2016 강서학생탐구발표대회 탐구보고서

유전 알고리즘 프로그래밍 : 큐브와 삼목 AI	출품번호
	미기재

출품 부문	산업・에너지

201. 03 . 12.

소 속 청	학 교 명	학 년	성 명
강서	염경중학교	2	강준서
강서	염경중학교	2	윤준서

목 차

I. 서론

ⅰ. 개요

다윈이 〈종의 기원〉을 출판하여 세계의 역사, 신학, 그리고 과학을 뒤집어 놓았던 해로부터 벌써 158년이 지났다. 생명체들은 태어나고, 생존을 위해 경쟁한다. 그리고 유성 생식을 통해 자신의 유전자를 후대에 남기는데, 이 과정에서 상동 염색체들이 일부분 섞인다. 하지만 그 섞임은 인간과 생명체들이 살아남는 것에 있어 굉장히 필수적인 역할을 하였다. 유전자의 혼합은 곧 변이를 낳았고, 변화하는 환경 속에서 유리한 변종들은 생존 경쟁에서 승리하여 종족 전체의 진화를 불러왔다. 이것이 인류와 생명체들이 변화하는 환경 속에서도 살아남을 수 있었던 가장 강력한 무기, '진화'이다

맨해튼 프로젝트로 알려져 있고, 노이만 구조를 고안한 컴퓨터의 아버지 존 폰 노이만은 컴퓨터의 자가복제 능력에 주목하였고, 또 두려워했다. 노이만이 인공지능에 관련된 연구를 할 여력과 능력이 충분함에도 죽기 직전까지 부정하고 미루었던 것은 이 때문인데, 컴퓨터의 자가복제란 앞에서 설명한 진화, 즉 도킨스의 이기적 유전자 이론에서도 설명되었던 수프 속의 우연과 그 산물을 의미하는 것으로, 노이만이 두려워했던 것은 컴퓨터가 자신을 복제하며 발생하는 우연-변이, 그 변이로 인한 진화였다. 노이만은 결국 이 '오토마타 이론'을 미완성으로 남긴 채 세상을 떠나고 말았다.

하지만 오토마타 이론은 그의 제자 아서 벅스에 의해 정리해서 발표되게 되고, 이것이 현대 인공지능 이론의 토대이다. 그리고 MIT에서 교편을 잡던 아서 벅스에 의해 키워진 제자가 바로 미국 최초의 컴퓨터사이언스 박사학위를 취득하게 되는 존 홀랜드이다.

존 홀랜드는 스승의 스승, 즉 노이만의 오토마타 이론에 깊은 관심을 갖게 되었고, 동시에 한 책을 읽고 큰 감명을 받았다. 그 책은 현대 통계학의 아버지 로널드 피셔의 "The Genetical Theory of Natural Selection"으로, 홀랜드는 그 책을 읽고 유전학과 자연선택 이론에 빠지게 된다.

홀랜드는 자연선택 이론과 노이만의 오토마타 이론을 융합해 유전 알고리즘(Genetic Algorithm)을 고안한다. 허나 당시의 이론으로는 그의 유전 알고리즘을 이해할 수 있는 사람은 매우 드물었고, 20년이 넘도록 외면당했다. 그의 책은 겨우 2000부 남짓만이 팔렸으며 관련 논문은 겨우 20편 남짓이었는데, 이마저도 그의 몇몇 제자들이 쓴 것이다.

허나 시간이 지나고, 인공지능 연구가 진척됨에 따라, 유전 알고리즘의 효용성이 검증되었고 현대에 이르러서는 강화학습의 한 분야로 굳건하게 자리 잡고 있으며 수식으로 탐색하기 어려운 문제, 꼭 최고의 해가 아닌 그 근접한 해를 찾아도 되는 문제 등에 주로 쓰인다.

본 탐구에서는 유전 알고리즘의 효용성을 증명하기 위해 Python을 사용하여 루빅스 큐브와 삼목에 각각 유전 알고리즘을 사용하여 해결하는 방법을 제시하고, 그 원리에 대해서 탐구한다.

ⅱ. 탐구의 목적

본 탐구의 목적은 **존 홀랜드의 유전 알고리즘을 적용**하여 **루빅스 큐브의 해법을 제시**하고, 그 원리를 응용하여 삼**목 인공지능을 개발하는 방법을 제시**하여 **유전 알고리즘의 가용성을 증명**하는 데 있다.

탐구 과제 1) 유전 알고리즘을 이용한 루빅스 2×2×2 큐브의 해결

Python을 사용해 유전 알고리즘의 연산자들을 라이브러리화 시킨 모듈을 프로그래밍하고, 모듈을 이용하여 루빅스 큐브의 스크램블을 입력받아 최소 회전의 해결 방법을 출력하는 프로그램을 제작하고, 실제로 테스트한다.

탐구 과제 2) 유전 알고리즘과 최대-최소 트리를 이용한 삼목 AI 설계

유전 알고리즘, 최대-최소 트리, Pygame 모듈을 이용하여 삼목 인공지능을 프로그래밍하고, 삼목 게임 프로그램을 제작하여 유전 알고리즘을 활용할 수 있는 방법을 제시한다.

탐구 과제 3) 유전 알고리즘의 효용성 확인

3. 탐구 준비

1) 탐구 재료

- 사용 컴파일러 : Pycharm JetBrains
- 사용 언어 : Python, G.M.L
- 사용된 큐브 : 란란 2x2x2 큐브, 베리퍼즐 터트밍크스, 용쥔 뭐위 웨이룽GTS, 다얀 2x2x2 큐브

2) 탐구 기간 : 2016년 7월~2017년 3월

3) 탐구 수행 과정

1. 유전알고리즘을 이용하지 않는 2x2x2 큐브의 해결
2. 유전알고리즘을 이용한 2x2x2 큐브의 해결
3. 유전알고리즘과 최대-최소 트리를 이용한 오목 AI 설계

Ⅱ. 선행 연구

ⅰ. 유전 알고리즘 고찰

0) 개요

유전 알고리즘은 1975년 존 홀랜드가 개발한 전역 최적화 기법이다. 다윈의 적자생존 이론을 베이스로 자연세계의 진화의 양상을 본 따 해 집단들의 적합도를 점차 세대를 반복하며 올려나가 특정 해 집단을 찾는 알고리즘으로, 변이나 교배 등 생물학 용어들을 그대로 차용한 부분이 많다.

알고리즘의 처음에는 최초의 생명체처럼, 난수에 의해 해 집단들(세대, 해 집단 하나를 염색체, 해 집단의 원소 하나를 유전자라고 생각한다)을 설정한다. 물론 특정 해 집단을 프로그래머가 직접 설정할 수 도 있다. 이제 다음 차례로 적합도 함수를 짜야 한다. 프로그래머가 해 집단을 평가하는 적합도 함수를 작성하면, 세대의 염색체들이 적합도 함수에 의해 적합도를 평가받고, 선택 연산을 통해 우수한 염색체들을 골라내고, 그 염색체들을 토대로 교차하여 다음 세대의 염색체들을 만들어낸다. 염색체들이 만들어지면 그 세대에 대한 적합도를 평가하고, 완전히 만족하는 염색체를 찾아냈을 경우, 역시 프로그래머가 지정한 특정 행동(대부분 출력 후 종료)을 실시하고 루프를 빠져나온다. 하지만 찾아내지 못했을 경우, 다시 선택 연산을 실시하여 우수한 염색체들을 교차하여 다음 세대의 염색체들을 만들어내고, 변이 연산에 의해 돌연변이를 생산한다. 또 만족하지 못했을 경우 반복하고 반복하여 점진적으로 적합도가 0%에서 2%, 2%에서 8%, 8%에서 14%……와 같이 증가하게 한다.

1) 선택 연산

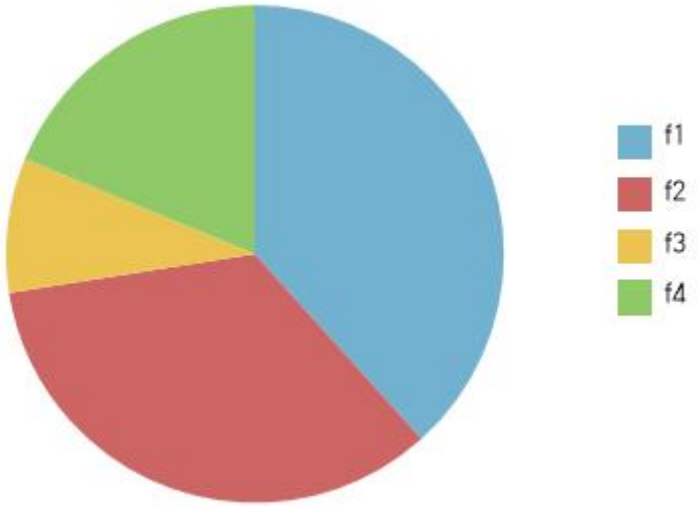
우수한 해 집단을 고르기 위해 시행하는 연산이다. 다양한 연산이 존재하지만, 우수한 해를 가려내야 하는 점은 공통적이다. 프로그래머는 선택압을 조절하여 선택되는 확률을 조정할 수 있다. 선택압은 가장 우수한 해와 가장 열등한 해의 차이를 말한다.

- 품질 비례 룰렛휠 연산

초등학교 저학년 시절 EBS에서 6시경에 방송하던 프로그램 '생방송 톡톡 보니하니'는 우리들에게 무한도전 급의 인기를 자랑했었다. 그 중 가장 인상 깊었던 코너는 뭐라 해도 〈돌려돌려 돌림판〉이었을 것이다. 직접 전화할 용기는 없었지만, 남의 사연을 듣고 그 사람이 원하는 경품을 탈 수 있을지 못 탈지 두근두근해 하던 그 기억이 난다. 품질 비례 룰렛휠 연산은 이 돌려돌려 돌림판과 비슷한 방식을 사용한다. 먼저 세대에서 가장 우수한 적합도를 가진 유전체의 적합도를 C_w라 하고, 가장 나쁜 적합도를 C_b라 한 뒤, 구하고자 하는 유전체의 적합도를 C_i라 한다. 그리고 아래의 식을 통해 품질을 구해 준다.

$f_i = (C_w - C_b) + (C_w - C_b)/(k-1)$

k가 바로 위에서 말했던 선택압이다. 프로그래머는 k값을 조정하여 부모가 되는 유전체의 폭을 조정할 수 있다. k 값으로는 주로 3을 쓴다 . 그리고 계산했던 f_i값을 토대로 룰렛휠에 공간을 배정한다.



그리고 이제 돌림판을 돌릴 시간이다.
적합도를 모두 더한 값 〉= x 〉= 0 인 실수 난수 x를 생성하고, 해당되는 범위를 부모로 지정한다.
이 방법은 유전적 다양성을 유지할 수 있고, 무난한 연산이어서 가장 많이 쓰이는 선택 연산법 중 하나이다.

- 토너먼트 선택 연산

토너먼트 선택 연산은 교차 연산의 방식인 균등 교차 연산과 상당히 유사한 연산이다. 세대에서 두 개의 유전체를 임의로 선택하고, 임의의 난수 p를 설정한다. 그 후 프로그래머가 조정할 수 있는 선택압 값 k와 난수 p의 대소를 따져 p가 작을 경우 적합도가 낮은 유전체를, p가 클 경우 적합도가 높은 유전자를 부모로 삼는다. 유전 알고리즘의 목표는 적합도를 높여가는 것이기에, 선택압 k를 낮게 하면 좋지 않은 결과를 보여줄 수 있으며, 0.6~0.8이 적절하다. 토너먼트 연산은 그 과정이 단순하고 간단한 만큼 수행에 필요한 시간을 획기적으로 줄일 수 있다는 점이 장점이다, 유전 알고리즘은 여러 유전체들을 동시에 병렬적으로 선택, 교차, 변이하기 때문에 규모가 많이 커진다면 일반 가정용 컴퓨터로는 많이 오래 걸리기 때문에 토너먼트 연산의 효율성은 절대 무시할 수 없다.

2) 교차 연산

위의 연산에서 두 부모 염색체 s_0, s_1를 골랐으면 그들에서 새로운 유전체 s_2를 만들어내야 한다. 이때, 교차 연산을 실시하여 두 부모 염색체 s_0, s_1를 토대로 자식 염색체 s_2를 만들어낸다.

- 균등 교차 연산

균등 교차 연산은 먼저 선택압 k를 0~1의 실수로 프로그래머가 선택하게 한다. 그 후 해 집단의 원소 f_i에 대한 임계 확률 p를 0~1의 실수 난수로 생성한다. 그리고 p와 k의 대소를 따져서 p〉k일 경우 s_0의 원소 f_i를 자손에 유전시키고, p≤k일 경우 s_1의 원소 f_i를 자손에 유전시킨다.

(k = 0.5)

S_0	0	1	1	1
S_1	1	0	1	0
임계 확률 p	0.458	0.98	0.14	0.256
s_2	1	1	1	0

- ~점 교차 연산

대표적인 교차 연산 방식으로, 점 p를 정하고, p보다 앞에 위치한 유전자는 모계를, 뒤에 위치한 유전자는 부계를 따르는 방식이다. 이 p의 개수에 따라 일점 교차와 다점 교차로 나뉜다.

S_0	0	1	1	1
S_1	1	0	1	0
p			p_0	
s_2	0	1	1	0

S_0	0	1	1	0	1	0	0	1	1
S_1	1	1	0	1	1	1	1	0	1
p			p_0			p_1			
s_2	0	1	0	1	1	0	0	1	1

3) 변이 연산

변이 연산은 세대를 거듭해나감에 따라 하나의 양상만 반복해서 나오는 문제를 극복하기 위해 도입된 방법이다. 알고리즘은 변이 연산을 통해 나쁘거나 그저 그런 값을 얻는 것이 보통이지만, 좋은 변이 하나가 문제 해결의 키가 되는 경우가 많다. 수많은 세대를 반복해 나갈 시에 두드러지게 나타나는데, 이는 상술했듯이 세대를 거듭해 나가면서 비슷한 유전자들로만 이루어진 유전체들로 세대가 구성되어져 가 문제 해결에 오랜 시간을 소요하거나 풀지 못하게 되기 때문이다.

- 단순 변이

이진 난수 r을 생성하고, 변이 확률 k와의 대소를 따져 변이 여부를 결정한다, 변이는 프로그래머가 설정한 범위 내의 난수로 지정하는 등 프로그래머의 재량에 따른다. 품질이 좋은 변이는 후대에 자신을 복제하지만, 그렇지 않은 변이는 자연스레 도태된다.

- 비균등 변이

당연하게도, 프로그램의 초기에는 유전체들의 품질이[1] 그렇게 좋지 못하다. 난수로 생성된 유전체들에게 무엇을 바라겠는가, 그러나 유전 알고리즘 특성상 프로그램이 실행될수록 유전체들의 비용이 좋아지고, 따라서 변이가 품질에 좋지 않은 영향을 줄 확률이 늘어난다. 반대로 초기에는 변이를 주는 확률을 크게 만들어 많은 경우의 수들 중 좋은 유전자들을 수렴해야 한다. 따라서 세대가 진척해 나감에 따라 변이의 정도를 줄이는 비균등 변이 연산을 사용하면 역시 필요 세대 수를 줄이면서 효율적으로 연산을 수행할 수 있다. (r은 이진 난수, 0≤r≤1 범위의 난수)

$$\triangle(t, y) = y.(1-r_2(1-t/T)^b)v = \begin{pmatrix} v+\triangle(t, UB-v) \text{ if } r=0 \\ v+\triangle(t, v-LB) \text{ if } r=1 \end{pmatrix}$$

위의 식에서, 임의의 유전자 v에 대해 이진 난수 r의 값에 따라 $\triangle(t,y)$값을 설정한다. t는 현재의 세대이고, UB와 LB는 각각 변이의 최대치, 최소치를 나타낸다. 그리고 $\triangle(t,y)$는 전체 세대인 T와 현재 세대 t에 따라 변이의 강도를 조절하는 함수이며, t가 늘어나면 함숫값이 0으로 근접하는 특성을 갖는다. ($0 \le \triangle(t,y) \le y$)[2]

4)스키마

계획, 도식을 뜻하는 단어 Schema에서 유래한 단어로 존 홀랜드가 도입하였다. 스키마는 0, 1, *로 이루어진 문자열 집합으로, *는 와일드카드로 아무 값이나 들어가도 상관 없는 위치이다.

0	1	*	*	1

- 스키마와 염색체의 관계

부합 : *를 제외한 스키마와 염색체의 모든 유전자가 일치할 때

0	1	*	*	1

- 스키마(H)

0	1	0	0	1

- 유전체1(s_0) : 부합

0	1	1	1	1

- 유전체2(s_1) : 부합

0	1	1	0	0

- 유전체3(s_2) : 스키마가 부합하지 않음

위의 스키마 H에서는 와일드카드(*)가 아닌 비트가 총 3개이므로 스키마 H의 차수는 3이다. 또한 스키마 H는 s_0과 s_1에 부합하는데, 이때 s_0과 s_1을 스키마 H의 인스턴스라고[3] 한다.

- 스키마타(Schemata), 유전 알고리즘

유전 알고리즘은 스키마타를 조정한다. 스키마타는 스키마의 복수형으로 스키마 정리를 통해 다음 세대에 해당 스키마가 존재할 확률을 예측할 수 있다. 유전 알고리즘에서 선택과 교차 연산의 성질에 의거, 적합도가 높은 스키마는 다음 세대에도 남을 확률이 크고, 적합도가 낮은 스키마는 도태된다. 즉, 선택과 교차 연산은 스키마의 인스턴스를 파괴하거나 생성한다. 이것이 유전 알고리즘의 원리이다.

ii. 게임 트리, 최소최대 알고리즘 고찰

0) 개요

미니맥스(MiniMax, 최소-최대) 알고리즘과 게임 트리는 알파고와 같은 게임 인공지능 개발에 매우 중요하게 다루어지는 이론이다. 본 탐구에서 진행할 오목 인공지능 (이름)의 개발에 앞서 탐구 전반적으로 다루어지는 미니맥스 알고리즘, 게임 트리 이론에 대해 고찰을 한다.

1) 게임 트리의 정의

게임을 하다 보면, 여러 가지 상황에 마주하게 되는데, 이런 상황들을 노드로 하여 연결한 유향 그래프가 게임 트리이다. 각각의 노드의 자식 노드들은 부모 노드의 한 수 이후에 도달할 수 있는 다음 상태를 의미한다.

게임 트리는 인공지능 관련 연구에서 중요하게 다뤄지는데, 체스, 장기, 오목 등의 경우 그 상황에서 게임 트리를 이용해 몇 수 앞의 경우를 예상한 후, 최적의 수를 찾는다.

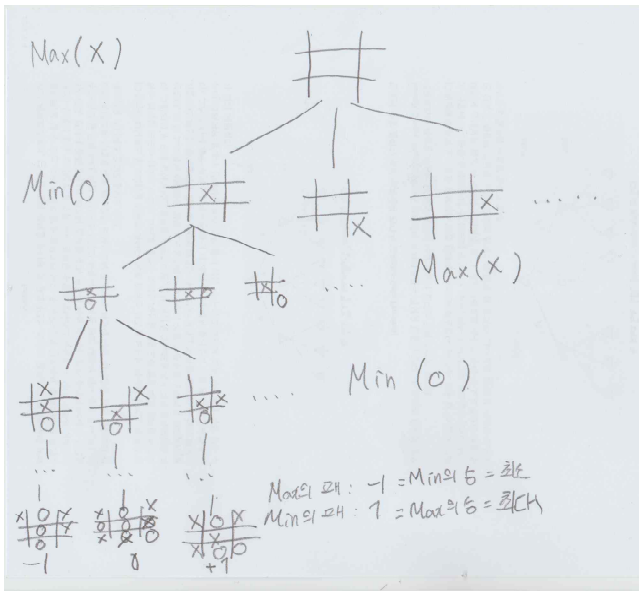
2) 제로섬 게임(Zero-Sum Game)

제로섬 게임은, A와 B가 게임을 할 때, A가 이득을 취하면 B는 손해를 입어 결국 A와 B의 합이(Sum) 0이(Zero) 되는 게임을 뜻한다. 즉, 0점에서 시작하는데, A가 5점을 얻을 경우 B는 5점을 잃어 A(5) + B(-5) = 0이 되는 게임이다. 제로섬 게임의 대표적인 예시로는 선거, 포커, 오목이나 Tic-Tac-To[4]가 있다.

3) 제로섬 게임에서의 게임 트리

제로섬 게임에서, 'A에게 유리한 수 == B에게 유리한 수' 라는 가정은 유효하다. 즉, 게임 트리에서 내 이익을 최대화하고 상대방의 이익을 최소화하는 단말 노드를 선택하는 것이 제로섬 게임에서 게임 트리를 활용하여 최선의 수를 찾는 방법이다.

아래의 게임은 대표적인 제로섬 게임인 Tic-Tac-To의 미니맥스 알고리즘을 이용한 게임 트리의 일부분이다.

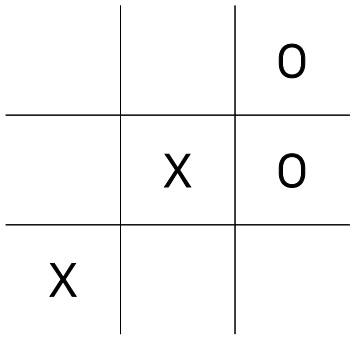


4) 최대 최소 알고리즘 : Mini-Max Algorithm

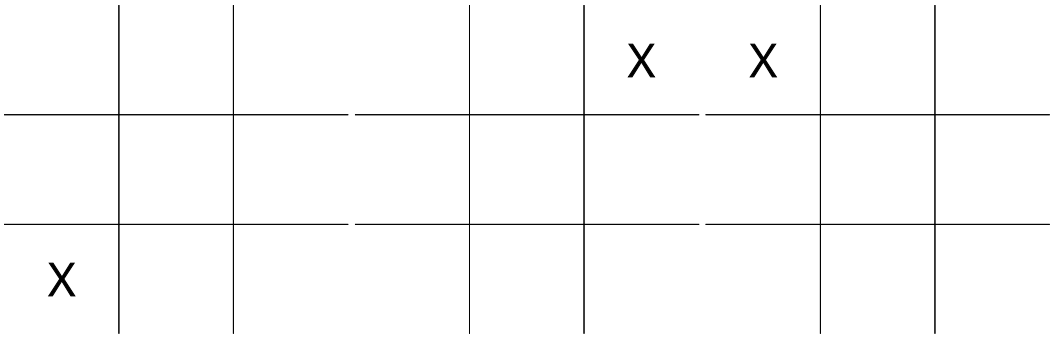
게임 트리에서, A의 시점에서 봤을 때 A의 차례에는 A의 평가치가 최대가 되는 선택을 할 것이고, B의 차례에는 A의 평가치가 최소가 되는 선택을 할 것이다. A를 MAX라고 하고 B를 MIN이라고 하면, MAX와 MIN의 행동의 쌍인 겹(depth)가 게임 트리에서의 탐색의 단위가 된다.

Tic-Tac-Toe를 예시로 들면, 평가함수 f(x)는 아래와 같다(MAX는 X로 표기, MIN은 O)

1. MAX가 이기는 경우	f(x) = 1 (가장 큰 값)
2. MIN이 이기는 경우	f(x) = -1 (가장 작은 값)
3. 승패가 갈리지 않는경우	f(x) = (MAX가 가능한 행, 열, 대각선의 수) -(MIN이 가능한 행, 열, 대각선의 수) * 0.1



하여 이 때 이 노드에 대한 f(x)의 값은 5-2=3이 된다. 또한 자식 노드는 대칭성을 고려하므로



위의 세 노드는 전부 같은 것으로 처리한다. 이 때 발견할 수 있는 특성이 게임 초기에도 게임 후반에도 분기계수가 크게 늘거나 줄지 않는다는 것이다. 초반에는 대칭성으로 인하여 분기 계수가 적고, 후반에는 선택할 수 있는 선택지가 줄어 분기 계수가 많지 않다.

그러면 틱-택-토에 대해 미니맥스 알고리즘을 적용하여 게임 트리를 그리면 아래와 같다

Ⅲ. 유전 알고리즘을 이용한 2*2*2 큐브의 해결

ⅰ. 탐구 계획 수립

우선 간편한 탐구 수행을 위해 유전 알고리즘에 사용되는 메서드들을 라이브러리화 시킨 모듈을 제작한다. 그 후

ⅱ. 탐구 수행 : 유전 알고리즘 모듈 Pygene 프로그래밍

1) 프로그램의 구조

Pygene 모듈은 Generation 클래스와 연산에 이용되는 두 함수로 구성되어 있다. 먼저 Generation 클래스는 생성시에 염색체들의 집합과(이하 '세대') 연산 타입을 넘겨받는다. 그리고 사용자가 객체의 evol 메서드를 호출하면 넘겨받은 연산 타입에 따라 내부적인 진화 프로세스를 진행한다.

```
    def evol(self, k, t, g):
        self.Generation = self.offspring(k, t)
        self.Generation = self.mutant(self.mutchance, g)
        mxa = self.func(self.Generation[0])
        for i in range(self.len) :
            if mxa < self.func(self.Generation[i]) :
                self.Generation.insert(0, self.Generation.pop(i))
        return self.Generation
    def offspring(self, k, t):
        result = []

        for _ in range(self.len):
            if self.choice == 1 : parents = self.choice_R(k)
            elif self.choice == 2 : parents = self.choice_T(k)
            dad, mom = parents

            if self.cross == 1 : son = self.cross_U(t, dad, mom)
            elif self.cross == 2 :  son = self.cross_P(t, dad, mom)
            result.append(son)
        return result
```

evol, offspring 메서드 코드 전문

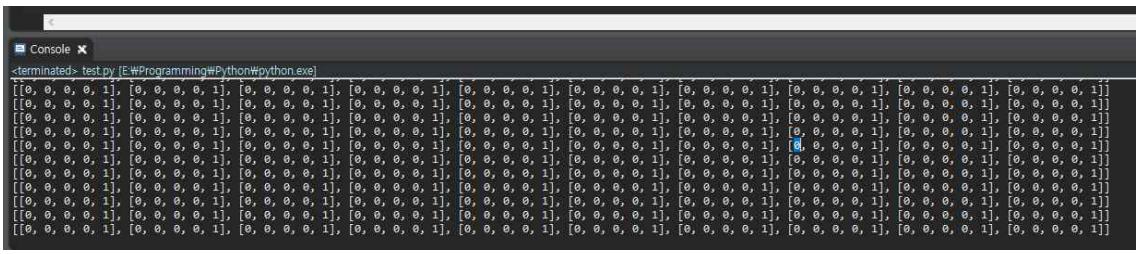
내부적인 진화 프로세스를 거치는 작업을 '한 세대'라고 표현한다. 한 세대가 끝나면 offspring 메서드에서 생성된 result-진화를 거친 세대를 반환한다.

2) 모듈 테스트

제작한 Pygene 모듈의 테스트를 위해 Pygene을 활용하는 간단한 문제를 해결해 본다

```
import pyGene
import random

olist = [0, 0, 0, 0, 0]
gen  =  [[random.randint(0,  1),  random.randint(0,  1),  random.randint(0,  1),
random.randint(0, 1), random.randint(0, 1)] for i in range(10)]

def func(l):
    f = 0
    for i in range(5):
        if olist[i] == l[i]:
            f+=1
    return f

Gen = pyGene.Generation(gen, 2, 1, 0, 1, 0, func, 1000)

for i in range(100):
    print(Gen.evol(50, 50, i)[0])
```

먼저, 목표 리스트와 초기 세대를 초기화하고 적합도 함수 func를 작성한다.
그 후 Generation 클래스의 Gen 객체를 토너먼트 선택, 균등교차 연산, 변이확률을 0, 선택압을 각각 0.5로 설정하여 100회 세대를 반복시킨다.



세대를 반복시키면 위와 같은 결과값을 출력하게 된다. 진화의 방향이 잘못 잡혀 더 나아가지 못하는 양상을 보이고 있다. 이 문제는 세대의 유전적 다양성이 극도로 적어졌기 때문이다.
이 문제를 해결하기 위해 pyGene에 아래의 식을 사용하는 비균등 변이 메서드를 추가한다.

$$v = \begin{pmatrix} v + \triangle(t, UB - v) \text{ if } r = 0 \\ v + \triangle(t, v - LB) \text{ if } r = 1 \end{pmatrix} \quad \triangle(t, y) = y.(1 - r_2(1 - t/T)^b)$$

위의 식에서 v는 변이를 적용할 유전자, r과 r2는 (0, 1)의 이진난수, T는 반복할 세대의 수, t는 반복한 세대의 수(t≠0) UB는 생성될 수 있는 난수의 최댓값, LB는 생성될 수 있는 난수의 최솟값을 뜻한다. 위 식에서 △함수는 0~y 사이의 난수를 생성하되, 세대가 진행될수록 그 값이 0에 수렴하게 된다. 즉, 위의 식은 v가 변이하는 정도가 t의 크기, 즉 세대의 진행 정도에 반비례한다.

```
def mutant(self,k, g): #g는 세대의 진행 정도
        result = [ [0]*5 for _ in range(10) ]
        d = lambda t, y: y*(1-r*((t+1)/self.Mg))
        for h in range(self.hei):
            for w in range(self.wid):
                r = pick(0,1, self.mutchance) #변이 여부를 결정하는 난수
                s = random.randint(0,1) #변이 시 유전자의 크기가 증가/감소
                if s==0 : result[h][w] = self.Generation[h][w]+math.floor(d(g,
self.mutMax-self.Generation[h][w]))
                elif s==1 : result[h][w] = self.Generation[h][w]-math.floor(d(g,
self.Generation[h][w]-self.mutMin))
        return result
```

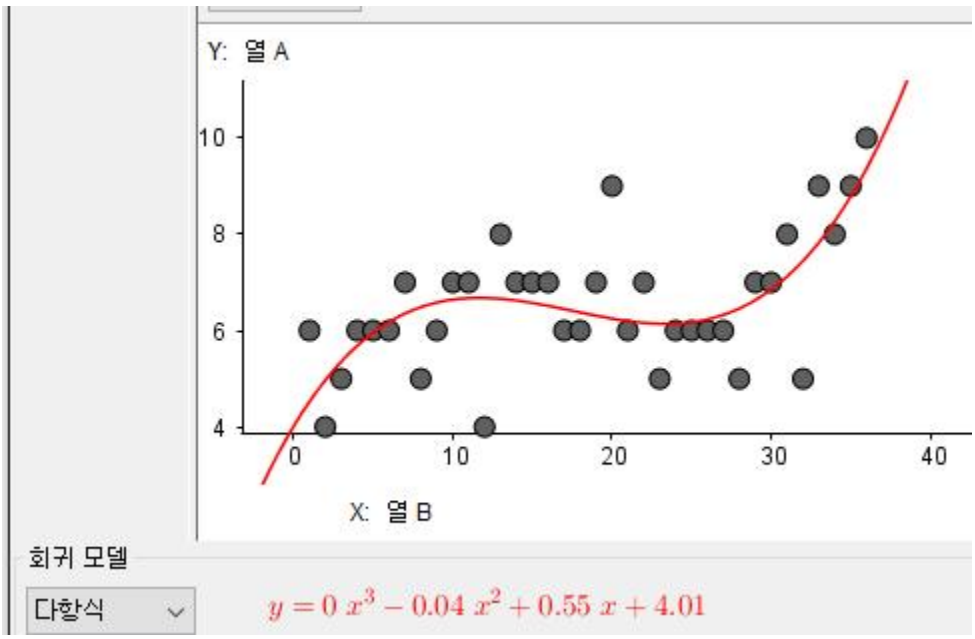
위의 변이 메서드를 적용시킨 뒤 세대를 다시 진화시키면

```
[0, 0, 0, 1, 1]
[0, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
[0, 1, 0, 0, 1]
[0, 0, 0, 1, 0]
[0, 0, 0, 1, 0]
[0, 0, 1, 0, 0]
[0, 1, 1, 0, 0]
[0, 0, 0, 0, 0]
```

위와 같이 정상적으로 진화하여 목표 리스트인 [0, 0, 0, 0, 0]에 도달하는 것을 확인할 수 있다.

3) 시각화

먼저 유전자가 5개뿐인 초기의 테스트 파일은 목표 리스트에 도달하는데 걸리는 세대가 그래프로 나타내기에는 매우 짧기 때문에, 목표 리스트의 유전자를 10개로 바꾼다.
그 후 test.py의 출력 부분에 수정을 가해 최상위 유전자 대신 func(g)값을 출력하게 변경한다.
GA를 통한 진화 과정을 x축에 세대 수, y축을 해당 세대의 적합도로 두고 산점도를 그린 후



회귀식을 구하면 이와 같이 상승세를 띄는 삼차함수의 그래프가 된다. 이를 통해 GA가 해의 품질을 상승시키는 모습을 시각적으로 나타내어 보았다.

4) 코드 전문 : Github 저장소

유전알고리즘 모듈

iii. 탐구 수행 : Pygene을 이용한 루빅수 2×2×2 큐브 해결 프로그래밍

1) 프로그램의 구조

프로그램은 크게 '유전알고리즘' 파트와 '큐브 회전' 파트로 나누어져 있다.

1) 유전알고리즘

```
def getGeneByRandom(self):
    return
```

유전자를 무작위로 생성한다.

```
def isTargetScore(self,score):
    return True
```

목표 점수를 리턴한다. 이것은 큐브 객체에서 추가로 정의될 것이다.

```
def byProb(sef,iProbability):
    return random.randint(1, 10000) < iProbability * 100
```

입력된 확률에 따라 참 또는 거짓을 리턴한다. 나중에 돌연변이를 생성하는데 쓰인다.

```
def crossover(self, iNewChromoNo, iChromoNo1, iChromoNo2):
    if self.byProb(50):
        for i in range(0, int(self.geneCnt / 2)):
            self.newChromo[iNewChromoNo][i] = self.chromo[iChromoNo1][i]
            self.newChromo[iNewChromoNo][int(self.geneCnt / 2) + i] = self.chromo[iChromoNo2][
                int(self.geneCnt / 2) + i]
    else:
        for i in range(0, int(self.geneCnt / 2)):
            self.newChromo[iNewChromoNo][i * 2] = self.chromo[iChromoNo1][i * 2]
            self.newChromo[iNewChromoNo][i * 2 + 1] = self.chromo[iChromoNo2][i * 2 + 1]
```

유전자를 섞는 행위로, 위에서 설명된 byProb 함수를 이용해 50% 확률로 True이면 앞뒤로 크로스오버를, False면 홀짝의 위치를 교차하여 크로스오버를 한다.

```
def mutate(self, iNewChromoNo):
    if self.byProb(self.MUTANT_PROBABILITY):
        for i in range(0, self.geneCnt):
            if self.byProb(50):
                self.newChromo[iNewChromoNo][i] = self.getGeneByRandom()
```

돌연변이를 생성하는 함수로, 먼저 유전자 변이 확률로 바꿀지 말지를 결정하고, 바꾸게 될 경우에는 각각의 유전자를 하나씩 50%확률로 바꿔준다.

```
def calFitness(self, iChromoNo):
    return
```

각각의 개체의 적합도를 계산하는 함수이다. 이것은 어떨 때 좋은 것인지가 각각의 프로그램에 따라 다르기 때문에 큐브 쪽에서 구현한다.

```
def selectChromoByRoulette(self,exceptChromoNo=-1):

    vRoulette       = []
    vSelectedChromoNo = 0
    vFitnessSum     = 0


    for i in range(0, self.chromoCnt):
        vFitnessSum+=self.fitness[i]
        vRoulette.append(vFitnessSum)


    while True:
        r = random.randint(1, vFitnessSum)
        for i in range(0, self.chromoCnt):
            if vRoulette[i] >= r:
                vSelectedChromoNo = i
                break
        if vSelectedChromoNo != exceptChromoNo:
            break

    return (vSelectedChromoNo)
```

룰렛휠 방식으로 크로스오버할 염색체를 고른다. 적합도가 높은 염색체일수록 채택 확률이 높아진다.

```
def isMaxGeneration(self):
    return self.MAX_GENERATION <= self.curGeneration
```

이 함수는 각각의 염색체의 적합도가 종료 조건에 맞는지를 판단하는 함수이다.

```
def printInit(self):
    print("=============================================================================================")
    print("시간:{}".format(time.asctime()))


def printProcessing(self,topScore,chromoNo):
    print("시간:{},세대:{},염색체:{},적합도:{}".format(time.asctime(),self.curGeneration,self.chromo[chromoNo],topScore))

def printSuccess(self,topScore,chromoNo):
    print("##성공## 시간:{},세대:{},염색체:{},적합도:{}".format(time.asctime(),self.curGeneration,self.chromo[chromoNo],topScore))

def printFail(self):
    print("##실패## 시간:{},세대:{}".format(time.asctime(),self.curGeneration))
```

시작, 진행 과정, 결과를 표시하는 함수이다. 여기서 실패란 2,000,000세대를 넘어간 경우를 뜻

한다.

```
def makeSon(self):
    for i in range(0,self.chromoCnt):
        vSelectedChromo1 = self.selectChromoByRoulette()
        vSelectedChromo2 = self.selectChromoByRoulette(vSelectedChromo1)
        self.crossover(i, vSelectedChromo1, vSelectedChromo2)
        self.mutate(i)
        self.chromo[i]=self.newChromo[i]
```

위에서 만든 함수인 crossover함수와 mutate함수를 사용해 다음 세대를 만든다.

```
def solve(self):

    vTopScore = 0
    vStartTime = time.time()

    self.printInit()


    self.createFirstChromo()

    while not self.isMaxGeneration():
        self.curGeneration += 1

        vTopChromoNo=0
        for i in range(0,self.chromoCnt):
            vScore=self.calFitness(i)
            if self.isTargetScore(vScore):
                vTopScore=vScore
                self.printSuccess(vTopScore,i)
                return ((time.time() - vStartTime),True,self.curGeneration)
            if vScore > vTopScore:
                vTopScore =vScore
                vTopChromoNo=i
                self.printProcessing(vTopScore, vTopChromoNo)


        self.makeSon()

    self.printFail()

    return ((time.time() - vStartTime), True, self.curGeneration)
```

이 함수는 유전알고리즘에 따라 문제를 해결하는 함수이다. 걸리는 시간을 리턴하고, 최고 점수를 기록한다. 또한, 2,000,000세대 안에 끝내지 못하면 실패로 처리한다.

2)큐브

```
CUBE_SIZE = 24
# 큐브의 정답상태.
solvedCube = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24)

# 큐브의 회전 기호
DIR = ('U', 'U`', 'D', 'D`', 'F', 'F`', 'B', 'B`', 'R', 'R`', 'L', 'L`')
```

이것은 초기조건을 입력한 것으로, 맞춰진 큐브의 상태와 각각의 회전 기호를 컴퓨터는 0,···,11로 받아들이기 때문에 인간이 알아볼 수 있도록 회전 기호로 변환하는 용도이다.

```
def __init__(self,chromocnt,geneCnt,mutantProb,maxGeneration,targetScore):
    CGeneticAlg.__init__(self,chromocnt,geneCnt,mutantProb,maxGeneration,targetScore)



    self.transform = [[4, 1, 2, 3, 5, 6, 7, 8, 17, 18, 11, 12, 13, 14, 23, 24, 15, 16, 19, 20, 21, 22, 9, 10],  # U  0
                      [2, 3, 4, 1, 5, 6, 7, 8, 23, 24, 11, 12, 13, 14, 17, 18, 9, 10, 19, 20, 21, 22, 15, 16],  # U' 1
                      [1, 2, 3, 4, 8, 5, 6, 7, 9, 10, 21, 22, 19, 20, 15, 16, 17, 18, 11, 12, 13, 14, 23, 24],  # D  2
                      [1, 2, 3, 4, 6, 7, 8, 5, 9, 10, 19, 20, 21, 22, 15, 16, 17, 18, 13, 14, 11, 12, 23, 24],  # D' 3
                      [1, 2, 24, 21, 20, 17, 7, 8, 12, 9, 10, 11, 13, 14, 15, 16, 4, 18, 19, 3, 6, 22, 23, 5],  # F  4
                      [1, 2, 20, 17, 24, 21, 7, 8, 10, 11, 12, 9, 13, 14, 15, 16, 6, 18, 19, 5, 4, 22, 23, 3],  # F' 5
                      [18, 19, 3, 4, 5, 6, 22, 23, 9, 10, 11, 12, 16, 13, 14, 15, 17, 7, 8, 20, 21, 1, 2, 24],  # B  6
                      [22, 23, 3, 4, 5, 6, 18, 19, 9, 10, 11, 12, 14, 15, 16, 13, 17, 1, 2, 20, 21, 7, 8, 24],  # B' 7
                      [1, 10, 11, 4, 5, 14, 15, 8, 9, 6, 7, 12, 13, 2, 3, 16, 20, 17, 18, 19, 21, 22, 23, 24],  # R  8
                      [1, 14, 15, 4, 5, 10, 11, 8, 9, 2, 3, 12, 13, 6, 7, 16, 18, 19, 20, 17, 21, 22, 23, 24],  # R' 9
                      [13, 2, 3, 16, 9, 6, 7, 12, 1, 10, 11, 4, 5, 14, 15, 8, 17, 18, 19, 20, 24, 21, 22, 23],  # L  10
                      [9, 2, 3, 12, 13, 6, 7, 16, 5, 10, 11, 8, 1, 14, 15, 4, 17, 18, 19, 20, 22, 23, 24, 21]]  # L' 11


    self.mixedCube = self.solvedCube[:]

    self.mixedChromo = []
    self.cubeRotatedByGene=[[]]

    self.cubeRotatedByGene.clear()

    self.cubeRotatedByGene.clear()
```

이것은 큐브 class에서 기본적으로 실행하는 것으로, 큐브를 회전했을 때 조각의 이동 위치를 표현하고, 큐브를 섞고, 각각의 염색체와 큐브를 초기화한다.

```
    for i in range(0,self.chromoCnt):
        self.cubeRotatedByGene.append(list(self.solvedCube[:]))

    self.mixedChromo.clear()
    for i in range(0,self.geneCnt):
        self.mixedChromo.append(0)


    self.chromo.clear()
    self.newChromo.clear()
    for i in range(0,self.chromoCnt):
        self.chromo.append(self.mixedChromo[:])
        self.newChromo.append(self.mixedChromo[:])
```

각각의 염색체의 큐브와 염색체를 초기화 시킨다.

```
def createFirstChromo(self):
    for i in range(0,self.chromoCnt):
        for j in range(0, self.geneCnt):
            self.chromo[i][j]=self.getGeneByRandom()

    self.rotateCubeByChromo()
```

첫 염색체를 생성하고 그 염색체에 따라 각각의 큐브를 회전시킨다.

```
def scrambleCube(self,inputChromo=[]):

    if len(inputChromo) == 0:
        for i in range(0,self.geneCnt):
            self.mixedChromo[i]=self.getGeneByRandom()
    else:
        if len(inputChromo) != self.geneCnt:
            print("입력 염색체의 유전자 갯수를 확인하세요.")
            raise NotImplementedError

        self.mixedChromo = inputChromo[:]


    for i in range(0, self.geneCnt):
        self.rotate(self.mixedChromo[i],-1)
```

섞는 유전자에 따라 큐브를 섞는다.

```
def getGeneByRandom(self):
    return random.randint(0,self.geneCnt-1)
```

큐브를 섞을 유전자를 만든다.

```
def calFitness(self, iChromoNo):

    vFitness=0
    for i in range(0, self.CUBE_SIZE):
        if self.cubeRotatedByGene[iChromoNo][i] == self.solvedCube[i]:
            vFitness += 2
        else:
            vFitness += 1
    self.fitness[iChromoNo] = vFitness
    self.fitnessSum         += vFitness

    return (vFitness)
```

각각의 염색체의 적합도를 계산한다. 맞을 경우 2점, 틀릴 경우 1점. 그 결과 값을 리턴해 룰렛휠 연산에 사용한다.

```
def isTargetScore(self,score):
    return (score >= self.TARGET_SCORE )
```

위에 나왔던 함수를 다시 정의한다. 현재 점수가 목표 점수 이상이면 TRUE. 이 경우에 프로그램이 종료된다.

```
def rotate(self,direction,cubeNo):
    tmpCube = self.cubeRotatedByGene[cubeNo][:]

    for i in range(0, self.CUBE_SIZE):
        self.cubeRotatedByGene[cubeNo][i]= tmpCube[(self.transform[direction][i] - 1)]


    if cubeNo ==-1:
        self.mixedCube = self.cubeRotatedByGene[cubeNo][:]
```

돌릴 큐브를 방향에 따라 돌린다. cubeNo가 -1일 경우에는 mixedCube를 돌린다.

```
def rotateCubeByChromo(self):
    for i in range(0,self.chromoCnt):
        self.cubeRotatedByGene[i] = self.mixedCube[:]
        for j in range(0, self.geneCnt):
            self.rotate(self.chromo[i][j],i)
```

각각의 큐브를 각각의 유전자에 맞춰 돌린다.

```
def makeSon(self):
    super().makeSon()
    self.rotateCubeByChromo()
```

현재의 유전자로 자식을 만든다.

```
def convertGeneCode(self,chromo):
    return list(self.DIR[chromo[i]] for i in range(0,self.geneCnt))


def convertCodeGene(self,chromo):
    return list( self.DIR.index(chromo[i]) for i in range(0,self.geneCnt) )
```

위에서 설명했던 회전 코드를 인간이 알아볼 수 있게 바꾸고, 인간이 알아볼 수 있는 문자를 회전 코드로 바꾼다.

```
def printInit(self):
    super().printInit()
    print("초기염색체:{}  ,초기큐브상태{}".format(self.convertGeneCode(self.mixedChromo),self.mixedCube))


def printProcessing(self,topScore,chromoNo):
    super().printProcessing(topScore,chromoNo)
    print("     큐브섞음공식:{},큐브상태:{}".format(self.convertGeneCode(self.chromo[chromoNo]),self.cubeRotatedByGene[chromoNo]))


def printSuccess(self,topScore,chromoNo):
    super().printSuccess(topScore,chromoNo)
    print("     초기염색체:{},초기큐브상태{}".format(self.convertGeneCode(self.mixedChromo),self.mixedCube))
    print("     해답염색체:{},   큐브상태:{}".format(self.convertGeneCode(self.chromo[chromoNo]),self.cubeRotatedByGene[chromoNo]))
```

처음, 진행과정, 성공 결과를 표시한다. 진행 과정은 적합도가 더 높을 때만 표시한다.

3)메인 프로그램

```
vElapsedTimeSum=0
vSuccessCnt    =0
vGenerationSum =0


for i in range(0,100):
    cube=C22CubeGene(10,8,5,2000000,48)
    cube.scrambleCube()
    vElapsedTime,vSuccess,vGeneration = cube.solve()
    vElapsedTimeSum=(vElapsedTimeSum+vElapsedTime)
    vGenerationSum =(vGenerationSum+vGeneration)
    if vSuccess:
        vSuccessCnt+=1
        vElapsedTimeSum=(vElapsedTimeSum+vElapsedTime)
        vGenerationSum =(vGenerationSum+vGeneration)
"""
cube=C22CubeGene(10,8,5,2000000,40)
cube.scrambleCube(cube.convertCodeGene(['B', 'F', 'U', 'B', 'U', 'B', 'D``, 'B`']))
vElapsedTime,vSuccess,vGeneration = cube.solve()
print("   경과시간 {},성공여부  {},세대 {} ".format(vElapsedTimeSum,vSuccess,vGeneration))
"""
print("   평균경과시간 {},성공갯수 {},평균세대 {} ".format(vElapsedTimeSum/vSuccessCnt,vSuccessCnt,vGenerationSum/vSuccessCnt))
```

위에서 만든 프로그램에 숫자를 넣어(10,3,5,2000000,48)큐브를 맞춘다. 맨 위에 있는 변수는 평균 시간, 성공 개수, 평균 세대를 구하기 위한 것이다. 중간에 막혀있는 함수는 사람이 직접 섞는 함수이다.

2) 프로그램 테스트 결과

테스트 부분이 길어져 외부 링크로 대신한다.

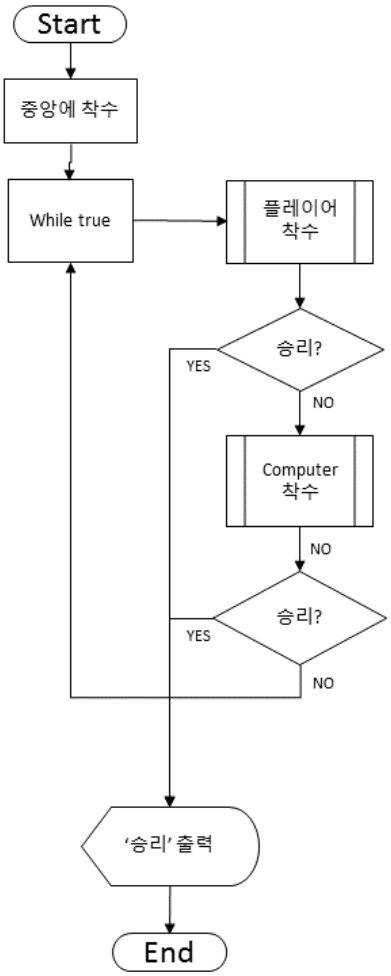
Ⅳ. 유전 알고리즘을 이용한 삼목 인공지능 프로그래밍

ⅰ. 탐구 계획 수립

1) 탐구의 목적

선행 탐구에서 제작한 유전 알고리즘 모듈(Pygene)을 활용하여 삼목(틱택토)게임을 플레이하는 인공지능을 제작하여 유전 알고리즘의 효용성을 증명하고, 유전 알고리즘을 오목과 같은 제로섬 게임의 인공지능 개발의 한 방법으로서 제시한다.

2) 프로그램의 흐름



프로그램의 시작에는 클래스와 상수들을 정의하고 옆의 순서도에 따라 프로그램을 진행한다.
먼저 중앙에 착수한다. 이는 통계적인 근거에 따른 것인데, 모든 경우의 수를 따졌을 때 중앙에 착수하여 승리할 확률이 50%, 변과 귀가 각각 30%와 20% 순으로 도출된다.
그 뒤 메인루프를 시작하고, 플레이어가 착수를 하면 승리 여부를 체크하고 아니라면 컴퓨터가 착수한다. 마찬가지로 승리 여부를 체크하고 루프를 반복한다. 두 경기자 중 하나라도 승리하면 루프를 빠져나오고 “Victory”문자열을 출력한 뒤 프로그램을 종료한다.
유전 알고리즘을 이용한 컴퓨터의 최적수 탐색 부분이 이 탐구주제의 핵심으로, 뒤에서 상세히 다루도록 한다.

ii. 탐구 수행

1) Import 파트

```
import copy
import pygene
```

파이썬은 그 독자적인 특징(명시성 등)을 살리기 위해 일종의 코딩 규칙을 제안하고 있다. 이것이 바로 PEP8인데, 이 탐구에 나오는 코드 또한 PEP 규칙을 최대한 준수하면서 작성되었다.
그런 이유로, 모듈 임포트는 언제나 코드의 맨 앞에 위치하게 된다. 이 프로그램에서는 두 가지 모듈을 필요로 하는데, 하나는 copy 모듈이고 나머지 하나는 선행 탐구에서 제작한 pygene이다.

2) Define 파트

모듈을 임포트한 후에는, Main 파트에서 사용할 여러 클래스, 상수와 함수들을 정의한다.
정의하는 클래스들은 다음과 같다.

상수	HEI: 게임판의 높이	WID: 게임판의 너비	DEP:탐색할 최대 깊이	
함수	convert_c(실수): 실수 인자를 받아 HEI, WID값에 따라 순서쌍 형식으로 변환		convert_r(순서쌍): 순서쌍을 HEI, WID값에 따라 실수 형식으로 변환	
Board	게임판 클래스, data로 게임판 리스트를 갖는다			
	Draw:출력	Func:평가	Move:착수	is_empty: 비었는지 체크
Node	노드 클래스, data로 Board 인스턴스를 갖는다			
	parent:부모 노드		children[]:자식 노드 리스트	
	depth:트리에서의 깊이		__add__[5]:자식 노드로 추가	
AI	AI 클래스, 게임의 메인 Board 인스턴스를 board로 갖는다			
	root:최상위 노드		board: 게임판 인스턴스	
	optimum: 최적수 탐색 총괄	search_possible: 보드에서 가능한 모든 경우의 수 탐색 후 반환	connect: 게임 트리 건설	search: 순회탐색
Player	ply_move : 플레이어 이동			

자홍색 : 메서드 | 파란색 : 상수 혹은 변수

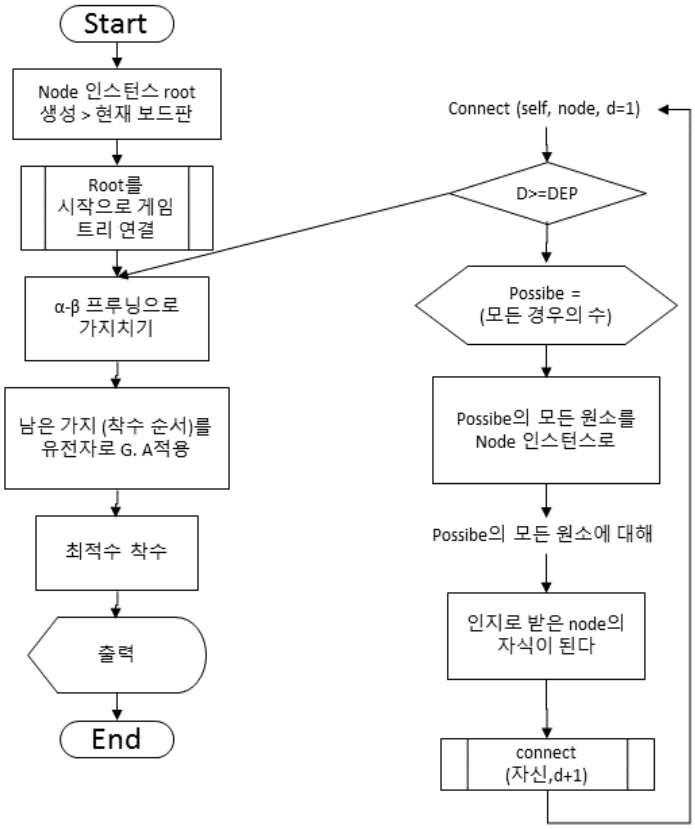
- Board 클래스

  게임판 데이터를 관리하는 클래스이고, 리스트 data를 갖고 있으며 게임판의 상황을 WID*HEI 크기의 2차원 리스트로 나타낸다. 내장 메서드 Draw는 data를 출력하며, Func는 data의 평가치를 구해 반환한다. Move는 순서쌍(이하 튜플)형식 인자를 받아 해당 위치에 착수하고 승리 여부를 반환한다.

- Node 클래스

  게임 트리를 구성하는 기본 성분이다. data에는 노드에 대응하는 Board 인스턴스를 받는다. parent는 해당 노드의 부모 노드를 저장하고, 마찬가지로 children은 해당 노드의 자식 노드들의 집합이다. depth는 트리에서의 깊이를 저장하며 디폴트값은 0이다. 또한 + 연산자를 이용하여 자식 노드로 추가할 수 있다.

- AI 클래스



이 프로그램의 꽃이자 주인공인 인공지능을 담당하는 클래스이다. root는 최상위 노드, 즉 현재 게임판의 상태를 데이터로 갖는 노드를 참조하고, board는 게임에 단 하나뿐인 메인 Board 객체를 참조한다.

AI는 optimum 메서드를 통해 그 상황에서 최적의 수를 결정한다. 나머지 메서드는 모두 optimum(이하 최적수) 메서드를 보조한다. 최적수 메서드는 호출될 때 현재 보드의 상태를 root에 지정한다. 그 후 root에서 시작해서 모든 경우의 수를 따지는 게임 트리를 세우는 connect 메서드를 호출한다.

connect 메서드는 재귀의 방식을 채용하여 진행된다. 인자로 노드를 받아 노드의 data에서 가능한 모든 경우의 수를 찾고, 노드 객체로 만든 후 인자로 받은 노드의 자식 노드로 설정한다. 그리고, 찾은 각각의 경우의 수 노드에 대해 connect 메서드를 호출한다. 단, 깊이가 상수로 지정한 DEP 이상일 경우 다시 호출하지 않는다.

게임 트리를 세우는 동시에 알파-베타 가지치기를 진행하고, 그 후 유전 알고리즘(Genetic Algorithm)을 적용하여 최적수를 선별하고 반환한다. 이 때 Pygene 모듈을 사용한다. Generation 객체에 전달하는 인자는 다음과 같다.

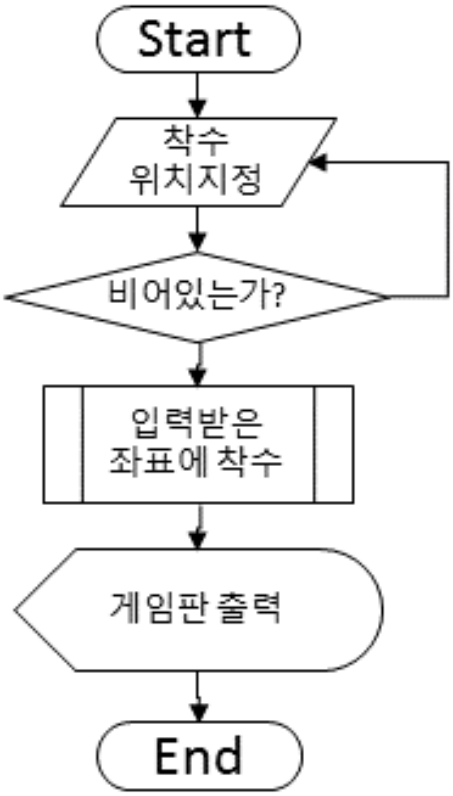
유전자로 저장하는 정보	평가함수	교차 방식	변이 방식
착수 순서(게임트리)	유전자의 순서대로 착수했을 시의 단말 노드의 Board.Func 메서드	순서 교차	비균등 변이

선택 방식	변이 확률	선택압
룰렛휠 연산	0.3	0.5

최적화 메서드는 유전 알고리즘을 통해 최적수를 도출하고, 반환한다.

-Player 클래스

플레이어의 조작을 담당하는 클래스이다. Ply_move 메서드만을 가지며, 이동할 좌표를 입력받아 착수하고, main 파트에 전달한다.



3) Main 파트

메인 프로그램을 담당하는 부분이다. 크게 초기화와 메인 루프로 나뉘며, 위에서 정의한 클래스와 함, 상수를 이용하여 사용자와 삼목 게임을 진행한다.

```
# Main Program
board = Board(INIT_BOARD)  # 게임판 객체 생성
board.move(MID)  # 중앙에 AI가 착수

oBot = AI(board)  # AI 객체 생성
ply = Player()  # 플레이어 객체 생성
```

초기화 파트

게임판 객체 board, 인공지능 객체 oBot, 플레이어 객체 ply를 생성하고 중앙에 AI가 착수한다.

```
while(0):
    if board.move(ply.ply_move(), -1, 1) :  # 플레이어가 착수
        print("VICTORY")
        break

    if board.move(oBot.optimum()) :  # AI가 착수
        print("DEFEAT")
        break

    board.draw()  # 보드 드로우
```

메인 파트

메인 루프를 시작하고 이미 중앙에 착수함으로서 AI의 턴이 지났으므로 플레이어가 먼저 착수를 한다. 승리를 체크하고 AI에게 턴이 넘어간다. AI도 마찬가지로 최적수 메서드를 통해 착수하고, 턴을 넘긴다. 두 에이전트의 턴이 끝나면 보드를 드로우하고 루프를 진행한다.

4) 데이터 분석

완성된 프로그램을 통해 게임을 진행해보면 아래 표와 같은 결과를 보였다.

(O : 컴퓨터 승, X : 플레이어 승, MMA : MiniMax Algorithm)

유전 알고리즘(+MMA) 적용	미니맥스 알고리즘 적용
O	X
O	O
X	X
X	O
O	O
O	X
O	X
O	X
X	O

이를 통해 유전 알고리즘의 효용성을 증명할 수 있었다. 아직 19*19 크기의 바둑판에 유전 알고리즘을 적용하기는 방대한 연산량으로 인해 연산 시간이 부족하다. 하지만 유전 알고리즘을 최적화하고, 경우의 수를 줄이는 조치를 취하거나, 하드웨어의 성능을 끌어올린다면 바둑판 역시 적용될 가능성은 배제할 수 없고, 유전 알고리즘을 보드게임에 적용시키면 성능 향상을 기대할 수 있을 것이다.

Ⅴ. 결론

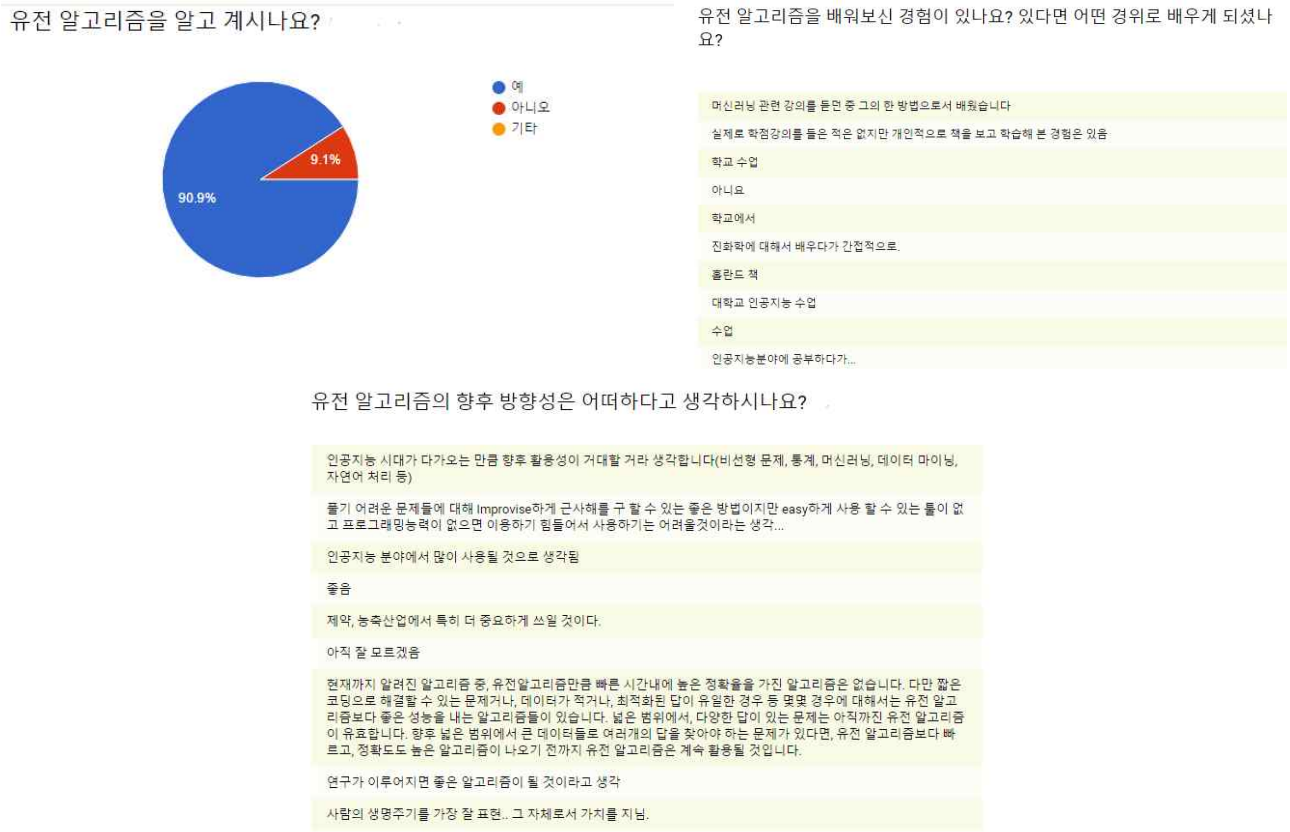
ⅰ. 결론

위의 두 탐구를 통해 유전 알고리즘을 활용하는 방법을 제시했고, 어떻게 활용하는지를 알아보았다. 유전 알고리즘은 염색체-유전자의 데이터 구조를 갖는다면 적용할 수 있고, 그 이점 덕분에 두 번째 탐구의 마지막에서 언급했듯 오목이나 바둑에 사용되어 연산의 정확도를 높일 가능성이 높다.

또한 유전 알고리즘을 이용한 루빅스 큐브 해결 프로그램도 시중에 나와 있는 루빅스 큐브 언스크램블(해결) 프로그램에 비해 처리 속도의 차이를 보였고, 다른 큐브에도 적용할 수 있는 잠재성을 보았다.

ⅱ. 향후 이용성

여러 프로그래밍 커뮤니티의 통계 결과 유전 알고리즘의 향후 방향성에 대해 긍정적이었던 사람이 대다수였고, 인지도도 높으며 미래 전망도 우수하다고 여겨진다. (아래 설문)



두 탐구를 진행하며 보았듯 유전 알고리즘에는 아직 잠재성이 많이 남아 있다. 특히 알파고와 같은 보드게임 인공지능에 적용되면 연산의 정확도를 보강하고 효율적인 연산을 진행하는 데 도움이 될 것이며, 설문 결과에서 보이듯 넓은 범위에서 다양한 해를 찾는 문제에서는 유전 알고리즘이 독보적이다. 인공지능이 새로운 시대를 개척하는 지금, 유전 알고리즘은 앞으로 더욱 발전할 것이라 예상된다.

Ⅵ. 참고 문헌

ⅰ.참고 도서

제리 슬로컴, The Cube, 보누스
정연숙, 파스칼이 들려주는 경우의 수 이야기, (주)자음과 모음
김승태, 피타고라스가 들려주는 리스트 이야기, (주)자음과 모음
문병로, 유전알고리즘, 두양사
레이 커즈와일, 특이점이 온다, 김영사
이시아 모루나/에사키 노리히데, 모두의 라즈베리 파이 with, 파이썬, 길벗미디어
제이슨 R. 브리그스, 누구나 쉽게 배우는 파이썬 프로그래밍, 비제이퍼
강충호 외 7인, 하이탑 중등 3 - 下, 동아출판
Michalewic, Evolutionary Algorithms in Engineering Applications
김의중, 인공지능, 머신러닝, 딥러닝 입문
Nils. J. Nilson, 인공지능-지능형 에이전트를 중심으로

ⅱ. 참고 잡지

수학동아 2012년 2월호
science ollze 2008년 8-9월 호

ⅲ. 참고 인터넷사이트

http://cafe.naver.com/cubemania - 큐브매니아〈네이버 카페〉
http://blog.naver.com/vincentcube/60122473587
vincentcube의 블로그〈네이버 블로그〉
https://brunch.co.kr/@hurderella/27
http://df3714.tistory.com/344
http://www.nicovideo.jp/watch/sm19212299?ref=search_key_video&ss_pos=2&ss_id=b022af68-b530-48e8-932b-b5a034c8f2bc
무니무니교수의 유전알고리즘으로 ~를 가르쳐 보았다 시리즈 - 니코니코동화(일본, 로그인 필요)
http://www.aistudy.com/biology/genetic/operator_moon.htm
유전알고리즘의 연산자들 - 문병로
http://kowon.dongseo.ac.kr/~dkkang/AI2011Fall/W0910.pdf
동서대학교 교육자료
http://www.pgr21.com/pb/pb.php?id=freedom&no=57463
인공지능 이야기 - 유전 알고리즘

ⅳ. 참고 논문

기정원, 루빅스 큐브와 관련된 군 이론
이병두, Analysis of Tic-Tac-Toe Game Strategies using Genetic Algorithm

Ⅴ. 용어 정리

1) 유전 알고리즘 관련 용어

1. 교차(Crossover) : 두 개의 유전체가 각각의 유전자를 조합하여 새로운 염색체를 생산하는 연산
2. 변이(Mutation) : 교차 연산 이후, 확률적으로 유전자의 정보가 바뀌는 연산
3. 세대(Generation) : 염색체들의 집합
4. 자손(Offspring) : 교차, 변이 연산으로서 생성된 새로운 염색체들의 집단
5. 적합도 함수(Fitness Function) : 염색체들의 적합도를 구하는 함수
6. 염색체(Chromosome) : 유전정보를 담고 있는 생물학적인 집합을 연속된 문자열로 추상화한 것
7. 유전자(Gene) : 염색체를 구성하는 요소, DNA와 혼용

2) 큐브 관련 용어

1. 순열성(Permutation) : 조각이 어떻게 바뀌어 있는지를 판단하는 것.
2. 방향성(Orientation) : 조각이 어떻게 돌아가 있는지를 판단하는 것.
3. 바보토끼이빨(Odd Parity) : 귀퉁이 조각과 모서리 조각의 홀짝성이[6] 맞지 않는 상황. 모서리 조각 2개를 바꿔야 큐브가 맞춰지는 형태로 되어 있다.

3) Python 프로그래밍 용어

1. 객체(Object, Instance) : 객체지향 프로그래밍에서의 클래스의 인스턴스, 클래스가 염색체라면 객체는 단일 개체이며, 클래스를 설계도에 비유했을 때는 객체를 건물에 비유할 수 있다
2. 클래스(Class) : 객체지향 프로그래밍에서의 객체를 생성하기 위한 변수와 메소드를 정의하는 틀, 염색체 혹은 설계도 등에 자주 비유된다.
3. 함수(Function) : 소프트웨어에서 특정 동작을 수행하는 일정 코드 부분(= 메서드(Method), 서브루틴(Subroutine), 루틴(Routine), 프로시저(Procedure))
4. 모듈(Module) : Python에서, .py 확장자를 가지며 파일에 저장된 함수나 변수를 사용할 수 있게 만든 일종의 패키지
5. 문자열(String) : 기호의 순차 리스트
6. 변수(Variable) : 수식에 따라서 변하는 값
7. 리스트(List) : ≒1차원 행렬, 문자나 같은 리스트도 담을 수 있다는 점이 다르다.
8. 다차원 리스트 : ≒행렬, 문자나 같은 리스트도 담을 수 있다는 점이 다르다

4) 미주

1) 적합도
2) Michalewic, Evolutionary Algorithms in Engineering Applications
3) OOP의 인스턴스와는 유사하나 다른 개념이다.
4) 삼목
5) 연산자 오버라이딩
6) 큐브에서의 홀짝성 : 귀퉁이 조각과 모서리 조각을 따로 생각해서 조각 2개를 바꾸는 행동을 홀수 번 해서 맞출 수 있으면 홀순열, 짝수 번 해서 맞출 수 있으면 짝순열이다.