COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

Introduction to Artificial Intelligence

Exercise 3 - Local search: hillclimb

October 5, 2022

OPTIMISATION PROBLEM

Type of task, that we will deal with today, is called the "optimization problem". Our goal is to find maximum/minimum of an unknown function which we can only use as a black-box - we provide it with input, and it returns an output (number). Input can be represented by arbitrary set of parameters, while output is always a single number. The task is to find such input, for which the output will be maximal (maximization) or minimal (minimization).
In order to formulate a given task (e.g. "8-queens") as an optimization problem, we need to specify three fundamental functions:

- *random_state()*: generates a random state (values of input parameters) in which the algorithm can start
- *neighbors(x)*: returns "neighbouring" states for state x - such states that slightly differ, but not too much.
- *fitness(x)*: evaluates the state x - this is the function that will be optimized.

HILL-CLIMBING

The simplest optimization algorithm - find the best neighbouring state and move there; if no neighbouring state is better than the current one, then we're at the top and finish:

Algorithm 1 - hill-climbing algorithm

```
1: procedure HILL_CLIMB()
2:     x = random_state()
3:     while True do
4:         best_neighbor = best neighbouring state of x
5:         if fitness(best_neighbor) ≤ fitness(x) then
6:             return x
7:         x = best_neighbor
```

Program:

The prepared program template contains three classes: $OptimizeMax$ and derived $MysteryFunction$ and $EightQueens$.

Abstract class $OptimizeMax$ is built to solve any optimization (maximization) problem. For the optimization, it can use the function, which you will have to implement:

- *hillclimb(x, max_steps)* - hill-climbing algorithm from given starting state x, while the number of steps is limited by max_steps.

In case of the MysteryFunction, in the HC algorithm call $self.plot(x, self.fitness(x))$ in each iteration (i.e. each time you obtain new value of x) - you'll see the progress of the optimization. All other functions are implemented in derived classes, not in the $OptimizeMax$ class itself. Each derived class represents single optimization problem, with defined functions:

- *fitness(x)* - evaluation of state x - the greater the fitness = the better state.
- *neighbors(x)* - returns list of neighbouring states for state x.
- *random_state()* - returns a random state, from which we can start the search

You'll be solving two optimization problems:

- Searching for maximum of a "Mystery function" ($MysteryFunction$ class). This function has global maximum in $x = 0$, but it also contains handful of local maxima. Hill-climbing will mostly end up in a local maximum (depending on random initialization). In this task you'll see the progress of your algorithm being plotted onto a graph.
- Eight queens puzzle ($EightQueens$ class). The task is to set the position of 8 queens on a chessboard, such that none of the queens is attacked. It's important to choose a convenient representation of the states - e.g. list of tuples (x, y), others are also possible (and may be better). Write a short explanation of your choice of state representation. Non-trivial part of this problem is also designing a suitable fitness function.

Task 1 (0.25p): Implement function $hillclimb(x, max\_steps)$ to find (local) maximum of a function.

Task 2 (0.75p): In class $EightQueens$: implement functions $fitness(x)$, $neighbors(x)$ and $random\_state()$, that will formulate the 8-queens problem as an optimization problem. Your implementation should be able to find a solution (at least once in about 10 runs), sometimes it can be stuck in an local maximum. **In the code clearly state which fitness value corresponds to the solution of the 8-queen problem**.