USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 05
Local and Meta-Heuristic Search in Artificial Intelligence

Objective:

This experiments demonstrates the use of Local Search schemes such as Hill Climbing and use of meta-heuristic based schemes such as Evolutionary and Genetic Algorithms for solution search in Artificial Intelligence.

Name of Student: ______________________________

Roll No: ____________________**Sec.** ______________

Date of Experiment: ______________________________

Marks Obtained/Remarks: ______________

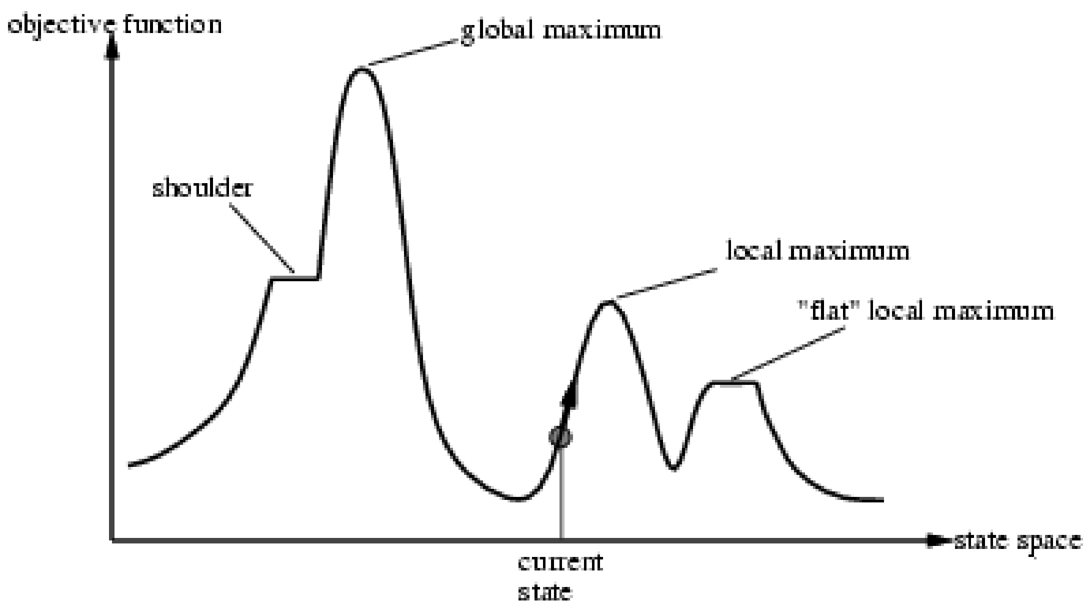
Signature: ______________

Lab 05: Local and Meta-Heuristic Search in Artificial Intelligence

Hill-climbing search

Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence. Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, mathematical optimization problems implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-Travelling salesman problem where we need to minimize the distance traveled by salesman.
- 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in reasonable time.
- The success of hill climbing depends very much on the shape of the state-space landscape: if there are few local maxima and plateau, random-restart hill climbing will find a good solution very quickly. On the other hand, many real problems have a landscape that looks more like a family of porcupines on a flat floor, with miniature porcupines living on the tip of each porcupine needle, ad infinitum.
- NP-hard problems typically have an exponential number of local maxima to get stuck on. Despite this, a reasonably good local maximum can often be found after a small number of restarts.



Algorithm

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

Variants of Hill-Climbing

- **Stochastic hill climbing**
  - Chooses at random from among the uphill moves;
  - The probability of selection can vary with the steepness of the uphill move.
  - This usually converges more slowly than steepest ascent, but in some state landscapes it finds better solutions.

- **First-choice hill climbing**
  - Implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state.
  - This is a good strategy when a state has many (e.g., thousands) of successors.
  - The hill-climbing algorithms described so far fail to find a goal when one exists because they can get stuck on local maxima.

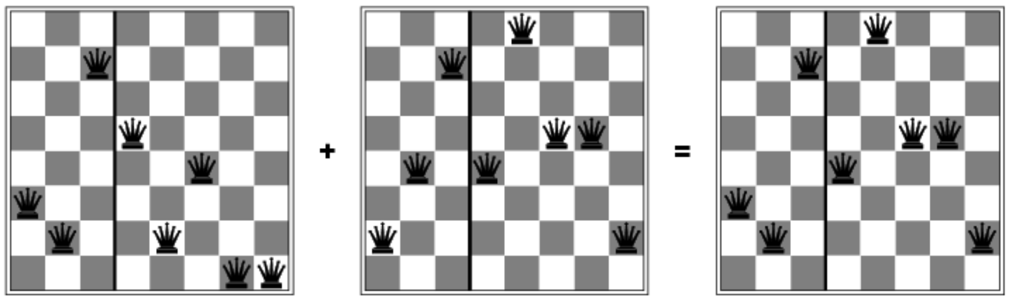
- **Random-restart hill climbing**
  - Adopts the well-known adage, "If at first you don't succeed, try, try again."
  - It conducts a series of hill-climbing searches from randomly generated initial state, stopping when a goal is found.

Genetic Algorithm

The genetic algorithm drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear.

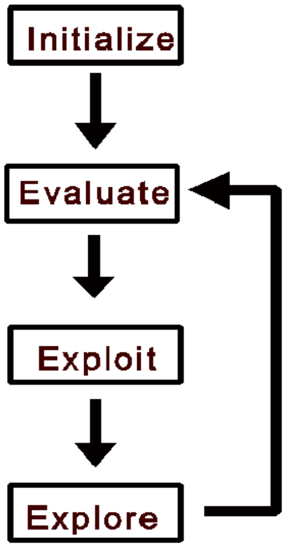
There are some key points to understand genetic algorithm:

- A successor state is generated by combining two parent states.
- Start with k randomly generated states (population).
- A state/individual is represented as a string over a finite alphabet (often a string of 0s and 1s).
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation.



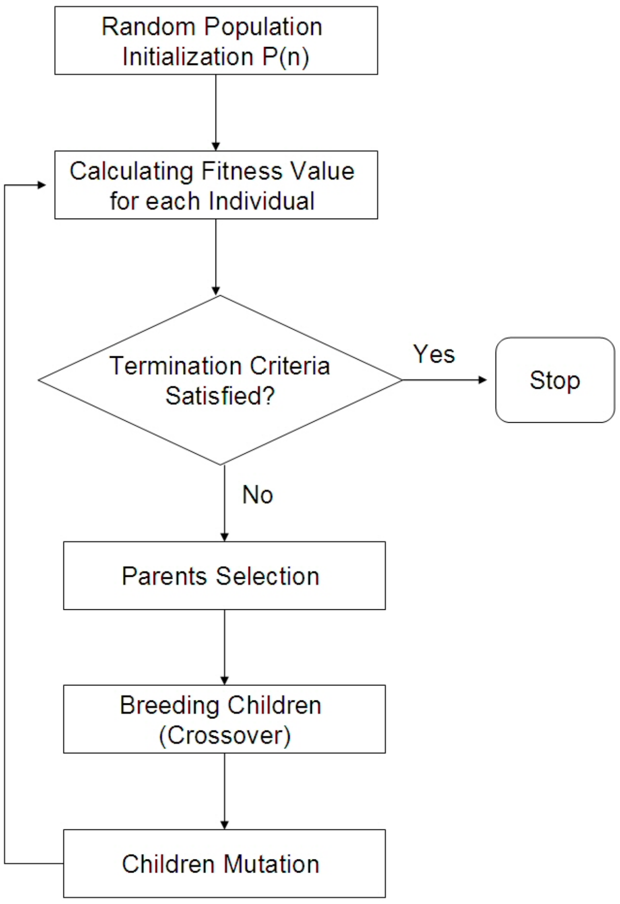
Flowchart of GA

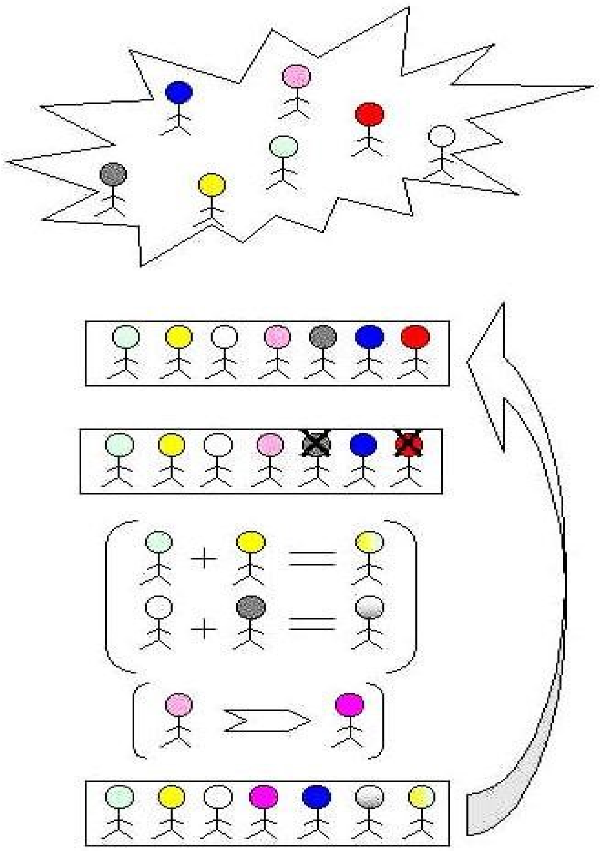
- All individuals in the **population** evaluated by the **fitness function**
- Fitter individuals allowed to **produce child states (selection), crossover, mutate**



Algorithm

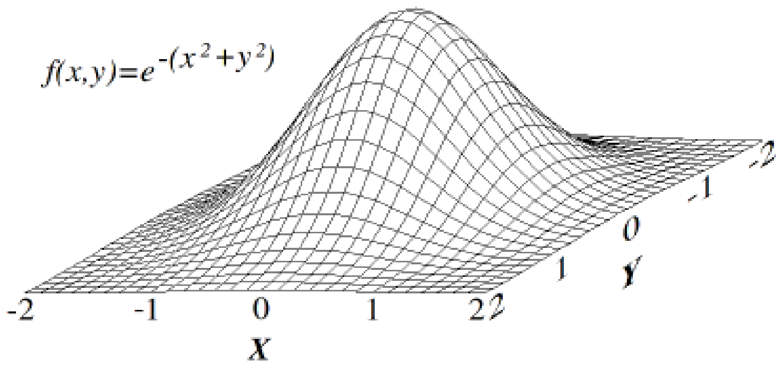
```
    1) Randomly initialize populations p
    2) Determine fitness of population
    3) Until convergence repeat:
        a) Select parents from population
        b) Crossover and generate new population
        c) Perform mutation on new population
        d) Calculate fitness for new population
```

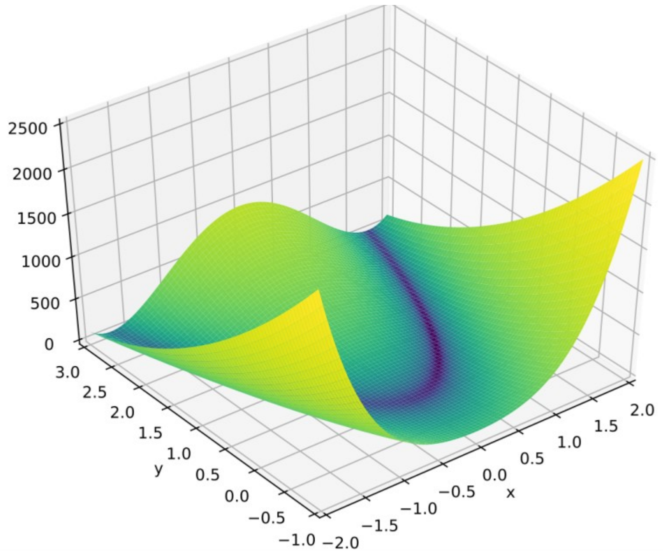

Random Population Initialization P(n)
Calculating Fitness Value for each Individual
Termination Criteria Satisfied?
Yes
Stop
No
Parents Selection
Breeding Children (Crossover)
Children Mutation



Student Exercise

Task 1





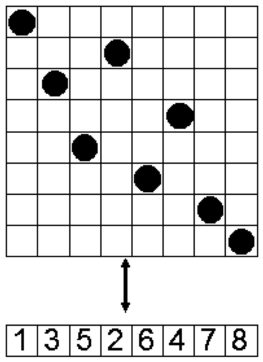
$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

Implement Hill climbing solutions for the given two function. The Hill Climbing solution should provide values for the x and y parameters where the value of the function maximizes. The figures also provide the range of values the solution exist within. Restrict your search within the domain of the variables for a quicker response.

Task2

Implement a Hill climbing program for searching a configuration for the 8 queen problem. Use the representation as suggested below.

- **Penalty of one queen: the number of queens she can check.**
- **Penalty of a configuration: the sum of the penalties of all queens.**
- **Note: penalty is to be minimized**
- **Fitness of a configuration: inverse penalty to be maximized**

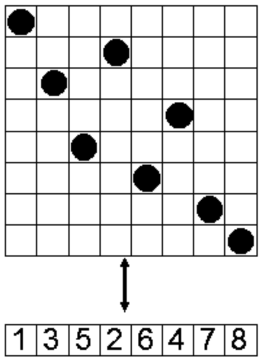


Task 3:

Implement a Genetic Algorithm using Genotypic representation for the two functions used in Task 1. Provide the optimal solution for both the functions using the GA.

Task 4:

Implement an Evolutionary Algorithm for searching a configuration for the 8 queen problem. Use the Phenotypic representation discussed below.



- **Penalty of one queen: the number of queens she can check.**
- **Penalty of a configuration: the sum of the penalties of all queens.**
- **Note: penalty is to be minimized**
- **Fitness of a configuration: inverse penalty to be maximized**