**실습2: N-Queens Problem and Local Search**

기한: 2021년 4월 18일 일요일 11:59PM

학번 이름: 2017320114 최재원

알고리즘 구현시 요구사항

* Hill-Climbing 알고리즘과 마찬가지로 verbose 변수가 True일 때, 함수 안에서 다음의 변수를 프린트 해야합니다.
  + 알고리즘 시작 프린트문
  + Loop 마다 현재 count와 current state
  + Loop마다 찾아진 best neighbor 또는 선택된 다음 state
  + (simAnnealing 경우) loop마다 현재 temperature
  + (simAnnealing 경우) value가 더 높은 state이 찾아졌을 때, 더 낮은 state이 찾아졌을 때 모두 state 프린트
  + 알고리즘 끝 프린트문
  + 알고리즘의 결과로 나온 마지막 state (solution 또는 final state)
  + 알고리즘이 끝났을 때의 count
* 구현한 모든 함수에 docstring 작성 (한글/영어 다됨)

1. 아래 의사코드를 바탕으로 localSearch.py에 stochastic hill-climbing 함수를 구현하시오.
   1. Input: startState, numNeighs = 5, maxRounds = 1000
   2. Output: value, maxValue, count

|  |
| --- |
| Algorithm stochHillClimb(startState, numNeighs=5, maxRounds=1000)   * **set** curr = startState * **set** value = value of current state * **set** maxValue = maximum value of current state * initialize count to zero * **while** current value is less than maximum value **and** count is less than maxRounds   + **set** neighs = generate numNeighs number of random neighbors from current state   + get the best neighbor using function stochFindBestNeighbor(neighs, value)   + **if** result is not False **then**     - **set** curr = the best neighbor     - **set** value = the value of the best neighbor   + increment count by 1 * **return** value, maxValue, count   Helper Function stochFindBestNeighbor(neighbors, currValue)   * **set** bestNeighs = an empty list to store best neighbors * **for** a neighbor **in** neighbors:   + Get current value of the neighbor   + **if** the neighbor’s value is greater or equal to currValue     - Append neighbor to bestNeighs * **if** list of best neighbors is empty   + **return** **False** * **deltaValues = [neigh.getValue() – currValue for neigh in bestNeighs]** * **bestPos = rouletteSelect(deltaValues)** * **return bestNeighs[bestPos]** |

* stochFindBestNeighbor 함수는 neighbor 리스트와 현재 state의 value를 받아서 현재 value보다 더 높은 value를 가진 하나의 neighbor을 반환합니다. 현재 value보다 나은 neighbor을 선택할 때 roulette wheel selection을 사용합니다. 이 함수는 stochastic hill climbing 함수에서 랜덤으로 neighbors를 생성한 후 현재 state의 value보다 높은 하나의 neighbor을 선택할 때 사용됩니다.

1. 아래 의사코드를 바탕으로 localSearch.py에simulated annealing 함수를 구현하시오.
   1. Input: startState, initTemp
   2. Output: currValue, maxValue, count

|  |
| --- |
| Algorithm simAnnealing(startState, initTemp=5.0)   * **set** currTemp = initTemp * **set** currState = startState * **currState.setPrintMode(full=False)** * **set** currValue = value of the current state * **set** maxValue = maximum value of current state * Initialize count * **while** currTemp is greater than zero **and** currValue ls less than maxValue **do**   + **set** nextState = make random move from current state   + **set** nextValue = get value of nextState   + calculate difference between nextValue and currValue   + **if** the difference is greater than or equal to zero **then**     - **set** currState = nextState     - **set** currValue = nextValue   + **else**     - **set** threshold = e^(difference/currTemp)     - generate a random value     - **if** random value is less than or equal to threshold **then**       * **set** currState = nextState       * **set** currValue = nextValue   + decrement currTemp by 0.1   + increment count by 1 * **return** currValue, maxValue, count |

1. 아래 의사코드를 바탕으로 localSearch.py에beam search 함수를 구현하고, 세 개의 helper function의 docstring을 완성하시오.
   1. Input: stateGen, numStates, stopLimit
   2. Output: value of final state, maxValue, count

|  |
| --- |
| Algorithm beamSearch(nqueenNum, numStates=10, stopLimit=500)   * declare an empty list named currStates * **loop for** numStates **do**   + **set** nextState = generate a random board of nqueenNum   + nextState.setPrintMode(full=**False**)   + **append** nextState to currStates * **set** maxValue = get maximum value of the first item of currStates * **call** sortByValue(currStates) * initialize count to zero * initialize foundOptimal to False * **while** optimal solution is not found **and** count is less than stopLimit **do**   + declare an empty list named bestNNeighs   + **loop for** each nextState in currStates **do**     - **set** neighs = get all neighbors of nextState     - (bestNNeighs, foundOptimal) = keepBestNNeighbors(bestNNeighs, neighs, numStates, maxValue)     - **if** optimal solution is found **do**       * **break**   + **set** currStates = bestNNeighs   + increment count   + set state = first item of currStates * **return** value of state, maxValue, count |

* Beam search는 랜덤한k states로 시작하기 때문에, startState보다 정해진 크기의 랜덤한 N-Queen state을 생성하도록 생성할 보드의 크기(queen의 개수)를 입력으로 받습니다.
* 함수 sortByValue, keepBestNNeighbors, insertState은 이미 주어졌습니다. beamSearch 함수를 구현할 때, 이 함수를 사용하기 바랍니다. 다만 이 세 함수들에는 docstring이 없습니다. 함수를 이해하고, 세 함수의 doctring을 작성하시기 바랍니다. Docstring은 함수의 1) 인풋과 2) 아웃풋을 묘사해야 하며 3) 함수의 기능을 간략하게 요약해야 합니다.

1. 테스트 함수를 사용하여 알고리즘이 실행되는데 걸리는 시간을 제시하고, 이 시간과 알고리즘의 원리를 기반하여 알고리즘들의 time complexity을 자유롭게 비교 및 분석하시오.
   1. Hill-Climbing vs Stochastic Hill-Climbing
   2. Hill-Climbing vs Simulated Annealing
   3. Hill-Climbing vs Beam Search
   4. Genetic Algorithm vs Beam Search

주의사항:

* 테스트 함수를 실행하기 전, localSearch.py의 verbose 변수를 False로 설정하시오.
* 테스트 함수의 reps, sizeList 변수와 알고리즘의 변수(maxRounds, numNeighs, initTemp, stopLimit 등등)를 자유롭게 변경하여 실험하시오.
* beam search와 genetic algorithm은 testVaryingPops 함수를 사용하여 테스트하고, 나머지 함수는 testRandomStarts 함수로 테스트 하시오.
* 시간을 제시할때는 모든 횟수를 제시하여도 되고 평균을 제시하여도 됩니다.

실습 채점 기준

* 실습 수업에서 진행한 코드를 모두 작성하였는가
* Stochastic hill climing, simulated annealing, beam search를 의사코드를 바탕으로 올바르게 구현 하였는가
* 함수에 verbose 변수가 참일 때, 요구되는 변수들이 출력되게 구현하였는가
* 구현한 모든 함수에 docstring을 작성하였는가
* Beam search의 세 helper function의 docstring을 올바르게 작성 하였는가
* 비교 및 분석에서 1) 테스트 실행 결과를 서술하였으며 2) 알고리즘의 원리/접근방법을 설명하였고 3) 1과 2에 기반하여 각 알고리즘의 time complexity를 비교하였는가
* 제출한 코드에 비교 및 분석을 위한 코드가 작성되어 있는가