AI Course

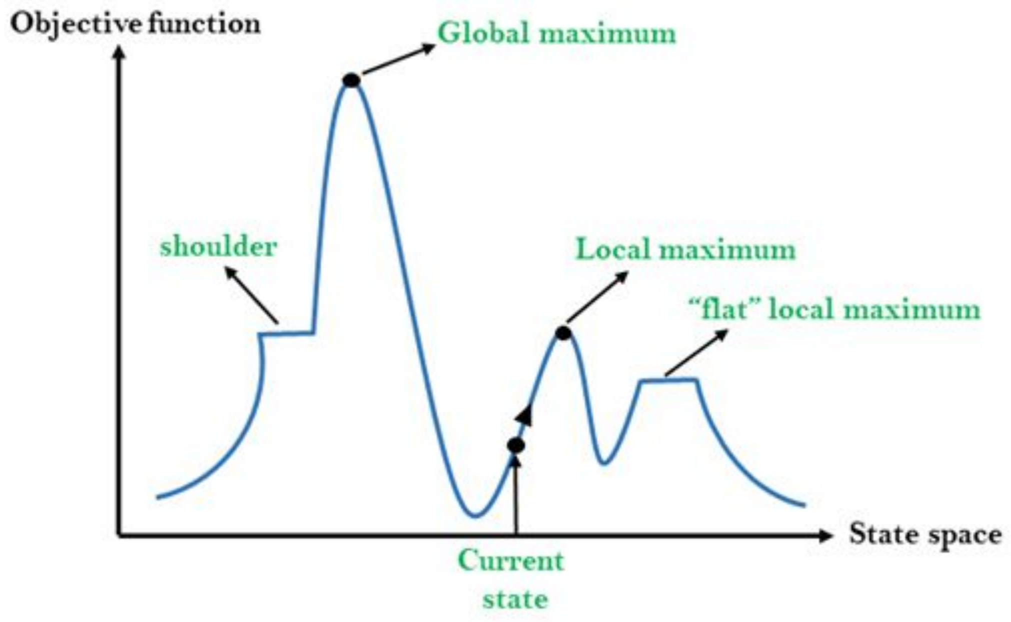
Dr. Mürsel Taşgın

Hill Climbing, Simulated Annealing

Hill Climbing- *Introduction*

Greedy local search

- Generate nearby successor states to the current state
- Pick the best and replace the current state with that one.
- Loop



Hill Climbing- *Introduction*

Types of Hill Climbing Algorithm:

- Simple hill climbing
- Steepest-Ascent hill-climbing
- Stochastic hill climbing

Hill Climbing- *1-Simple Hill Climbing*

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state.

This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
  - If it is goal state, then return success and quit.
  - Else if it is better than the current state then assign new state as a current state.
  - Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

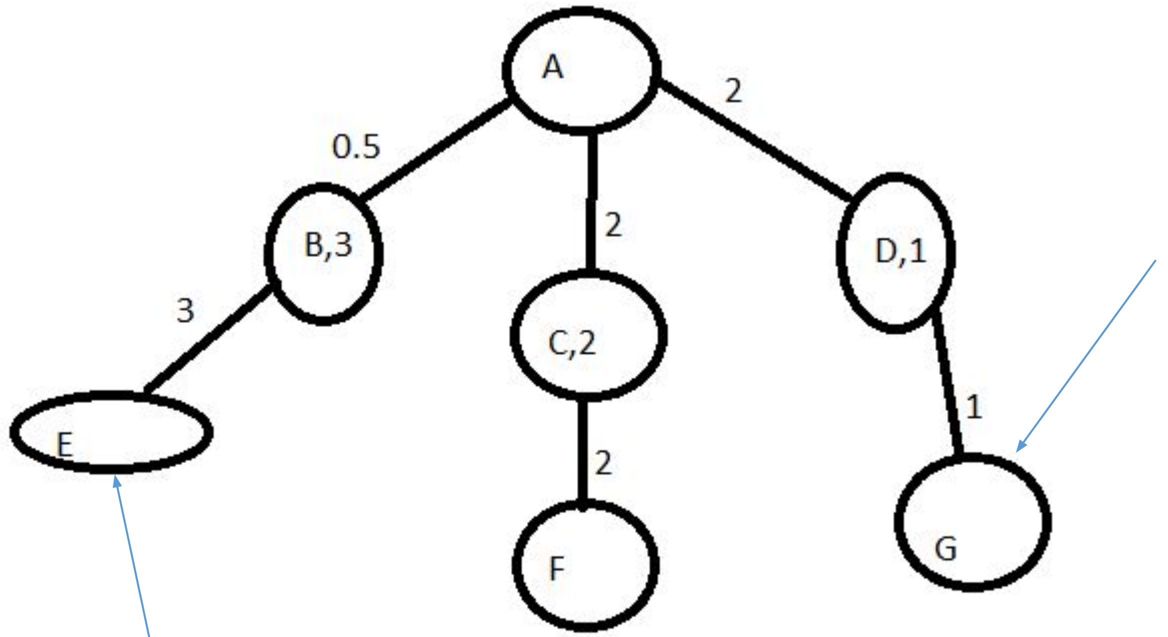
Hill Climbing- *2-Steepest-Ascent Hill Climbing*

The steepest-sscent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    - Let SUCC be a state such that any successor of the current state will be better than it.
    - For each operator that applies to the current state:
        - Apply the new operator and generate a new state.
        - Evaluate the new state.
        - If it is goal state, then return it and quit, else compare it to the SUCC.
        - If it is better than SUCC, then set new state as SUCC.
        - If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

Hill Climbing- *Simple vs Steepest-Ascent*



Steepest-Ascent hill-climbing will take path to D and then G as it will have minimum overall cost

Simple hill-climbing will choose B first and ends up to E. Total cost is not optimum

Hill Climbing- *3-Stochastic hill climbing*

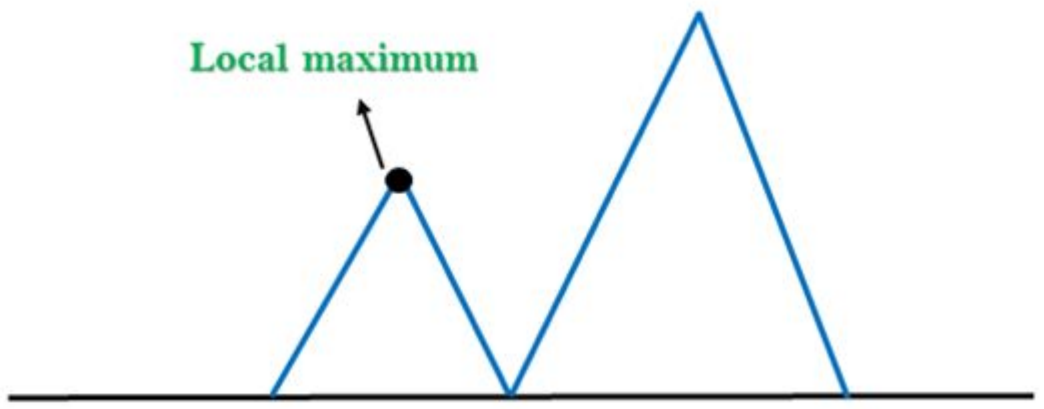
Stochastic hill climbing does not examine for all its neighbor before moving.

Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Hill Climbing- *Problems*

Local maximum:

A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

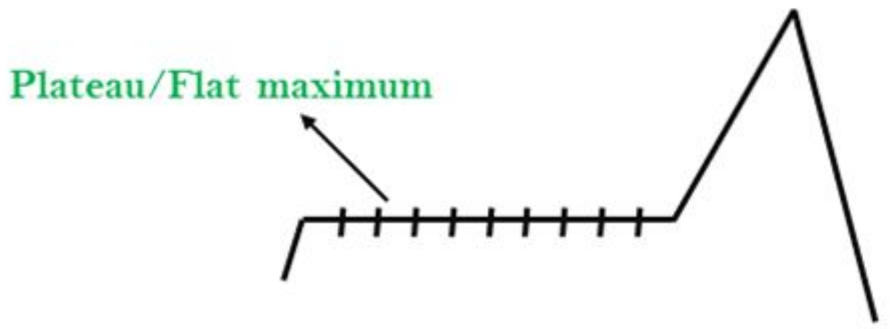


Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.

Hill Climbing- *Problems*

Plateau:

A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

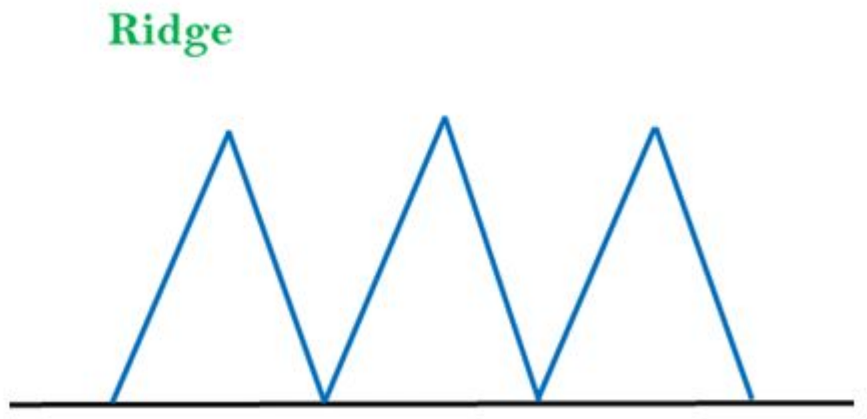


Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.

Hill Climbing- *Problems*

Ridges:

A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.



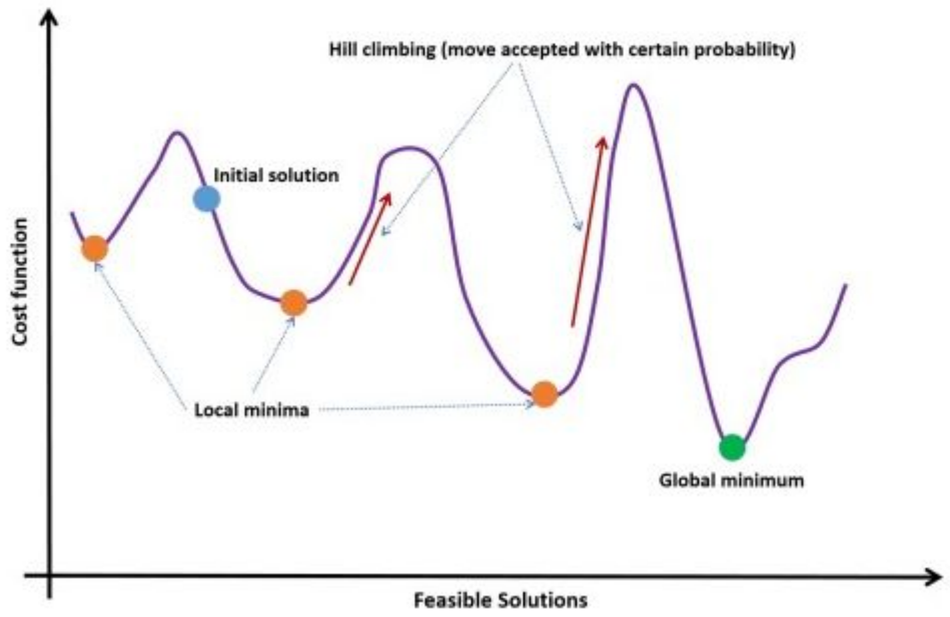
Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Simulated Annealing - *Introduction*

- Simulated Annealing (SA) is a probabilistic search optimization technique proposed in 1983 by Kirkpatrick et al. [93] and in 1985 by Černý [213].

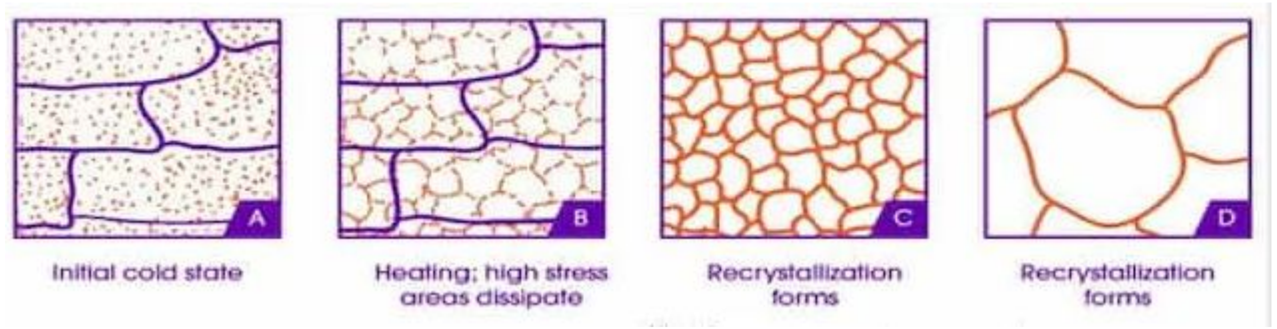
- Its main idea is to find a global minimum of a specific objective function attempting to escape local minima in the search process. Due to low complexity, it can be used in many various optimization problems, not only related to the VRPs.

- This method takes its name from the annealing in metallurgy, involving heating a solid material to a certain temperature at which it becomes liquid, followed by slow and controlled cooling down until the solid state is reached again and the metal particles are rearranged in a minimal energy molecular structure.



Simulated Annealing - *Introduction*

- A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient.
- **Simulated Annealing** is an algorithm which yields both efficiency and completeness.
- In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state.
- The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.



Metal annealing process

Simulated Annealing - *Introduction*

- The system heat is high in the beginning and decrease with iterations.

  temperature = initial_temperature / (iteration_number + 1)

- In high temperature, new points that are not as good as the current point (worse points) are accepted sometimes.
- A worse point is accepted probabilistically where the likelihood of accepting a solution worse than the current solution is a function of the temperature of the search and how much worse the solution is than the current solution. Acception criterion:

  criterion = exp(-(objective(new) – objective(current)) / temperature)

- The effect is that poor solutions have more chances of being accepted early in the search and less likely of being accepted later in the search. The intent is that the high temperature at the beginning of the search will help the search locate the basin for the global optima and the low temperature later in the search will help the algorithm hone in on the global optima.

Local search – *Simulated annealing*

https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python/

```
# simulated annealing search of a one-dimensional objective function
from numpy import asarray
from numpy import exp
from numpy.random import randn
from numpy.random import rand
from numpy.random import seed

# objective function
def objective(x):
	return x[0]**2.0

# simulated annealing algorithm
def simulated_annealing(objective, bounds, n_iterations, step_size, temp):
	# generate an initial point
	best = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
	# evaluate the initial point
	best_eval = objective(best)
	# current working solution
	curr, curr_eval = best, best_eval
	# run the algorithm
	for i in range(n_iterations):
		# take a step
		candidate = curr + randn(len(bounds)) * step_size
		# evaluate candidate point
		candidate_eval = objective(candidate)
		# check for new best solution
		if candidate_eval < best_eval:
			# store new best point
			best, best_eval = candidate, candidate_eval
			# report progress
			print('>%d f(%s) = %.5f' % (i, best, best_eval))
		# difference between candidate and current point evaluation
		diff = candidate_eval - curr_eval
		# calculate temperature for current epoch
		t = temp / float(i + 1)
		# calculate metropolis acceptance criterion
		metropolis = exp(-diff / t)
		# check if we should keep the new point
		if diff < 0 or rand() < metropolis:
			# store the new current point
			curr, curr_eval = candidate, candidate_eval
	return [best, best_eval]
```

```
# seed the pseudorandom number generator
seed(1)
# define range for input
bounds = asarray([[-5.0, 5.0]])
# define the total iterations
n_iterations = 1000
# define the maximum step size
step_size = 0.1
# initial temperature
temp = 10
# perform the simulated annealing search
best, score = simulated_annealing(objective, bounds, n_iterations, step_size, temp)
print('Done!')
print('f(%s) = %f' % (best, score))
```

Local Search – *Simulated annealing - TSP*

https://github.com/perrygeo/simanneal/tree/master/tests