진화 하드웨어를 위한 종분화 알고리즘의 체계적 성능 평가

한승일[0], 황금성, 조성배
연세대학교 컴퓨터과학과
{dukedove, yellowg}@candy.yonsei.ac.kr, sbcho@cs.yonsei.ac.kr

A Systematic Evaluation of Speciation Algorithms for Evolvable Hardware

Seung-Il Han[0], Keum-Sung Hwang, Sung-Bae Cho
Dept. of Computer Science, Yonsei University

요 약

진화 가능한 하드웨어의 개발은 유전자 알고리즘의 새로운 가능성을 열어주었고 이에 적합한 다양한 방법이 제시되어 왔다. 하지만 일반적인 유전자 알고리즘으로는 Genetic drift가 생기거나 지역해에 빠지는 등 한계가 있기 때문에 이를 해결하기 위한 방안으로 종분화 알고리즘이 도입되고 있다. 현재까지 다양한 종분화 알고리즘이 소개되었는데 이들은 이전의 알고리즘과 비교하였을 때 높은 다양성을 유지하면서 더 좋은 해를 찾아낸다. 이 논문에서는 진화 하드웨어상에서 이러한 종분화 알고리즘들의 장단점 및 특징을 여러 비교기준을 통해 제시한다. 실험결과 Deterministic Crowding과 Struggle GA가 가장 좋은 성능을 나타내었다.

1. 서 론

진화 하드웨어(Evolvable Hardware: EHW)는 환경과 상호 작용하여 동적으로 자신의 구조와 행위를 변경시킬 수 있는 하드웨어를 말한다[1]. FPGA (Field Programmable Gate Array)와 같은 재구성이 가능한 하드웨어가 출현함에 따라 복잡한 회로 설계에 있어서 인간이 직접 설계하는 것보다 빠르고 회로내의 오류나 고장에 대해 보다 유연하게 대처할 수 있게 되었다. EHW에 대한 관심은 이러한 하드웨어의 출현과 함께 시작되었고 진화 방법으로 사용된 것이 유전자 알고리즘(Genetic Algorithm: GA)이다.

그런데 기존의 유전자 알고리즘(simple GA: sGA)은 하나의 최적해를 탐색하는 문제에 있어서는 우수한 결과를 보여주고 있으나, 여러개의 최적해를 가지며, 사용 환경에 대해 유연한 특성을 갖는 EHW에 적용하기는 적합하지 않다[2]. 이러한 EHW의 특성을 만족시키기 위한 방안으로 종분화 알고리즘(Speciation Algorithm)을 들 수 있다.

본 논문에서는 여러가지 종분화 알고리즘들과 기존의 유전자 알고리즘인 sGA를 다양한 방법으로 비교, 분석하여 각 종분화 알고리즘의 장단점과 특성을 보이고 EHW에 적합한 방법을 제시한다.

2. 종분화 알고리즘

기존의 유전자 알고리즘은 자연 선택과 자연 발생과정에 기초한 탐색 알고리즘으로 다수의 개체를 동시에 진화시켜 최적의 해를 찾는다. 종분화 알고리즘이란 유전자 알고리즘에 종분화 기법을 적용한 것으로 높은 다양성을 유지하면서 더 좋은 해를 얻을 수 있다[3].

2.1 De jong's Crowding

De jong's Crowding 기법은 crowding factor model로서 '전체 집단크기×세대간격(자손으로 선택될 비율)' 만큼의 개체가 선택되어져 교차와 돌연변이 과정을 거친다. 진화 과정을 거친 새로운 개체들은 기존 개체들 가운데 임의로 선택된 CF(Crowding factor)개의 개체중 가장 비슷한 개체와 교체된다[4].

2.2 Deterministic Crowding

S. Mahfoud가 제안한 방법으로 Dejong's crowding의 변형이다[5]. 부모집단의 모든 개체를 임의로 짝을 지어 진화연산을 하여 각각 2개의 자손을 만들어 낸다. 각각의 자손을 부모와 비교하여 가까운 부모와 자손을 묶는다. 이렇게 묶인 두 개의 개체중 적합도가 높은 쪽이 남겨진다.

2.3 Restricted Tournament Selection

Harik이 제안한 방법으로 두 개의 개체가 임의로 선택되고 진화연산을 통하여 하나의 새로운 개체를 만든다. 이 새로운 개체는 기존의 개체중 임의로 선택된 CF개의 개체와 비교하여 거리가 가장 가까운 개체와 경쟁한다. 새로운 개체의 적합도가 기존의 개체보다 더 높으면 새로운 개체로 교체된다[4].

2.4 Fitness Sharing

Genetic drift 현상을 방지하기 위해 공유반경(sharing radius)내의 모든 개체들이 동일한 공유 적합도를 가지게 하는 방법으로 Holland가 제안했다[4]. 본 논문에서는 Explicit 적합도 공유 방식과 Implicit 적합도 공유 방식을 사용하였다. Explicit 방식은 공유적합도를 구하여 공유반경내의 모든 개체가 이 적합도를 공유하게 하는 방식이고 Implicit 방식은 σ개의 샘플 개체중에서 적합도가 가장 높은 것의 적합도를 emphasizing rate만큼 높여주어 선택될 확률을 높여주는 방식이다.

2.5 Struggle Genetic Algorithm

Crowding의 변형으로 우선 임의로 부모를 선택한 뒤 이들을 교배 및 돌연변이 연산하여 개체 C'을 만든다. 부모집단의 모든 개체중 C'과 가장 거리가 가까운 개체와 경쟁하여 C'의 적합도가 더 높으면 교체되는 방식으로 T. Gruninger가 제안했다[4].

3. 종분화 알고리즘의 실험적 평가

3.1 벤치마크 함수의 최적화 실험

최적화 실험에서 사용할 De jong의 제 1함수는 대표적인 벤치마크 함수로 다음과 같다.

$$f(x) = \sum_{i=1}^{n} x_i^2 \qquad -5.12 \le x_i \le 5.12$$
$$\text{global minimum:} \quad x_i=0 \text{ 일때 } f(x)=0 \tag{1}$$

본 논문에서는 염색체를 세부분으로 나누어서 십진수로 환산한 다음 이 세개의 십진수를 x_i에 대입하여 적합도를 계산한다(n=3).

Simple GA나 DC, Fitness sharing등이 한 세대마다 전체집단을 진화시키는 것과 달리 Dejong's Crowding, RTS, struggle GA는 한 세대에 1개체 또는 소량의 개체를 진화시키는 방식이기 때문에 이들을 서로 비교할 때 세대수로 비교하는 것은 적당하지 않다. 그래서 본 논문에서는 표 1과 같이 적합도 판정횟수를 그 기준으로 삼았다.

표 1. 세대당 적합도 판정 횟수(집단크기=100)

	sGA	Crowding	IFS	EFS	DC	RTS	Struggle
횟수	100	10(gap=0.1)	100	100	100	1	1

이 적합도 판정횟수를 기준으로 집단의 크기와 돌연변이율을 변화시켜 가면서 각 알고리즘이 최적해에 다다르는 평균 적합도 판정횟수를 비교하였다.

표 2. 최적해에 다다르는 적합도 판정횟수(10회 평균)

	집단크기	돌연변이율		
		0.01	0.005	0.001
Simple GA	100	6,320	6,310	7,620
	200	13,800	12,960	14,600
	평 균	10,060	9,635	11,110
Dejong's Crowding	100	39,620	35,852	53,260
	200	191,548	112,206	268,643
	평 균	115,584	74,029	160,951
IFS	100	7,380	7,820	14,210
	200	12,280	13,560	16,080
	평 균	9,830	10,690	15,145
EFS	100	23,960	36,240	27,830
	200	30,300	34,740	35,460
	평 균	27,130	35,490	31,645
DC	100	3,260	3,290	3,250
	200	6,200	5,200	5,800
	평 균	4,730	4,245	4,525
RTS	100	2,115	2,278	3,055
	200	3,861	4,045	4,052
	평 균	2,988	3,162	3,553
Struggle GA	100	4,694	5,033	6,836
	200	8,193	10,679	8,237
	평 균	6,443	7,856	7,536

표 2에 따르면 각 알고리즘의 집단크기가 증가함에 따라 적합도 판정횟수가 선형적으로 증가하는 것을 볼 수 있다. 이는 탐색공간이 넓어짐에 따라 더 많은 진화연산이 필요하기 때문이다. 돌연변이율에 있어서는 각 알고리즘마다 다른 결과를 나타냈는데 simple GA, Dejong's crowding, DC는 0.005에서 가장 좋은 성능을 보였고 나머지들은 0.01에서 가장 좋은 성능을 보였다.

또한 각 알고리즘을 최적 환경으로 설정한 다음 De jong 함수에서 해를 얼마나 다양하게 찾을 수 있는지를 실험하였다. 실험에서 3개의 입력을 받기 때문에 테스트 함수는 모두 2^3=8개의 해를 가진다.

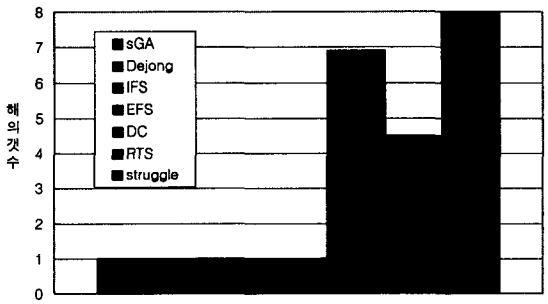

그림 1. 탐색된 최적해의 개수(10회 평균)

그림 1은 50,000번의 적합도 판정을 하는 동안 최적화된 각 알고리즘이 찾아내는 해 개수의 평균을 나타낸 것이다. 이 실험에서는 struggle GA가 8개의 해를 모두 찾아냄으로써 가장 좋은 성능을 보였고 그 뒤를 평균 6.9개의 해를 찾아낸 DC와 4.5개의 해를 찾아낸 RTS가 따랐다. 적합도 공유방식과 simple GA, De jong's crowding은 평균 1개의 해를 찾아내 상대적으로 저조한 성능을 보였다.

3.2 하드웨어 진화 실험

이 실험에서는 실제 하드웨어 모듈의 진화를 통해 각 알고리즘의 성능을 알아보고자 한다. 본 실험은 최적화된 상태의 각 알고리즘들로 표 3과 같은 실험 환경에서 실시되었다.

표 3. 실험 환경 변수

항 목	값
집 단 크 기	100
교 차 율	0.5
군집계수(CF)	3
세대간격(Gap)	0.1
최적해 유지	Yes

실험에 쓰인 하드웨어 모듈은 3mux, 6mux, 1bit adder, 2bit adder의 4가지이며 유전자 구조는 cell array구조를 사용하였다[6]. 각 모듈에 대해서 알고리즘별로 실험해본 결과 최적해에 다다르는 적합도 판정횟수(10회 평균)는 표 4와 같다.

표 4. 하드웨어 진화 실험에서의 적합도 판정 횟수(-:탐색실패)

	3mux	6mux	1bit adder	2bit adder
Simple GA	1,410	-	1,320	-
Crowding	3,831	-	-	-
IFS	910	-	1,870	-
EFS	910	-	2,280	-
DC	590	-	710	-
RTS	669	-	1,348	-
Struggle GA	608	-	862	-

이 표에서 보듯이 Struggle GA와 DC가 비교적 가장 빠르게 최적해를 찾아내었다. 이 실험은 적합도 판정횟수를 최대 50,000번으로 제한해서 실시하였는데 이 범위내에서는 어떤 알고리즘도 6mux와 2bit adder에 대해서는 최적해를 발견해 내지 못했다. 그리고 한계를 늘려 실험해 본 결과 DC의 경우에는 2bit adder를 약 25만번, 6mux를 68만번의 적합도 판정을 거쳐 최적해를 찾아내는 성능을 보였다.

하지만 이러한 결과만으로는 각 알고리즘이 종분화 알고리즘으로서 얼마나 다양한 해를 찾아내는가에 대한 결과는 얻을 수 없다. 그래서 이번에는 각 알고리즘이 1bit adder에 대해서 얼마나 많은 최적해를 찾아내는지를 실험하였다.

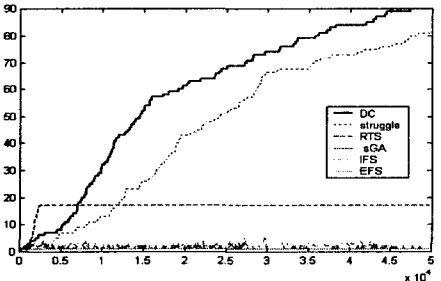

그림 2. 적합도 판정횟수에 따른 최적해의 개수(10회 평균)

그림 2는 총 5만번 적합도를 판정하는 동안 각 알고리즘이 찾아내는 최적해의 개수를 나타낸 것이다. DC와 Struggle GA는 해가 꾸준히 증가하는데 반해 RTS는 일정수에 다다르면 수렴한다. 이는 그림 3에서 볼 수 있듯이 다양성(diversity)[7]이 Struggle GA나 DC에 비해 RTS가 현저히 낮기 때문이다.

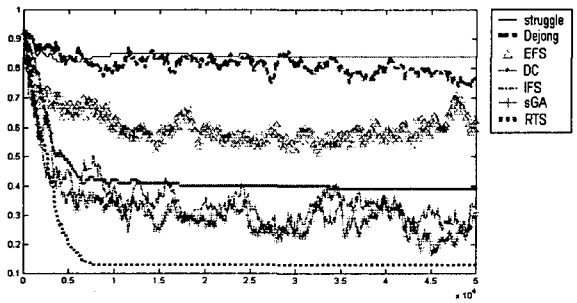

그림 3. Diversity 비교(10회 평균)

이러한 양상은 그림 4에서도 볼 수 있는데 평균적합도가 DC와 Struggle GA의 경우에는 차츰 증가하는데 비해 RTS는 급격히 증가하여 조기에 평균적합도 1에 도달한다. 이는 다양성이 낮기 때문이다. 그 이외의 알고리즘은 적합도의 비교없이 바로 교체되기 때문에 평균적합도가 증가하지 않고 진동한다.

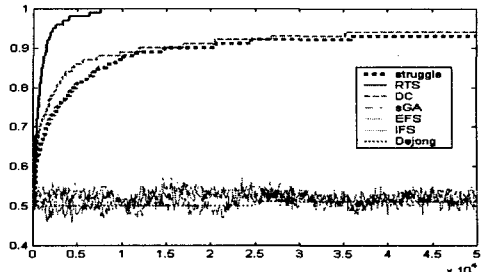

그림 4. 평균적합도의 변화(10회 평균)

4. 결론 및 향후 과제

본 논문에서는 여러 가지 종분화 알고리즘의 비교를 통해 각각의 장단점과 특성을 실험을 통하여 알수 있었다. DC와 Struggle GA가 가장 좋은 성능을 보이는 가운데, DC는 가장 빠르게 최적해를 찾으나 diversity가 떨어지는 반면 struggle GA는 탐색속도는 DC보다 느리나 diversity는 가장 좋게 나타났다. 그러나 본 실험은 한정된 모듈을 대상으로 이루어진 것이기 때문에 보다 다양한 모듈을 대상으로 연구할 필요가 있다.

5. 참고문헌

[1] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE T. SMC(C),* 29(1), pp. 87-97, February 1999.

[2] 이재성, 황금성, 조성배, "하드웨어 진화를 위한 Niching 방법," 한국정보과학회 봄 학술발표 논문집(B), 29권, 1호, pp. 232-234, 2002.

[3] 황금성, 조성배, "생명정보학에서의 거대규모 특징추출을 위한 종분화 GA의 활용," 한국정보과학회 봄 학술발표 논문집(B), 29권, 1호, pp. 229-231, 2002.

[4] T. Gruninger and D. Wallace, "Multimodal optimization using genetic algorithms: An investigation of a new crowding variation and the relationship between parental similarity and the effect of crossover," *MIT CADlab - Technical Report* 96.02, 1996.

[5] S.W. Mahfoud, "Niching method," *Evolutionary Computation 2: Advanced Algorithms and Operators,* Institute of Physics Publishing, pp. 87-92, 2000.

[6] 황금성, 조성배, "종분화를 이용한 다품종 하드웨어의 진화," 한국정보과학회 봄 학술발표 논문집(B), 28권, 1호, pp. 307-309, 2001.

[7] C. Fernandes, "A study on non-random mating and varing population size in genetic algorithms using a royal road function," *Proc. of 2001 Congress on Evolutionary,* pp. 60-66, 2001.