨어와 같이 기복이 심하고 함정이 많은 복잡한 해공간을 가진 경우 더 좋은 성능을 발휘할 가능성이 높다. 진화의 속도를 비교하기 위해 첫 해를 발견할 때까지 진행되도록 하고, 한계세대를 6,000으로 두어 2가지 평가 회로에 대해 실험한 결과를 표 3에 표현하였다(표 3). 또한 6멀티플렉서에 기존 툴랫휠

표 3 종분화 GA와 표준 GA의 해 발견 성능 비교 (50번 실험 후 평균, '-'는 진화 실패)

	분산 룰렛휠 선택				기존 룰렛휠 선택, 6멀티플렉서			
	6멀티플렉서		모듈 A		엘리트 유지 적용		엘리트 유지 미적용	
	표준GA	종분화GA	표준GA	종분화GA	표준GA	종분화GA	표준GA	종분화GA
발견 세대 평균	1,147.16	747.46	565.76	503.88	-	-	-	-
표준 편차	1,061.35	584.70	627.95	440.74	-	-	-	-
가장 빠른 발견 세대	143.00	77.00	76.00	76.00	-	-	-	-
가장 늦은 발견 세대	5,373.00	2,485.00	3,382.00	2,436.00	-	-	-	-
수행시간 (초)	435.52	610.98	175.49	400.98	-	-	-	-

표 4 성능 평가 실험에 사용된 모듈 A의 입출력 패턴

측정부위		디지털 신호
입력부	I_1	00000000000000000000000000000000111111111111111111111111111111111
	I_2	00000000000000001111111111111111000000000000000011111111111111111
	I_3	00000000111111110000000011111111000000001111111100000000111111111
	I_4	00001111000011110000111100001111000011110000111100001111000011110
	I_5	00110011001100110011001100110011001100110011001100110011001100110
	I_6	01010101010101010101010101010101010101010101010101010101010101010
출력부	O_1	00000000111111110000111100001111001100110011001100000000000000000

선택 연산자와 엘리트 유지 전략을 각각 적용했을 때와 하지 않았을 때에 대한 실험 결과도 서술하여 분산 룰렛휠의 효과를 살펴보았다. 이때 실험에서 사용된 '모듈 A'는 6멀티플렉서와 복잡도가 유사한 임의의 모듈로서, 단순화된 GAL로 구현 가능하도록 입출력 패턴을 정의한 회로이다. '모듈 A'의 입출력 패턴은 표 4에 나타내었다.

우선 분산 룰렛휠 선택을 사용하여 비교한 결과를 보면 종분화 GA가 표준 GA보다 탐색 능력이 우수함을 알 수 있다. 종분화 GA는, 6멀티플렉서의 경우 발견 세대가 35%정도 앞서고, 표준 편차는 45% 향상되었으며, 모듈 A의 경우 발견 세대는 11%, 표준편차는 30% 정도 향상되었다. 이것은 종분화 기법을 사용한 경우 표준 GA를 사용한 경우에 비해 우수한 탐색 성능을 안정적으로 보여줌을 의미한다. 특히, 표준 GA의 경우에는 늦게 발견할 경우 5,373세대까지 진행하기도 하여서 유전적 해쏠림 현상을 짐작할 수 있었다.

표 3에서 제안된 분산 룰렛휠 선택을 사용하지 않고 기존의 룰렛휠 선택 방법과 엘리트 유지 적용 방법을 사용한 경우에는 제한된 시간 내에 해의 발견을 한 번도 성공하지 못하고 있다. 이는 선택 노이즈가 큰 기존 룰렛휠 방법을 사용해서는 복잡한 진화 하드웨어 설계 문제를 해결하기가 어렵다는 것을 나타낸다. 모듈 A일 경우와 한계세대를 늘린 경우에 대해서도 진화를 시도해 보았으나 수렴에 성공하는 경우가 드물어서 비교가 불가능하였다.

결국 진화 하드웨어에서의 탐색은 선택 노이즈가 적은 방법을 사용하면서 종분화 알고리즘을 적용하는 것이 유리함을 알 수 있다. 하지만 사용된 종분화 GA의 경우 추가되는 연산으로 인해 전체 실행 속도는 느리다. 위의 실험에서 종분화 GA가 표준 GA에 비해 세대당 2.1배정도의 시간을 소모하여 최초해 발견 시간 평균이 약 39.6% 정도의 늦는 결과를 보여주었다. 하지만 표준 편차의 감소로 인한 안정성 증가 및 독창적인 부가 기능 발견 가능성, 여러 해의 동시 획득 등의 장점을 얻을 수 있으며, 종분화 진행시 대부분의 시간을 차지하는 개체간 거리 계산 부분은 하드웨어에 상에서 병렬수행이 가능한 부분이기 때문에 실제로는 좋은 성능을 기대할 수 있다고 보여진다.

5.3 종분화 진화된 회로의 다양성과 그 효용성

종분화 알고리즘에 의해 얻어진 하드웨어들이 가진 다양성과 그에 따른 성능의 우수성을 확인하기 위해 여러 진화 하드웨어 디자인 문제에 대해 비교 실험을 하여 보았다. 이 실험은 5.1절과 5.2절에서 다룬 진화 실험을 통해 얻어진 회로들을 대상으로 하였으며, 사용된 회로들은 모두 표 5와 같은 오류 가능성을 가지고 있다고 가정하였다. 표 5의 가설을 적용한 예를 들면, 그림 15의 회로 (a)의 경우 표 3의 1번 가설에 의해 입력 a와 입력 c에서 0.00001의 확률로, 그리고 3번 가설에 의해 입력 e에서 0.00005의 확률로 오류를 일으킬 수 있다.

그림 17은 실험 결과 얻어진 결과를 그래프로 나타낸 것이다. 그림에서 simple GA는 표준 GA를 통해 얻은 하드웨어의 극한 테스트 오류 결과를 나타내고, speciated GA는 논문에서 제안하여 사용된 종분화 GA를 이용하여 얻은 약 15개의 회로에 대해 극한 테스트를 한

표 5 디지털 회로 극한 환경 테스트 조건

※ OR 게이트의 8개의 입력을 a, b, c, d, e, f, g, h라고 할 때, 회로를 한계 주파수로 5 시간 동안 사용한 뒤 계속하여 사용할 경우
1. 3 AND 게이트는 a, c에서 0.00001의 확률로 오류를 일으킨다.
2. 4 AND 게이트는 d, g, h에서 0.00003의 확률로 오류를 일으킨다.
3. 5 AND 게이트는 b, c, e, f에서 0.00005의 확률로 오류를 일으킨다.
4. 6 AND 게이트는 a, c, e, g, h에서 0.00007의 확률로 오류를 일으킨다.

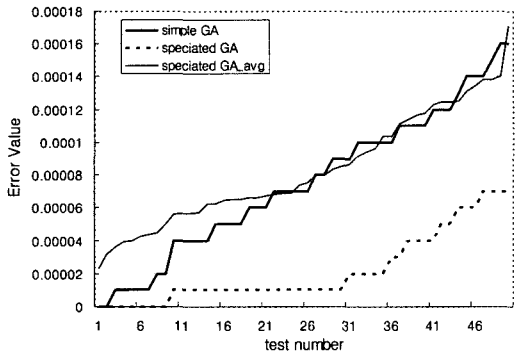


그림 17 극한 환경 테스트 결과 비교 그래프 (50회 실험 결과)

뒤 가장 우수한 회로를 선택한 결과를 나타내며, speciated GA_avg는 종분화된 회로들의 평균 오류값을 나타낸다. 그리고 성능 차이가 잘 구분되도록 오류가 0인 것부터 오름차순으로 정렬하여 표현하였다.

그림 17의 결과를 보면 speciated GA와 simple GA에 비해 오류가 적게 나타나고 있음이 명백하다. 한 번의 진화에서 얻은 다양한 회로(약 15개)에서 테스트를 거쳐 가장 좋은 회로를 선택하기 때문에 얻어지는 결과이다. simple GA는 이같은 효과를 얻기 위해서 약 15번 이상의 진화를 되풀이해야 할 것이다. 한 번의 진화로는 다양한 여러 회로를 얻기 어렵고 여러 번 수행하더라도 다양한 결과가 나오기 힘들기 때문이다. speciated GA_avg는 종분화를 통해 얻은 각각의 실험에서 얻은 약 15개 회로의 평균 오류값을 나타내는데, 표준 GA의 결과와 비슷한 경향을 보이고 있다. 따라서 종분화된 개체 하나하나는 표준 GA에 의해 얻은 한번의 실험 결과와 큰 차이가 없음을 알 수 있다.

표 6은 극한 환경 테스트 결과를 수치적으로 분석한 결과이다. 실험에서 발견된 무오류 회로의 수는 표준GA가 2번인데 비해, 종분화 GA는 9번으로 상대적으로 높았다. 이 같은 결과는 표준 GA에서 획득한 회로가 50개인데 비해 종분화 GA는 766개의 회로를 얻을 수 있었기 때문에 나타난 것이다. 표 3에서 6MUX를 발견하는 평균 시간이 표준 GA는 약 436초, 종분화 GA는 611초로 종분화 GA가 좀 느렸지만, 이를 무오류 회로 발견율로 나누어 무오류 회로를 발견하는데 필요한 시간을 역으로 계산해 보면, 해발견 평균 시간의 비가 약 3.2 : 1로 종분화 GA가 효율적인 것으로 나온다. 또한 이것은 진화에 의한 시간만 계산한 결과이므로, 극한 테스트를 하는데 걸린다고 가정한 5시간을 더할 경우 훨씬 더 많은 차이가 생길 것이다.

6. 결론 및 향후 연구

본 논문에서는 진화 하드웨어를 효율적으로 수행하기 위해서 종분화 기법을 사용하는 것을 제안하고 그 성능을 평가하였다. 진화 하드웨어는 적합도 공간 분석 결과 기복이 심하고 함정이 많으며 여러 개의 해가 존재하는 상당히 복잡한 해공간을 가지고 있어서, 기존 GA로는 진화에 어려움이 많다. 따라서 다양성이 우수한 종분화 기법을 적용할 경우 탐색 성능 향상을 기대할 수 있었으며, 실제 실험 결과 발견 세대가 앞서고 탐색 안정성이 향상되었다. 그리고 종분화를 통해서 한번의 진화로 얻은 여러 종의 하드웨어가 다양한 구조를 가지고 있음을 확인하였고, 공통적인 기능 부분 이외에도 독특한 부가 기능이 존재하고 있어 더 좋은 성능을 기대할 수 있음을 알 수 있었다. 본 논문에서 가정하여 수행된 극한 테스트 결과에서도 설계 단계에서 고려하지 않았던 오류를 피하는 회로를 발견할 확률이 더 높음을 확인할 수 있었다. 향후 이러한 부가 기능의 연구를 통해 독창적이고 우수한 성능의 하드웨어 설계가 가능할 것이다. 또한 본 논문에서 제시한 실험 결과는 비교적 단순한 하드웨어에 대한 결과를 통해 가능성만을 제시하고 있는데, 향후에는 복잡한 하드웨어에 대한 연구를 통해 좀

표 6 종분화 GA와 표준 GA의 극한 환경 테스트 결과 비교 (50회 실험)

	표준GA	종분화GA	비율
무오류 회로 발견율	2 / 50 = 0.04	9 / 50 = 0.18	1 : 4.5
평가한 회로의 수	50	766	1 : 9.3
해발견 평균 시간	435.52	610.98	1 : 1.4
무오류 회로 발견 예상 시간	10,888	3394	3.2 : 1

더 확실한 부가 기능을 찾을 필요가 있다.

최근에 진화 하드웨어 분야는 좀더 큰 기능을 좀더 빨리 얻기 위한 많은 연구가 수행되고 있다. 무어의 법칙(Moore's Law)에 따라 폭발적으로 증가되고 있는 하드웨어의 규모에 비해 인간의 디자인 기술과 소프트웨어의 발전은 더디기만 한 현실에 있어, 진화에 의한 하드웨어 디자인은 앞으로 더욱 필요한 기술이라 할 수 있다[20]. 이러한 현실에 있어 하드웨어 설계를 위한 진화 기법으로 종분화 방법이 유용하게 사용될 수 있을 것으로 기대된다.

참 고 문 헌

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[2] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Trans. on Systems, Man, and Cybernetics*, part C, vol. 29, pp. 87-97, February, 1999.

[3] D. Albert, "Evolutionary hardware overview," *http://citeseer.nj.nec.com/201089.html*, 1997.

[4] T. Higuchi, et al., "Evolvable Hardware with Genetic Learning," *Proc. of IEEE Int. Symposium on Circuits and Systems*, vol. 4, pp. 29-32, 1996.

[5] M. Sipper, et al., "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 1, pp. 83-97, 1997.

[6] A Thompson, *Hardware Evolution: Automatic Design of Elecronic Circuits in Reconfigurable Hardware by Artificial Evolution*, London: Springer-Verlag, 1998.

[7] T. Higuchi, et al., "Evolvable hardware," *Massively Parallel Artificial Intelligence*, pp. 398-421, MIT Press, 1994.

[8] W. Liu, et al., "ATM cell scheduling by function level evolvable hardware," *Proc. of the First Int. Conf. Evolvable Systems: From Biology to Hardware*, pp. 180-192, Tsukuba, Japan, October, 1996.

[9] T. Kalganova, "An Extrinsic Function-Level Evolvable Hardware Approach," *Proc. of the Third European Conf. on Genetic Programming*, Springer-Verlag, pp. 60-75, 2000.

[10] J. Torresen, "A divide-and-conquer approach to evolvable hardware," *Proc. of the 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware*, vol. 1478 of Lecture Notes in Computer Science, Springer-Verlag, pp. 57-65, Heidelberg, 1998.

[11] V. K. Vassilev, "Scalability Problems of Digital Circuit Evolution: Evolvability and Efficient Designs," *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware*, pp. 55-64, IEEE Computer Society, July, 2000.

[12] K. Deb and W. M. Spears, "Speciation methods," *Evolutionary Computation 2: Advanced Algorithms and Operators*, Institute of Physics Publishing, ch. 14, pp. 93-99, 2000.

[13] R. I. McKay, "Fitness sharing in genetic programming," *Genetic and Evolutionary Computation Conf.*, pp. 435-442, July, 2000.

[14] 황금성, 조성배, "종분화를 이용한 다품종 하드웨어의 진화", 정보과학회 봄 학술발표 논문집(B), 제28권, 제1호, pp.307-309, 2001.

[15] T. Back, "Introduction to evolutionary algorithms," *Evolutionary Computation 1: Basic Algorithms and Operators*, Institute of Physics Publishing, ch. 7, pp. 59-63, 2000.

[16] S. W. Mahfoud, "Niching Methods for Genetic Algorithms," *PhD thesis*, University of Illinois at Urbana-Champaign, 1995.

[17] P. Darwen and X. Yao, "Every niching method has its niche: Fitness sharing and implicit sharing compared," *Proc. of Parallel Problem Solving from Nature (PPSN) IV*, vol. 1141 of Lecture Notes in Computer Science, Springer-Verlag, pp. 398-407, 1996.

[18] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," *Proc. 3rd Int. Conf. Genetic Algorithms*, pp. 42-50, 1989.

[19] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," *In Proceedings of the Second Int. Conf. on Genetic Algorithms and their Application* (J. J.Grefenstette ed.), pp. 14-21, 1987.

[20] S. Trimberger, "Reconfigurable Devices in the 21st Century," *Presentations of Invited Speakers on the Third NASA/DoD Workshop on Evolvable Hardware*, July, 2001.

황 금 성

2000년 3월 연세대학교 컴퓨터과학과 졸업(학사). 2002년 3월 연세대학교 컴퓨터과학과 석사과정 졸업. 2004년 3월 연세대학교 컴퓨터과학과 박사과정 입학. 관심분야는 진화 하드웨어, 베이지안 기법, 개미집단 시스템

조 성 배

1988년 연세대학교 전산과학과(학사) 1990년 한국과학기술원 전산학과(석사) 1993년 한국과학기술원 전산학과(박사) 1993년~1995년 일본 ATR 인간정보통신연구소 객원연구원. 1998년 호주 Univ. of New South Wales 초청연구원. 1995년~현재 연세대학교 컴퓨터과학과 정교수. 관심분야는 신경망, 패턴인식, 지능정보처리