ASSIGNMENT -4
LOCAL SEARCH

ID: 11527196
NAME: ANANYA SAMALA

AIM: In the informed search we use search algorithms for the quick and best solution by using less complexity in time and space.

e.g: We can understand with the example of travelling sales person who travels all the cities in less time in minimum distance.
In the below methods we go through the TSP Travelling sales man problem in detail.

CONTENTS:

1.PROBLEM

psource(Problem)

```
class Problem:
__init__(self, initial, goal=None):
actions(self, state):
result(self, state, action):
 goal_test(self, state):
 path_cost(self, c, state1, action, state2):
 value(self, state):
```

2.NODE

psource(Node)

```
class Node:
__init__(self, state, parent=None, action=None, path_cost=0):

expand(self, problem):
child_node(self, problem, action):
solution(self):
def path(self):
```

These 4 methods override standards Python functionality for representing an object as a string, the less-than (<) operator, the equal-to (=) operator, and the hash function.

```
repr    (self):
   lt    (self, node):
__eq__(self, other):
   hash    (self):
```

*Abstract class Problem to define our real **problem** named GraphProblem. You can see how we define GraphProblem by running the next cell.

psource(GraphProblem)

```
class GraphProblem(Problem):
__init__(self, initial, goal, graph):
actions(self, A):
result(self, state, action):
path_cost(self, cost_so_far, A, action, B):
find_min_edge(self):
h(self, node):
```

TASK 1 - Analysis and visualization for Romania map

3.HILL CLIMBING

Hill Climbing is a heuristic search used for optimization problems. 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 or may not be the global optimum. The algorithm is a variant of generate and test algorithm.
As a whole, the algorithm works as follows:

- Evaluate the initial state.
- If it is equal to the goal state, return.
- Find a neighboring state (one which is heuristically similar to the current state)
- Evaluate this state. If it is closer to the goal state than before, replace the initial state with this state and repeat these steps.

“ If it is goal then quit, of if it is better than current state move to the next state’’

PROBLEMS IN HILL CLIMBING

1. LOCAL MAXIMA:

   If a person blindfolded and moving forward on hill or mountain ,if he reaches a final point on the hill(i.e. local maxima) he doesn’t know in the next place there will be an another hill with highest point (i.e. global maxima)

2. PLATEAU / FLAT SURFACE:

   When the person is moving ahead when all the distances are same hill climbing method is not used.

3. RIDGE:

   If a person is climbing the hill and reaches the highest point next step is to go downwards but there is a way towards right , In hill climbing changing the direction cannot be performed.

   Some of the problems has overcome by using ‘SIMULATION ANNEALING’

4.SIMULATED ANNEALING

The intuition behind Hill Climbing was developed from the metaphor of climbing up the graph of a function to find its peak. There is a fundamental problem in the implementation of the algorithm however. To find the highest hill, we take one step at a time, always uphill, hoping to find the highest point, but if we are unlucky to start from the shoulder of the second-highest hill, there is no way we can find the highest one. The algorithm will always converge to the local optimum. Hill Climbing is also bad at dealing with functions that flatline in certain regions. If all neighboring states have the same value, we cannot find the global optimum using this algorithm.

Compared to hill climbing in simulation when a person is climbing the hill downward steps are also allowed .

If a person blindfolded and moving forward on hill or mountain ,if he reaches a final point on the hill(i.e. local maxima) he doesn't know in the next place there will be an another hill with highest point (i.e. global maxima) in hill climbing

In simulation annealing, the person can come back i.e. on downward steps and can move to the global maxima i.e. second hill.

In Hill-Climbing is about 24 times faster than Simulated Annealing. (Notice that in example we ran Simulated Annealing for 100 iterations whereas we ran Hill Climbing only once.)
Simulated Annealing makes up for its tardiness by its ability to be applicable in a larger number of scenarios than Hill Climbing

5.GENETIC ALGORITHM

Genetic algorithms (or GA) are inspired by natural evolution and are particularly useful in optimization and search problems with large state spaces.

Given a problem, algorithms in the domain make use of a *population* of solutions (also called *states*), where each solution/state represents a feasible solution. At each iteration (often called *generation*), the population gets updated using methods inspired by biology and evolution, like *crossover*, *mutation* and *natural selection*.

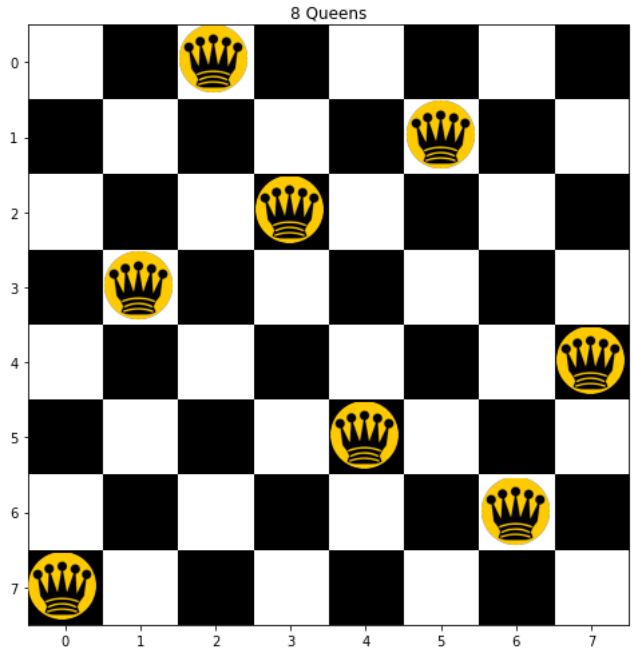
1.Initial population
2.Calculate fitness
3.Selection
4.Crossover
5.Mutation
6.Stoping Criteria
7.Optimal solution

6. N-QUEEN PROBLEM

In N queen problem, we use back tracking problem
Here queen attack in the same rank same file diagonally.
Queen should not be in the same row or same column and should not be diagonally .

"The problem of placing N queens on an NxN board with none attacking each other. A state is represented as an N-element array, where a value of r in the c-th entry means there is a queen at

column c, row r, and a value of -1 means that the c-th column has not been filled in yet. We fill in columns left to right. >>> depth_first_tree_search(NQueensProblem(8)) <Node (7, 3, 0, 2, 5, 1, 6, 4)> """



Simulation Annealing:

Let's now look at an algorithm that can deal with these situations.
Simulated Annealing is quite similar to Hill Climbing, but instead of picking the *best* move every iteration, it picks a *random* move. If this random move brings us closer to the global optimum, it will be accepted, but if it doesn't, the algorithm may accept or reject the move based on a probability dictated by the *temperature*. When the temperature is high, the algorithm is more likely to accept a random move even if it is bad. At low temperatures, only good moves are accepted, with the occasional exception. This allows exploration of the state space and prevents the algorithm from getting stuck at the local optimum.

The temperature is gradually decreased over the course of the iteration. This is done by a scheduling routine. The current implementation uses exponential decay of temperature, but we can use a different scheduling routine instead.
Next, we'll define a peak-finding problem and try to solve it using Simulated Annealing. Let's define the grid and the initial state first.
The peak is at 1.0 which is how gaussian distributions are defined.
This could also be solved by Hill Climbing as follows.

As you can see, Hill-Climbing is about 24 times faster than Simulated Annealing. (Notice that we ran Simulated Annealing for 100 iterations whereas we ran Hill Climbing only once.)
Simulated Annealing makes up for its tardiness by its ability to be applicable in a larger number of scenarios than Hill Climbing as illustrated by the example below.

Notice that even though both algorithms started at the same initial state, Hill Climbing could never escape from the planar region and gave a locally optimum solution of **0**, whereas Simulated Annealing could reach the peak at **32**.
A very similar situation arises when there are two peaks of different heights. One should carefully consider the possible search space before choosing the algorithm for the task.

Generic Algorithm:

A genetic algorithm works in the following way:

1) Initialize random population.

2) Calculate population fitness.

3) Select individuals for mating.

4) Mate selected individuals to produce new population.

* Random chance to mutate individuals.

5) Repeat from step 2) until an individual is fit enough or the maximum number of iterations is reached.

- Individual/State: A list of elements (called *genes*) that represent possible solutions.
- Population: The list of all the individuals/states.
- Gene pool: The alphabet of possible values for an individual's genes.
- Generation/Iteration: The number of times the population will be updated.
- Fitness: An individual's score, calculated by a function specific to the problem.

Crossover

Two individuals/states can "mate" and produce one child. This offspring bears characteristics from both of its parents. There are many ways we can implement this crossover. Here we will take a look at the most common ones. Most other methods are variations of those below.

- Point Crossover: The crossover occurs around one (or more) point. The parents get "split" at the chosen point or points and then get merged. In the example below we see two parents get split and merged at the 3rd digit, producing the following offspring after the crossover.

- Uniform Crossover: This type of crossover chooses randomly the genes to get merged. Here the genes 1, 2 and 5 were chosen from the first parent, so the genes 3, 4 were added by the second parent.

Mutation

When an offspring is produced, there is a chance it will mutate, having one (or more, depending on the implementation) of its genes altered.

For example, let's say the new individual to undergo mutation is "abcde". Randomly we pick to change its third gene to 'z'. The individual now becomes "abzde" and is added to the population.

Selection

At each iteration, the fittest individuals are picked randomly to mate and produce offsprings. We measure an individual's fitness with a *fitness function*. That function depends on the given problem and it is used to score an individual. Usually the higher the better.

The selection process is this:

1) Individuals are scored by the fitness function.

2) Individuals are picked randomly, according to their score (higher score means higher chance to get picked). Usually the formula to calculate the chance to pick an individual is the following (for population *P* and individual *i*):

$$chance(i)=fitness(i)\sum kinPfitness(k)$$

Implementation

The algorithm takes the following input:

- population: The initial population.
- fitness_fn: The problem's fitness function.
- gene_pool: The gene pool of the states/individuals. By default 0 and 1.
- f_thres: The fitness threshold. If an individual reaches that score, iteration stops. By default 'None', which means the algorithm will not halt until the generations are ran.
- ngen: The number of iterations/generations.
- pmut: The probability of mutation.

The algorithm gives as output the state with the largest score.

For each generation, the algorithm updates the population. First it calculates the fitnesses of the individuals, then it selects the most fit ones and finally crosses them over to produce offsprings. There is a chance that the offspring will be mutated, given by pmut. If at the end of the generation an individual meets the fitness threshold, the algorithm halts and returns that individual.

TASK 2: Hill climbing for Santa_barbara and Brest_map

SANTA_BARBARA MAP

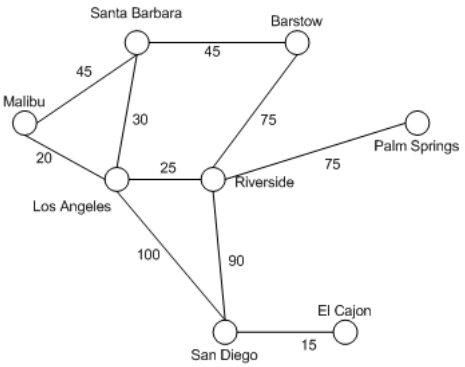
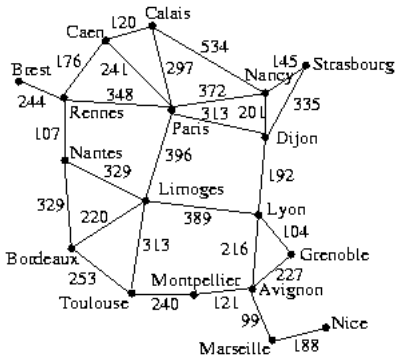


Fig 1: santa_barbara_map

hill_climbing(tsp)
['Barstow',
 'ElCajon',
 'LosAngeles',
 'Malibu',
 'PalmSprings',
 'Riverside',
 'SanDiego',
 'SantaBarbara']

BREST_MAP



['Brest',
 'Montpellier',
 'Dijon',
 'Calais',
 'Bordeaux',
 'Marseille',
 'Nantes',
 'Avignon',
 'Rennes',
 'Strasbourg',
 'Nancy',
 'Nice',
 'Grenoble',
 'Toulouse',
 'Lyon',
 'Paris',
 'Limoges',
 'Caen']
Above are the maps for the routes for brest map and Santabarbara map

Task 3 – Solving TSP using Hill climbing from scratch

1. We took the CVS file upload in google collab
2. Given the code from the scratch i.e pseudo code
3. Imported libraries
4. Took random cities
5. Generated successors by switching two adjacent cities
6. Took the lowest distance cities
7. Took the length of the tour of the best, worst and mean of 10 runs (with random starting tours) both with the 6 first cities, 12 first cities, 18 first cities and with all 24 cities.

a. 6 cities
b. The time for 6 cities, is 0.028624534606933594 seconds.
c. The best route is 5018.8099999999995
d. The worst route is 5135.74
e. The mean distance of the routes is 5030.502999999999
f. The standard deviaton of the routes is 35.07900000000003
g. array([[1., 4., 0., 5., 3., 2.],
h. [4., 0., 1., 3., 2., 5.],
i. [3., 5., 1., 2., 4., 0.],
j. [5., 4., 0., 3., 1., 2.],
k. [5., 4., 2., 3., 0., 1.],
l. [2., 3., 0., 1., 4., 5.],
m. [0., 2., 3., 4., 1., 5.],
n. [0., 2., 3., 5., 1., 4.],
o. [2., 1., 0., 5., 3., 4.],
p. [0., 1., 4., 3., 2., 5.]])

b. 12 cities
The time for 12 cities, is 0.0776054859161377 seconds.
The best route is 10150.67
The worst route is 15034.0
The mean distance of the routes is 13001.927999999998
The standard deviaton of the routes is 1433.4943471796466
array([[5., 0., 10., 2., 11., 9., 4., 6., 7., 8., 3., 1.],
[2., 0., 1., 3., 11., 9., 10., 7., 5., 8., 4., 6.],
[4., 10., 2., 3., 0., 7., 6., 1., 8., 11., 9., 5.],
[11., 4., 8., 7., 0., 3., 10., 2., 9., 1., 5., 6.],
[8., 6., 5., 10., 3., 9., 11., 1., 0., 4., 7., 2.],
[1., 11., 5., 6., 7., 10., 8., 2., 3., 0., 4., 9.],
[10., 8., 7., 11., 9., 5., 2., 4., 1., 0., 3., 6.],
[0., 5., 6., 1., 9., 3., 8., 7., 2., 10., 4., 11.],
[8., 4., 1., 3., 7., 10., 2., 5., 0., 11., 9., 6.],
[3., 7., 6., 10., 4., 1., 2., 11., 5., 0., 9., 8.]])

c. 18 cities

The time for 18 cities, is 0.04636073112487793 seconds.
The best route is 17434.38
The worst route is 23464.16

The mean distance of the routes is 20326.917
The standard deviaton of the routes is 1983.7282353893631
array([[6., 2., 15., 13., 11., 1., 4., 9., 5., 10., 3., 17., 14.,
12., 16., 0., 7., 8.],
[10., 14., 2., 6., 7., 11., 3., 8., 12., 1., 4., 0., 16.,
15., 13., 17., 9., 5.],
[7., 9., 14., 4., 2., 1., 17., 5., 10., 8., 15., 3., 0.,
11., 16., 13., 12., 6.],
[1., 5., 17., 10., 16., 2., 14., 0., 4., 7., 12., 15., 13.,
3., 11., 9., 8., 6.],
[17., 5., 11., 13., 16., 4., 10., 2., 8., 7., 12., 1., 6.,
0., 9., 3., 14., 15.],
[0., 7., 1., 5., 16., 9., 10., 6., 12., 11., 8., 17., 14.,
4., 15., 2., 13., 3.],
[5., 14., 6., 13., 1., 15., 8., 16., 10., 11., 0., 4., 3.,
2., 9., 17., 12., 7.],
[4., 14., 10., 3., 16., 6., 7., 9., 2., 17., 8., 12., 11.,
5., 13., 0., 15., 1.],
[9., 12., 14., 17., 10., 3., 6., 13., 1., 2., 16., 4., 11.,
0., 7., 5., 8., 15.],
[9., 14., 8., 2., 17., 6., 11., 10., 4., 3., 16., 7., 15.,
0., 1., 5., 12., 13.]])

d. <u>24 cities</u>

The time for 24 cities, is 0.2030317783355713 seconds.
The best route is 22282.230000000003
The worst route is 32838.369999999995
The mean distance of the routes is 28092.737
The standard deviaton of the routes is 2894.69133060176
array([[12., 13., 23., 5., 14., 17., 11., 2., 8., 15., 18., 16., 1.,
20., 10., 19., 6., 4., 21., 7., 0., 9., 22., 3.],
[0., 1., 15., 19., 12., 21., 9., 3., 22., 17., 16., 6., 18.,
23., 10., 5., 14., 7., 13., 2., 20., 4., 11., 8.],
[6., 20., 10., 7., 18., 8., 13., 17., 3., 2., 12., 15., 11.,
4., 0., 14., 16., 9., 19., 22., 21., 23., 1., 5.],
[11., 4., 1., 6., 17., 0., 18., 23., 22., 19., 8., 9., 20.,
21., 14., 3., 16., 15., 12., 2., 13., 7., 10., 5.],
[5., 0., 14., 6., 15., 23., 22., 7., 1., 18., 17., 2., 4.,
21., 11., 3., 8., 16., 20., 13., 12., 19., 10., 9.],
[0., 17., 19., 10., 23., 4., 1., 22., 20., 21., 8., 3., 12.,
9., 5., 18., 13., 16., 15., 7., 14., 6., 11., 2.],
[10., 4., 13., 9., 14., 8., 2., 11., 5., 22., 3., 15., 18.,
16., 12., 6., 1., 23., 21., 19., 17., 20., 0., 7.],
[21., 7., 8., 5., 18., 4., 22., 23., 15., 20., 11., 1., 12.,
19., 10., 0., 6., 16., 17., 2., 3., 9., 14., 13.],
[10., 23., 5., 22., 18., 13., 12., 0., 11., 6., 15., 8., 20.,
9., 1., 17., 7., 2., 3., 16., 21., 14., 4., 19.],
[16., 1., 21., 6., 11., 19., 14., 8., 20., 13., 10., 18., 3.,
15., 12., 0., 5., 7., 9., 22., 4., 2., 17., 23.]])

We observe for minimum cities the distance and mean is less .

XXX