실습 2: N-Queens Problem and Local Search

기한: 2021 년 4 월 18 일 일요일 11:59PM

학번: 2016160311 이름: 이재윤

테스트 함수를 사용하여 알고리즘이 실행되는데 걸리는 시간을 제시하고, 이 시간과 알고리즘의 원리를 기반하여 알고리즘들의 time complexity 을 자유롭게 비교 및 분석하시오.

a. Hill-Climbing vs Stochastic Hill-Climbing

Hill-Climbing 알고리즘의 실행 결과

```
Running tests on  <function hillClimb at 0x01ABEC40>
------------------
Size= 4
Run 1 : quality =  5.0 out of 6.0 count = 1 time = 0.00000
Run 2 : quality =  6.0 out of 6.0 count = 2 time = 0.00100
Run 3 : quality =  6.0 out of 6.0 count = 4 time = 0.00000
Run 4 : quality =  5.0 out of 6.0 count = 4 time = 0.00100
Run 5 : quality =  6.0 out of 6.0 count = 4 time = 0.00000
------------------
Size= 5
Run 1 : quality =  10.0 out of 10.0 count = 4 time = 0.00100
Run 2 : quality =  10.0 out of 10.0 count = 10 time = 0.00300
Run 3 : quality =  10.0 out of 10.0 count = 12 time = 0.00200
Run 4 : quality =  10.0 out of 10.0 count = 7 time = 0.00200
Run 5 : quality =  9.0 out of 10.0 count = 2 time = 0.00100
```

Stochastic Hill-Climbing 알고리즘의 실행 결과

```
Running tests on  <function stochHillClimb at 0x01AD7340>
------------------
Size= 4
Run 1 : quality =  5.0 out of 6.0 count = 1000 time = 0.11203
Run 2 : quality =  6.0 out of 6.0 count = 2 time = 0.00000
Run 3 : quality =  6.0 out of 6.0 count = 12 time = 0.00200
Run 4 : quality =  6.0 out of 6.0 count = 10 time = 0.00100
Run 5 : quality =  6.0 out of 6.0 count = 11 time = 0.00100
------------------
Size= 5
Run 1 : quality =  9.0 out of 10.0 count = 1000 time = 0.14203
Run 2 : quality =  8.0 out of 10.0 count = 1000 time = 0.14803
Run 3 : quality =  9.0 out of 10.0 count = 1000 time = 0.14203
Run 4 : quality =  10.0 out of 10.0 count = 10 time = 0.00100
Run 5 : quality =  8.0 out of 10.0 count = 1000 time = 0.14705
```

테스트를 진행하면서 max rounds 를 1000 으로 설정해두고 5 회 반복하여 시행하였으며 보드의 크기는 4 인 경우와 5 인 경우로 나누어 시행하였다. 이에 따른 시간은 평균적으로 보드의 사이즈가 4 일 때 Hill-Climbing 의 경우 0.0004 초 소요되었으며 Stochastic Hill-Climbing 의 경우 0.0232 초가 소요되었다. 보드의 사이즈가 5 인 경우 Hill-Climbing 은 0.0018 초 소요되었으며 Stochastic Hill-Climbing 은 0.1160 초가 소요되었다.

Hill-Climbing 알고리즘의 경우 초기 상태에서 시작하여 queen 을 한 칸 움직였을 때의 모든 경우의 수들 중 가장 나은 경우를 선택하는 방법으로 local maxima 에 도달했을 때의 상태를 반환하기 때문에 많은 반복을 거치지 않고 금세 결과값을 반환하는 것을 확인할 수 있었다. 하지만 그 결과값이 global maximum 값인 max value 에 도달하는 경우는 보장되지 않았다.

Stochastic Hill-Climbing 알고리즘의 경우 현재 상태에서 queen 을 한 칸 움직였을 때의 경우를 임의로 k 개 (이 실험에서는 5 개를 선택) 골라 그 중 현재 상태의 value 값보다 더 나은 경우를 선택(중복된 값이 있는 경우 roulette wheel selection 을 사용)하여 반복하는 방법으로 max value 에 도달하거나 max round 에 도달할 때까지 반복하였다. 따라서 여러 번 반복하여도 max value 에 도달하지 못하는 경우 시간이 오래 걸렸지만 local maxima 에 빠지지 않도록 하였으므로 global maxima 를 반환하는 확률을 높일 수 있었다.

이러한 원리의 차이에 의해 결과값에서 볼 수 있듯이 Stochastic Hill-Climbing 의 경우 Hill-Climbing 보다 평균적으로 더 많은 시간이 소요되었지만 보드의 사이즈가 4 일 때 Stochastic Hill-Climbing 이 Hill-Climbing 보다 max value 에 더 많이 도달한 것을 확인할 수 있었다. 다만 사이즈가 5 인 경우에서는 Stochastic Hill-Climbing 이 오히려 Hill-Climbing 보다 value 값이 좋지 않은 것을 확인할 수 있었는데, 이는 Stochastic Hill-Climbing 알고리즘의 구조상 임의로 k 개의 이웃을 선택하는 과정에서 value 값이 더 나은 이웃을 고를 확률이 Hill-Climbing 보다 낮아지게 되고 global maximum 에 도달하지 못한 상태로 max round 에 도달하면 강제로 그때의 값을 반환하게 되기 때문에 Hill-Climbing 의 경우보다 낮은 value 값을 반환하게 된 것으로 보인다. 이렇듯 Stochastic Hill-Climbing 이 Hill-Climbing 보다 훨씬 더 많은 loop 를 반복하므로 time complexity 역시 Stochastic Hill-Climbing 알고리즘이 Hill-Climbing 알고리즘보다 크다고 할 수 있다.

b. Hill-Climbing vs Simulated Annealing

Hill-Climbing 알고리즘의 실행 결과

```
Running tests on  <function hillClimb at 0x0238EC40>
------------------
Size= 4
Run 1 : quality =  5.0 out of 6.0 count = 0 time = 0.00000
Run 2 : quality =  5.0 out of 6.0 count = 4 time = 0.00100
Run 3 : quality =  5.0 out of 6.0 count = 4 time = 0.00100
Run 4 : quality =  6.0 out of 6.0 count = 3 time = 0.00000
Run 5 : quality =  5.0 out of 6.0 count = 2 time = 0.00100
------------------
Size= 5
Run 1 : quality =  10.0 out of 10.0 count = 11 time = 0.00189
Run 2 : quality =  8.0 out of 10.0 count = 3 time = 0.00152
Run 3 : quality =  10.0 out of 10.0 count = 5 time = 0.00099
Run 4 : quality =  10.0 out of 10.0 count = 4 time = 0.00100
Run 5 : quality =  9.0 out of 10.0 count = 6 time = 0.00100
```

Simulated Annealing 알고리즘의 실행 결과

```
Running tests on  <function simAnnealing at 0x023C03D0>
------------------
Size= 4
Run 1 : quality =  6.0 out of 6.0 count = 107 time = 0.00250
Run 2 : quality =  6.0 out of 6.0 count = 150 time = 0.00350
Run 3 : quality =  6.0 out of 6.0 count = 36 time = 0.00100
Run 4 : quality =  6.0 out of 6.0 count = 600 time = 0.01400
Run 5 : quality =  6.0 out of 6.0 count = 5 time = 0.00000
------------------
Size= 5
Run 1 : quality =  10.0 out of 10.0 count = 139 time = 0.00400
Run 2 : quality =  7.0 out of 10.0 count = 1001 time = 0.03001
Run 3 : quality =  10.0 out of 10.0 count = 490 time = 0.01500
Run 4 : quality =  10.0 out of 10.0 count = 562 time = 0.01700
Run 5 : quality =  10.0 out of 10.0 count = 40 time = 0.00100
```

비교 테스트를 진행하기에 앞서 Hill-Climbing 의 max rounds 값인 1000 과 loop 의 횟수를 동일하게 맞춰주기 위해 Simulated Annealing 의 초기 온도 값을 100.0 으로 설정하고 loop 마다 0.1 도씩 낮춰주었다. 보드의 사이즈는 4 일 때와 5 일 때로 나누어 5 번 반복 시행한 결과를 나타내었으며 5 번의 평균적인 소요 시간은 사이즈가 4 일 때 Hill-Climbing 의 경우 0.0006 초, Simulated Annealing 의 경우 0.0042 초가 소요되었고, 사이즈가 5 일 때 Hill-Climbing 은 0.0013 초, Simulated Annealing 은 0.0134 초 걸렸다.

Simulated Annealing 알고리즘의 경우 현재 상태와 queen 을 임의로 한 칸 움직였을 때의 상태를 비교하여 임의로 움직인 상태의 value 값에서 현재 상태의 value 값을 뺐을 때의 값(difference)이 무엇이냐에 따라 전략이 달라진다. 만약 difference 의 값이 양수인 경우 임의로 움직인 상태가 더 나은 값을 나타내므로 현재 상태를 갱신해준다. 반면 difference 값이 음수인 경우 현재 온도에 따라 확률적으로 현재 상태를 바꿀 수도 있고 바꾸지 않을 수도 있는데, 이 확률은 온도가 높을수록 현재 상태를 바꾸게 될 가능성이 커진다. 하지만 loop 를 거듭할수록 온도가 낮아지게 되므로 현재 상태보다 더 나쁜 상태를 고를 가능성이 낮아지게 된다.

Simulated Annealing 알고리즘은 온도가 0 이하로 내려가지 않는 이상 계속해서 global maximum 인 max value 를 찾고자 노력하므로 local maxima 에 빠질 위험이 있는 Hill-Climbing 보다 더 좋은 결과값을 찾는다. 하지만 그 과정에서 많은 loop 를 돌리게 되므로 시간은 더 오래 걸리게 된다. 결과값에서 알 수 있듯이 보드의 사이즈가 4 일 때와 5 일 때 모두 Simulated Annealing 이 Hill-Climbing 보다 더 많은 시간이 소요되었다. 따라서 time complexity 는 Hill-Climbing 보다 Simulated Annealing 이 더 크다고 할 수 있다.

```
Running tests on  <function simAnnealing at 0x020973D0>
------------------
Size= 4
Run 1 : quality =  6.0 out of 6.0 count = 103 time = 0.00200
Run 2 : quality =  6.0 out of 6.0 count = 194 time = 0.00500
Run 3 : quality =  6.0 out of 6.0 count = 27 time = 0.00100
Run 4 : quality =  6.0 out of 6.0 count = 47 time = 0.00100
Run 5 : quality =  6.0 out of 6.0 count = 262 time = 0.00600
------------------
Size= 5
Run 1 : quality =  10.0 out of 10.0 count = 180 time = 0.00500
Run 2 : quality =  10.0 out of 10.0 count = 86 time = 0.00300
Run 3 : quality =  10.0 out of 10.0 count = 432 time = 0.01300
Run 4 : quality =  10.0 out of 10.0 count = 43 time = 0.00100
Run 5 : quality =  10.0 out of 10.0 count = 714 time = 0.02100
```

참고로 Simulated Annealing 의 초기 온도 값을 100.0 이 아닌 10.0 으로 설정하고 대신 loop 마다 감소하는 온도를 0.01 로 설정하여 최대 loop 횟수가 1000 이 되도록 설정한 경우 평균 소요시간이 사이즈 4 일 때 0.0030 초, 사이즈 5 일 때 0.0086 초로 앞선 테스트의 설정 값보다 유의미하게 낮은 시간이 소요되었는데, 이러한 차이가 생긴 이유는 온도에 따른 선택 확률의 값에 차이가 생겼기 때문인 것으로 보인다.

c. Hill-Climbing vs Beam Search

Hill-Climbing 알고리즘의 실행 결과

```
Running tests on  <function hillClimb at 0x0172EC40>
------------------
Size= 4
Run 1 : quality =  6.0 out of 6.0 count = 0 time = 0.00000
Run 2 : quality =  6.0 out of 6.0 count = 2 time = 0.00000
Run 3 : quality =  5.0 out of 6.0 count = 1 time = 0.00000
Run 4 : quality =  5.0 out of 6.0 count = 2 time = 0.00100
Run 5 : quality =  5.0 out of 6.0 count = 3 time = 0.00000
------------------
Size= 5
Run 1 : quality =  8.0 out of 10.0 count = 1 time = 0.00100
Run 2 : quality =  10.0 out of 10.0 count = 4 time = 0.00100
Run 3 : quality =  8.0 out of 10.0 count = 0 time = 0.00000
Run 4 : quality =  8.0 out of 10.0 count = 0 time = 0.00000
Run 5 : quality =  10.0 out of 10.0 count = 6 time = 0.00200
```

Beam Search 알고리즘의 실행 결과

```
Running tests on  <function beamSearch at 0x0175E418>
------------------
Size= 4
Run 1 : quality =  6.0 out of 6.0 count = 1 time = 0.00000
Run 2 : quality =  6.0 out of 6.0 count = 2 time = 0.00200
Run 3 : quality =  6.0 out of 6.0 count = 2 time = 0.00200
Run 4 : quality =  6.0 out of 6.0 count = 1 time = 0.00000
Run 5 : quality =  6.0 out of 6.0 count = 1 time = 0.00100
------------------
Size= 5
Run 1 : quality =  10.0 out of 10.0 count = 2 time = 0.00400
Run 2 : quality =  10.0 out of 10.0 count = 4 time = 0.00700
Run 3 : quality =  10.0 out of 10.0 count = 2 time = 0.00300
Run 4 : quality =  10.0 out of 10.0 count = 2 time = 0.00200
Run 5 : quality =  10.0 out of 10.0 count = 2 time = 0.00500
```

알고리즘의 초기 설정은 Hill-Climbing 의 max rounds 와 Beam Search 의 stop limit 값을 둘 다 1000 으로 동일하게 설정하였으며 Beam Search 의 경우 state 의 개수를 10 개로 유지하였고 보드의 사이즈를 4 일 때와 5 일 때로 나누어 5 회씩 반복하였다. 그 결과 평균적인 소요시간은 Hill-Climbing 의 경우 사이즈가 4 일 때 0.0002 초, 사이즈가 5 일 때 0.0008 초였으며 Beam Search 의 경우 사이즈가 4 일 때 0.0010 초, 사이즈가 5 일 때 0.0042 초로 나타났다.

Beam Search 알고리즘의 경우 초기 상태부터 k 개(이 실험에서는 10 개)의 state 를 선택하여 계속해서 k 개를 유지하면서 더 나은 state 를 찾아가는 전략을 사용한다. 선택된 k 개의 states 에서 각각의 state 중 가장 나은 neighbor 를 골라서 만약 그 값이 현재의 states 의 value 값들 중 가장 나쁜 값보다 더 나은 값이라면 현재의 states 의 집합을 갱신한다. 갱신된 state 의 value 가 max value 에 해당한다면 그 값을 반환하고 알고리즘은 종료된다.

결과값을 보면 count 의 횟수는 두 알고리즘 모두 큰 차이가 없었으나 소요된 시간은 Beam Search 가 Hill-Climbing 보다 다소 더 오래 걸리는 것을 확인할 수 있었다. 이러한 차이가 생긴 이유는 Hill-Climbing 의 경우 하나의 state 를 기준으로 더 나은 값들을 찾아가지만 Beam Search 의 경우 10 개의 state 를 계속 유지하면서 비교하여 더 나은 값들을 찾아내기 때문인 것으로 보인다. 또한 Beam Search 의 경우 만일 stop limit 으로 설정된 count 이내에 max value 를 찾지 못하게 되면 여러 차례 loop 를 돌게 되어 훨씬 더 많은 시간이 소요된다. 따라서 time complexity 는 Hill-Climbing 보다 Beam Search 가 더 크다고 할 수 있다. 다만 Beam Search 는 충분히 적은 count 로도 global maximum 을 찾을 수 있다는 장점이 존재했다.

d. Genetic Algorithm vs Beam Search

Genetic Algorithm 의 실행 결과

```
Running tests on  <function geneticAlg at 0x01F47538>
------------------
Size= 4
Run 1 : quality =  6.0 out of 6.0 count = 366 time = 0.05301
Run 2 : quality =  6.0 out of 6.0 count = 228 time = 0.03401
Run 3 : quality =  6.0 out of 6.0 count = 1283 time = 0.18408
Run 4 : quality =  6.0 out of 6.0 count = 334 time = 0.04701
Run 5 : quality =  6.0 out of 6.0 count = 92 time = 0.01300
------------------
Size= 6
Run 1 : quality =  12.0 out of 15.0 count = 2000 time = 0.43910
Run 2 : quality =  12.0 out of 15.0 count = 2000 time = 0.43610
Run 3 : quality =  15.0 out of 15.0 count = 1704 time = 0.36808
Run 4 : quality =  13.0 out of 15.0 count = 2000 time = 0.43861
Run 5 : quality =  13.0 out of 15.0 count = 2000 time = 0.43510
```

Beam Search 알고리즘의 실행 결과

```
Running tests on  <function beamSearch at 0x01F47418>
-----------------
Size= 4
Run 1 : quality =  6.0 out of 6.0 count = 1 time = 0.00100
Run 2 : quality =  6.0 out of 6.0 count = 1 time = 0.00000
Run 3 : quality =  6.0 out of 6.0 count = 1 time = 0.00100
Run 4 : quality =  6.0 out of 6.0 count = 2 time = 0.00100
Run 5 : quality =  6.0 out of 6.0 count = 2 time = 0.00200
-----------------
Size= 6
Run 1 : quality =  14.0 out of 15.0 count = 2000 time = 6.41661
Run 2 : quality =  14.0 out of 15.0 count = 2000 time = 6.44709
Run 3 : quality =  15.0 out of 15.0 count = 2 time = 0.00400
Run 4 : quality =  15.0 out of 15.0 count = 3 time = 0.01000
Run 5 : quality =  14.0 out of 15.0 count = 2000 time = 6.67546
```

테스트 초기 설정으로는 Genetic Algorithm 의 max generation 과 Beam Search 의 stop limit 을 2000 으로 동일하게 설정해주었고 Genetic Algorithm 의 population 과 Beam Search 의 state 개수도 10 개로 동일하게 설정하여 진행하였다. Genetic Algorithm 의 crossover 확률은 0.8, mutation 확률은 0.01 로 설정하였다. 보드의 사이즈는 4 일 때와 6 일 때로 나누어 5 회씩 반복하였다. 그 결과 평균적인 소요시간은 Genetic Algorithm 의 경우 사이즈가 4 일 때 0.0662 초, 사이즈가 6 일 때 0.4234 초가 걸렸으며 Beam Search 의 경우 사이즈가 4 일 때 0.0010 초, 사이즈가 6 일 때 3.9106 초가 걸렸다. Genetic Algorithm 의 경우 초기에 k 개의 population 을 생성하여 각 individual 의 fitness 값을 구하고 fitness 의 크기만큼 가중치를 두어 roulette wheel selection 으로 k 개의 parent 를 선택한 뒤 설정한 확률만큼 crossover 와 mutation 을 시행한다. 이렇게 얻은 새로운 population 에서 maximum fitness 값이 나올 때까지 계속 반복하거나 max generation 에 도달하면 그때의 population 에서 가장 좋은 fitness 값을 가진 individual 의 value 를 반환하고 알고리즘이 종료된다.

두 알고리즘의 결과값은 상당한 차이점이 발견되었는데, 우선 보드의 사이즈가 4 인 경우에는 Beam Search 가 Genetic Algorithm 보다 훨씬 시간이 적게 걸렸지만 보드의 사이즈가 6 인 경우에는 Genetic Algorithm 이 Beam Search 보다 훨씬 시간이 적게 소요되는 것을 확인할 수 있었다. 그리고 같은 횟수의 반복을 하게 되더라도 Genetic Algorithm 이 Beam Search 보다 훨씬 적은 시간이 걸렸으며 Beam Search 는 반복하는 횟수가 커질수록 더 많은 시간이 걸리는 것을 확인할 수 있었다. 사이즈가 작은 보드의 경우 Beam Search 가 유리하지만 사이즈가 충분히 큰 경우 Genetic Algorithm 의 시간이

훨씬 적게 걸리므로 Genetic Algorithm 의 time complexity 가 Beam Search 보다 작다고 할 수 있다. 하지만 반복 횟수에 대비했을 때는 Beam Search 가 더 좋은 value 를 얻을 수 있었다. 이러한 차이가 생긴 이유는 Beam Search 의 경우 Genetic Algorithm 과 달리 crossover 나 mutation 이 일어나지 않기 때문에 빠르게 더 나은 value 값을 찾아낼 수 있지만 엘리트 전략으로 인해 좁은 region 으로 집중되면서 값이 정체되었기 때문인 것으로 보인다.