COL333/671: Introduction to AI

Semester I, 2021

Local Search Algorithms

Rohan Paul

Outline

- Last Class
  - Constraint Satisfaction Problems
- This Class
  - Local Search Algorithms
- Reference Material
  - AIMA Ch. 4.1

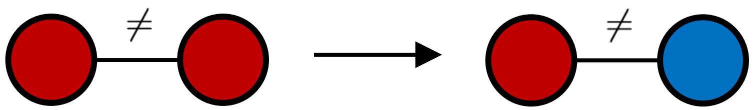
Acknowledgement

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Dan Klein, Nicholas Roy and others.

Iterative Approaches to Solving CSPs

- Local search methods
  - Keep track of “complete” states, i.e., all variables assigned.
  - If there are conflicts, then try to improve the assignment iteratively till a solution is obtained.

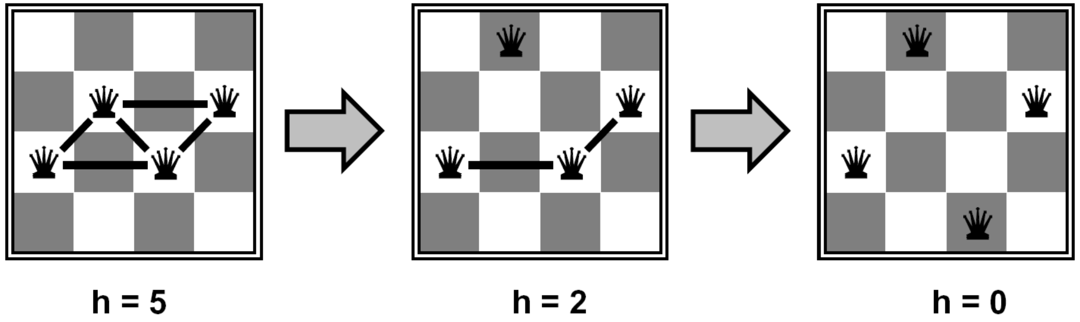
- Local search for CSPs:
  - Take an assignment with unsatisfied constraints
  - Reassign variable values
  - Repeat: Till the CSP does not have a solution,
    - Variable selection: randomly select any conflicted variable
    - Value selection: min-conflicts heuristic:
      - Choose a value for the variable that violates the fewest constraints
      - I.e., “hill climb” with h(x) = total number of violated constraints
  - Hill climbing (a general technique for search)
    - Start at a state
    - Repeat: move to the best neighboring state
    - If no neighbors better than current, quit



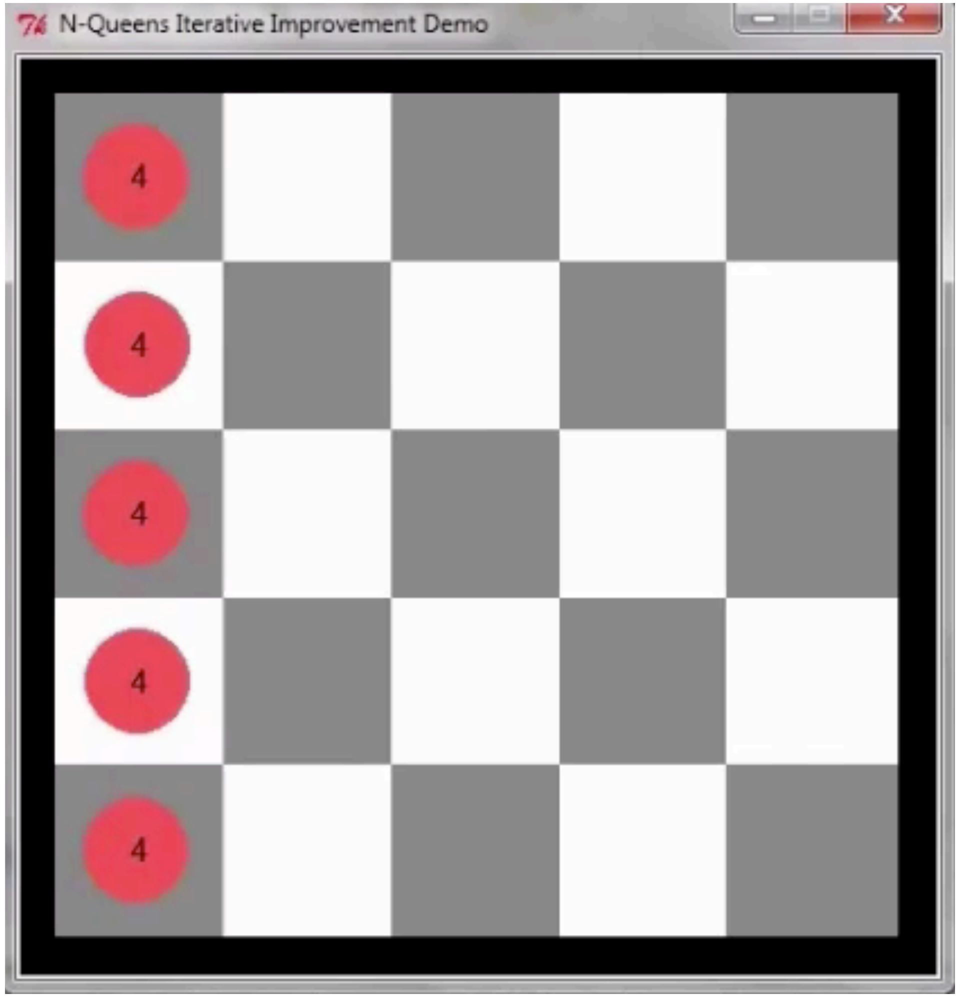
Keep track of complete assignments and explore at local changes.

Hill climbing

Example: 4 Queens

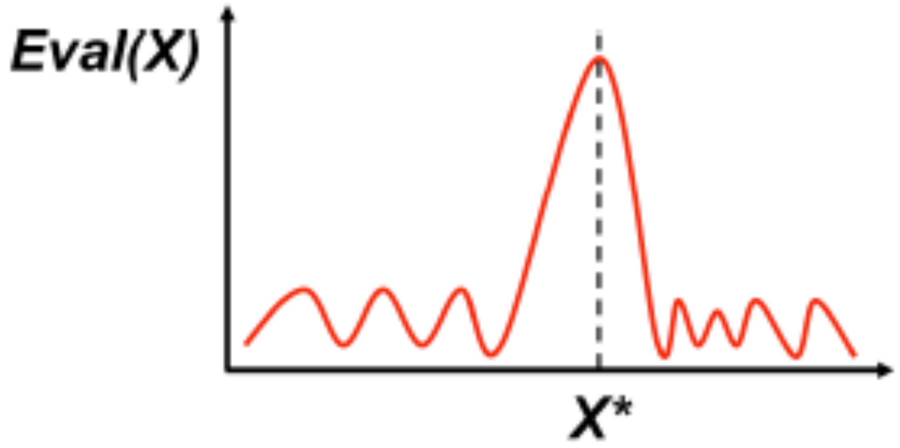


- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(x) = number of attacks (number of violated binary constraints)

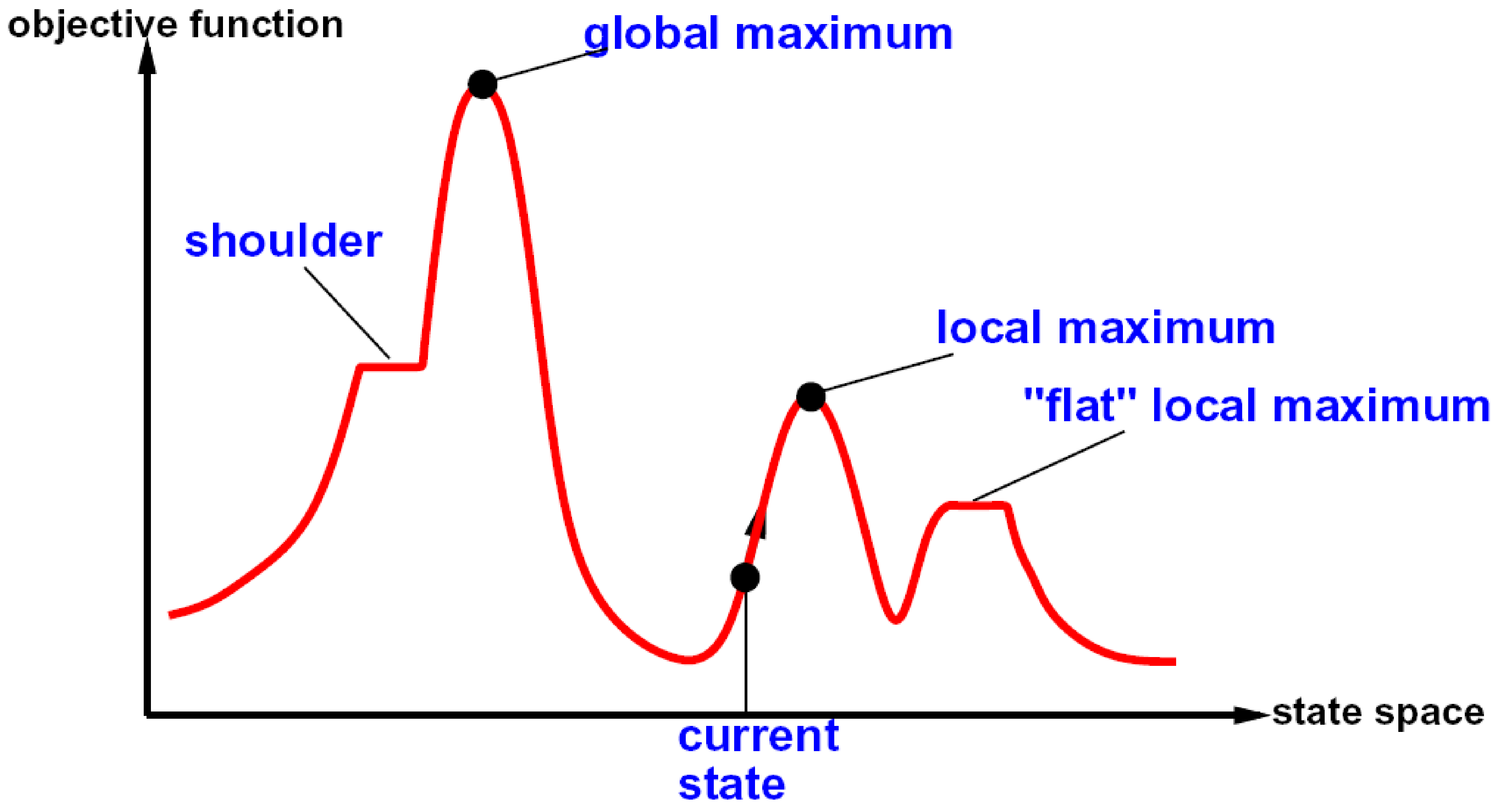


Optimization Problems: Generic Setup

- There is (discrete) combinatorial structure to the problem (maybe with constraints).
- Cost function, which we want to optimize.
- Searching all possible solutions is likely to be infeasible.
  - At least, we want a “good” solution.



Optimization Landscape



Simulated Annealing

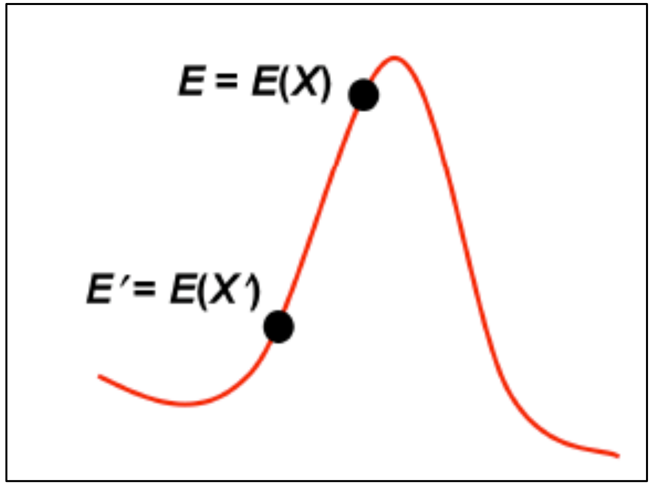
- Allows some apparently **bad moves** - to escape local maxima.
- **Decrease** the size and the frequency of bad moves over time.

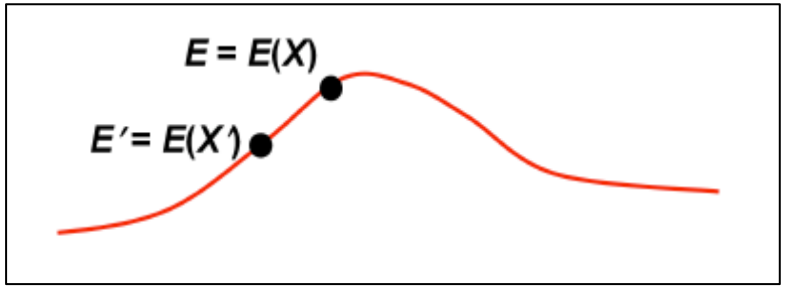
- Algorithm sketch
  1. Start at initial configuration X of value E (high is good)
  2. Repeat:
     (a) Let X_i be a *random neighbor* of X and E_i be its value
     (b) If $E < E_i$ then let $X \leftarrow X_i$ and $E \leftarrow E_i$
     (c) Else, *with some probability* p, still accept the move: $X \leftarrow X_i$ and $E \leftarrow E_i$
- Best solution ever found is always remembered

A form of Monte-Carlo Search. Move around the environment to explore it instead of systematically sweeping. Powerful technique for large domains.

Simulated Annealing: How to decide *p*?

- Considering a move from state of value E to a lower valued state of E'.
- If (E – E') is large:
  - Likely to be close to a promising maximum.
  - Less inclined to to go downhill.
- If (E – E') is small:
  - The closest maximum may be shallow
  - More inclined to go downhill is not as bad.





Simulated Annealing: Selecting Moves

- If the new value E_i is ***better*** than the old value E, move to X_i

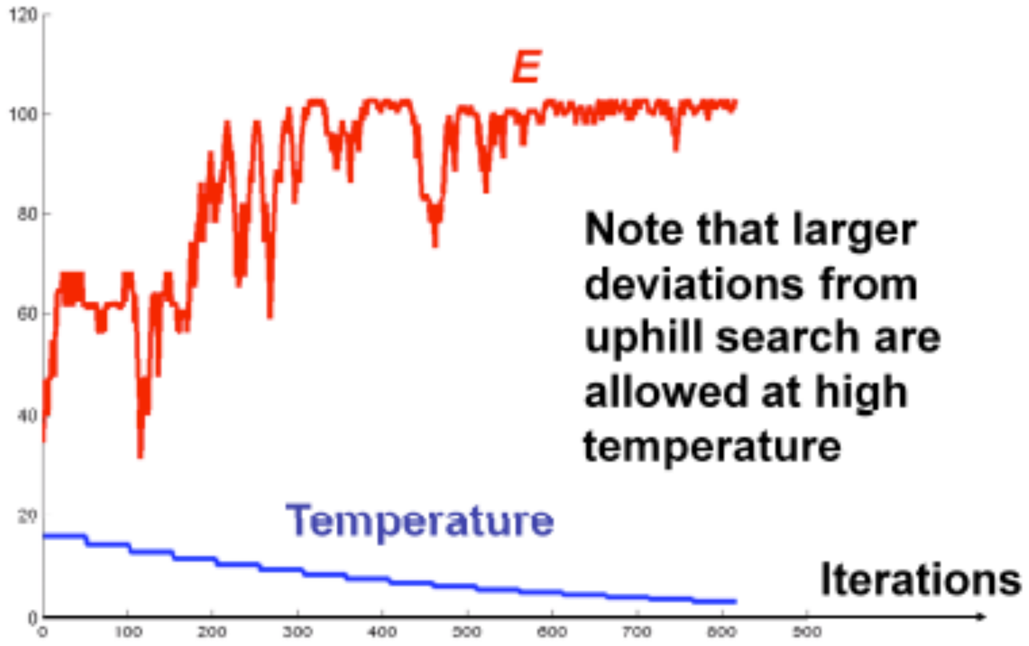
- If the new value is ***worse*** ($E_i < E$) then move to the neighboring solution as per *Boltzmann* distribution.

$$\exp\left(-\frac{E - E_i}{T}\right)$$

- Temperature (T>0)
  - **T is high**, exp is ~0, acceptance probability is ~1, high probability of acceptance of a worse solution.
  - **T is low**, the probability of moving to a worse solution is ~ 0, low probability of acceptance of a worse solution.
  - Schedule T to reduce over time.

Simulated Annealing: Properties

- **T is high**
  - The algorithm is in an exploratory phase
  - Even bad moves have a high chance of being picked)
- **T is low**
  - The algorithm is in an exploitation phase
  - The “bad” moves have very low probability
- **If T is decreased slowly enough**
  - Simulated annealing is guaranteed to reach the best solution in the limit.



Local Beam Search

- Look for solutions from multiple points in parallel.
- Algorithm
  - Track k states (rather than 1).
  - Begin with k randomly sampled states.
  - Loop
    - Generate successors of each of the k-states
    - If anyone has the goal, the algorithm halts
    - Otherwise, select only the k-best successors from the list and repeat.
  - Note:
    - Each run is <u>not</u> independent, information is passed between parallel search threads.
    - Promising states are propagated. Less promising states are not propagated.
    - Problem: states become concentrated in a small region of space.

Stochastic Beam Search

- Local beam search
  - Problem: states become concentrated in a small region of space
  - Search degenerates to hill climbing
- Stochastic beam search
  - Instead of taking the best k states
  - Sample k states from a distribution
  - Probability of selecting a state *increases* as the *value* of the state.