Algorithm, 2023
Homework for the latter part of the class

GENETIC ALGORITHM USING EXPRESSION TREES

2021-1327 2021-1362
류경빈 심우현

1 문제 해결 설계

Dynamic programming 을 이용해 6 가지 Feature 을 바탕으로 조합돼 나타나는 현상을 예측하기 위한 GA 구조를 설계.

1.1 해의 표현

Table 1 해의 표현

해 하나를 full binary expression tree 로 표현. Operand 는 리프 노드에, Operator 는 그 외의 노드에 있으며 항상 offspring 을 2 개를 갖는 트리이다. 가능한 모든 연산자 조합을 해로 표현할 수 있기 때문에 이와 같은 해의 표현을 선택함.

2.2 사용한 GA 구조

Table 2 GA 구조

(1) 6 개 이하의 연산자 조합으로 이루어진 해 생성, mse 의 원리로 cost 계산하여 fitness 책정 후 초기 부모 집합으로 사용. **(2)** 생성한 population 중 fitness 가 가장 좋은 두 해를 부모로 선정. 자식은 하나 생성, 해의 각 자리마다 50% 확률로 부모의 유전자 하나 선택, 돌연변이는 해의 각 자리마다 일정 확률로 값을 재선택. 자식이 무효하면 제거. **(3)** 자식은 기존의 population 에서 최악의 fitness 를 가지는 유전자 대체, 만약 기존 population 의 최악의 fitness 가 offspring 의 fitness 보다 좋다면 population 유지. **(4)**Population 중 fitness 가 나쁜 10%의 해를 새로운 랜덤한 해로 대체**(5)** 세대 수를 정한 뒤 그 수 만 큼 세대 교체 후 보유한 최선의 해를 반환.

2.3 사용한 연산자에 대한 설명

Table 3 GA 연산자

* dynamic programming: 수식 트리에 따른 fitness 값을 재귀적으로 계산하고, memo 에 수식 트리와 fitness 를 저장해 효율적으로 해당하는 수식 트리의 fitness 를 반환하도록 dynamic programming 으로 구현. * generate random population: 사용 가능한 operator 를 random 하게 골라 구성해 연산식 구성. Operator 수에 따라 random 하게 feature attribute 을 선택해 각 데이터의 해의 결과를 측정 후 result attribute 을 추가적으로 사용해 mse 기법을 이용한 cost 측정. * selection: Elitisim 사용. Fitness 를 기준으로 가장 우월한 두 해를 부모로 선택. * crossover: 두 부모의 유전자를 crossover 해 새로운 offspring 해 생성. * mutation: Leaf node 를 제외한 수식을 나타내는 노드 중 random 하게 노드를 선택 일정 확률로 이를 수행함. * replacement: steady-state GA 방식 이용, 기존 population 에서 최악의 fitness 를 가지는 해를 대체한다. 이에 더해 가장 안 좋은 10%의 population 을 아예 새로운 트리들로 교체

* crossover: 랜덤하게 edge 선택 후 자르고 subtree 를 바꿈. 1. Random 하게 왼쪽 또는 오른쪽 edge 를 선택할지 고름. 이때 Full binary tree 를 위배하는 특성의 트리가 생성될 수 있기 때문에 leafnode 를 연결하는 edge 는 고르지 않음 2. Node 와 edge 의 상관관계를 파악해 두개의 이진트리의 subtree 를 교환 3. 새롭게 형성된 offspring 트리 반환 * mutation: Leaf node 를 제외한 수식을 나타내는 노드 중 random 하게 노드를 선택 확률에 따라 기존의 operator node 대체 . 1. Expression tree 를 순회하며 leaf node 가 아닌 노드 random 하게 선택 2. Operator 랜덤하게 선택해서 기존의 operator 대체 * replacement: Steady-state GA 사용. 최소 fitness 유전자와 자식 유전자 fitness 비교 및 교체. Variation 을 주기 위해 Fitness 를 정렬해 가장 안 좋은 10%의 population 을 아예 새로운 트리들로 교체.

3 Dynamic programming 을 활용한 방안

해당 수식 트리의 fitness 값을 구하는 과정에서 Cost 함수를 호출할 때마다 수식 트리의 fitness 값이 반환되고, memo 에 fitness 가 저장된다. 여기서 memo 는 hash table 을 이용했고, 수식 트리(string type)를 key 값으로 갖도록 구현했다. 따라서, 이미 memo 에 존재하는 수식트리가 Cost 함수를 거쳐 calculateCost 함수에 들어오게 되면 추가적인 계산 없이 바로 memo 에 저장된 값을 반환하도록 구현했다. Recursion 부분은 calculateCost 함수에서 재귀 호출을 통해 dataset 의 1 ~ 200 행까지의 개별 fitness 값을 계산하고 모두 더한 값을 cal[key]에 저장하도록 했다. 여기서 cal[key]는 해당 수식 트리의 fitness 값만 계산한다.

4 문제 해결 시도

각자 GA 설계 후 수식 트리가 최적의 fitness 값으로 수식을 출력하는지 확인한다. (Fitness 는 'average error you can obtain from 200 data'가 아닌 'sum of error/2.0'으로 측정하였다. 따라서 다음 fitness 는 'average error you can obtain from 200 data'와 100 배 차이나는 값이다.')

4.1 테스트 컴퓨터 사양과 정답 도출에 걸린 시간

컴퓨터 사양은 프로세서가 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 이고, RAM 은 32.0GB 이다. GA run 을 30, 60, 90 으로 설정했을 때의 따라 generation, population 을 다르게 설정해 최적의 fitness 값과 수식 트리를 살펴보고자 한다. 실제로 돌렸을 때 GA run 의 횟수를 늘려도 대체로 비슷한 fitness 값을 얻는다. 한 가지 아쉬운 점은 Replace 시 안 좋은 fitness 의 해들을 새로운 랜덤한 트리로 대체하는 방식 때문에 population 을 조금이라도 늘리면 run time 큰 폭으로 증가한다는 점이다.

Table 4 Sample Test 통계

GA run	generation	population	최적 fitness	걸린 시간	수식 트리
30	100	20	23.1102	3:03	(cosh(tan(cos(sinh(tanh(4.030000+D)+D)+-1.160000)+0.240000)+B))
60	50	50	21.9477	38:13	(cosh(((sin(tanh(B+D)+B))/A)+B))

90	20	20	22.6552	1:05	(cosh(exp(np.log(np.log(D+C)+E)+-7.680000)+B))

4.2 최종 수식 도출

결과적으로 여러 번의 ga run을 거친 결과 특성이 약간씩 바뀌는 차이가 있지만, 전반적인 수식의 형태는 **(np.log(np.log(np.cosh((D*B)+B)+C)+B))**가 최종 수식으로 도출됐다.

5 DISCUSSION - 프로젝트 리뷰

팀원별로 프로젝트 진행 중 느낀 점, 잘 안 되는 점, 의외의 현상, 예상대로 된 점 등을 서술한다.

5.1 심우현

Cost 함수에 Dynamic programming 의 memo 와 recursion 을 적용하는데 있어 여러 시행착오가 있었지만, 이번 기회에 Dynamic programming 의 작동 원리에 대해 더 배울 수 있었다. 예를 들어, memo 를 벡터 형태로 만들어야만 된다는 틀에 사로잡혀 구현 진행에 어려움을 가졌다. 하지만, option 문제에 적용함으로써 이번 기회에 구현해 놓은 dynamic programming 의 틀을 다른 문제에도 유연하게 적용할 수 있음을 확인했다. 또한, 그동안 다뤄온 유전 알고리즘 과제를 수식 트리에 적용해 문제를 해결하는 과정에서, 따로 배웠던 개념을 융합해 구현해보니 실제 알고리즘과 자료구조가 실생활에 어떻게 적용되는지 직접 경험할 수 있어 좋았다.

5.2 류경빈

Expression tree 를 사용한 해의 생성에 많은 고민이 필요하였다. 어떻게 하면 모든 경우의 수를 다 고려할 수 있을지 그리고 불가능한 연산은 어떻게 처리해야 하는지를 기존에 배운 트리 개념을 이용해 함수를 새롭게 구성하는 것이 흥미로움과 동시에 고민이 많이 됐던 것 같다. Crossover 과 mutation 역시 기존의 과제에서는 다소 심플했지만 트리의 특성을 고려하여 코드를 구성하며 자료구조에서 배운 트리에 대해 더 깊이 있는 이해가 생긴 것 같다.