CS 1571 Introduction to AI
Lecture 9

Finding optimal configurations

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Search for the optimal configuration

Objective:
- **find the optimal configuration**

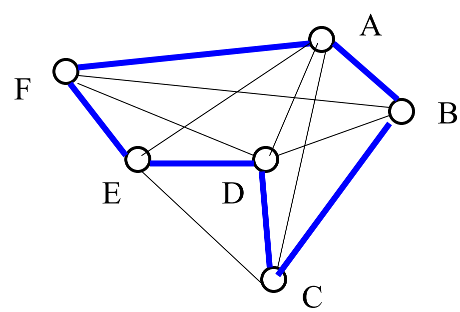
Optimality:
- Defined by some **quality measure**

Quality measure
- reflects our **preference towards each configuration** (or state)

Example: Traveling salesman problem

Problem:

- A graph with distances



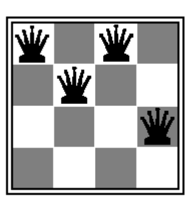
- **Goal:** find the shortest tour which visits every city once and returns to the start

  An example of a valid **tour:** ABCDEF

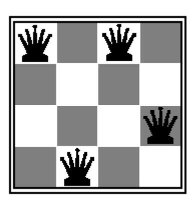
Example: N queens

- A CSP problem can be converted to the 'optimal' configuration problem
- **The quality of a configuration in a CSP**

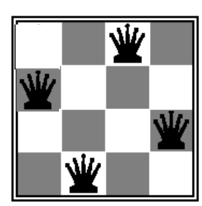
  = the number of constraints violated
- **Solving:** minimize the number of constraint violations



# of violations =3



# of violations =1



# of violations =0

Iterative optimization methods

- Searching systematically for the best configuration with the **DFS** may not be the best solution
- Worst case running time:
  - Exponential in the number of variables
- Solutions to **large 'optimal' configuration** problems are often found using iterative optimization methods

- **Methods:**
  - **Hill climbing**
  - **Simulated Annealing**
  - **Genetic algorithms**

Iterative optimization methods

Properties:

- **Search the space of "complete" configurations**
- **Take advantage of local moves**
  - Operators make "local" changes to "complete" configurations
- **Keep track of just one state (the current state)**
  - no memory of past states
  - **!!! No search tree is necessary !!!**

Example: N-queens

- **“Local” operators for generating the next state:**
  – Select a variable (a queen)
  – Reallocate its position

Example: Traveling salesman problem

“Local” operator for generating the next state:

- divide the existing tour into two parts,
- reconnect the two parts in the opposite order

Example:

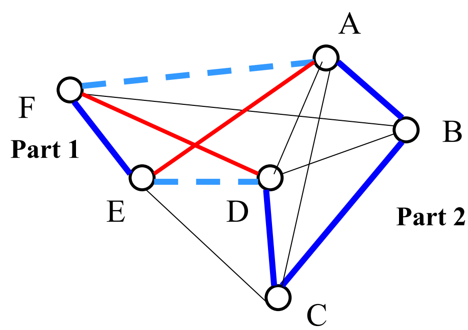
ABCDEF

⇩

ABCD | EF |

⇩

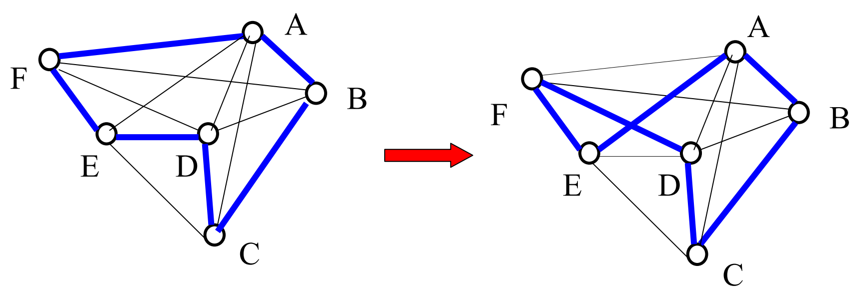
ABCDFE



Example: Traveling salesman problem

"Local" operator:

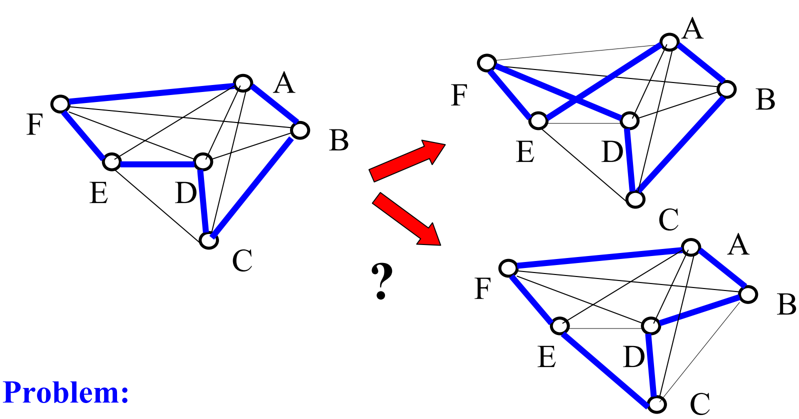
– generates the next configuration (state)



Searching the configuration space

Search algorithms

- keep only one configuration (the current configuration)



Problem:

- How to decide about which operator to apply?

Search algorithms

Two strategies to choose the configuration (state) to be visited next:

- **Hill climbing**
- **Simulated annealing**

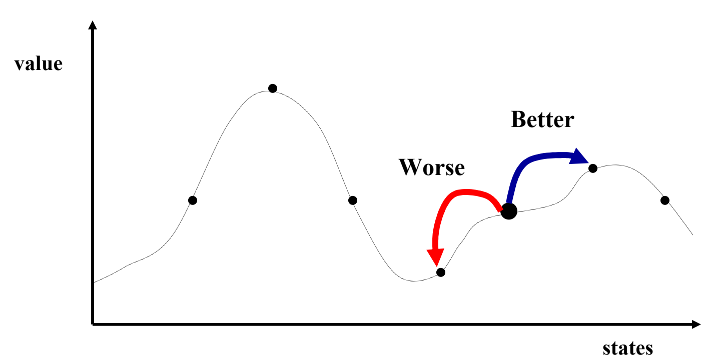
- Later: Extensions to multiple current states:
  - **Genetic algorithms**

- **Note:** Maximization is inverse of the minimization

$$\min\ f(X) \Leftrightarrow \max\left[-f(X)\right]$$

Hill climbing

- Look around at states in the local neighborhood and choose the one with the best value
- Assume: we want to maximize the



Hill climbing

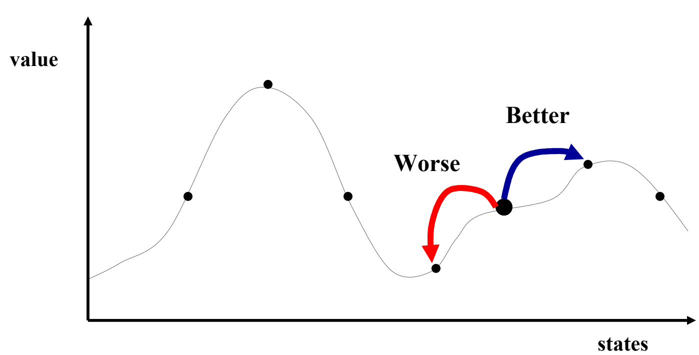
- Always choose the next best successor state
- Stop when no improvement possible

```
function HILL-CLIMBING(problem) returns a solution state
   inputs: problem, a problem
   static: current, a node
           next, a node

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


Hill climbing

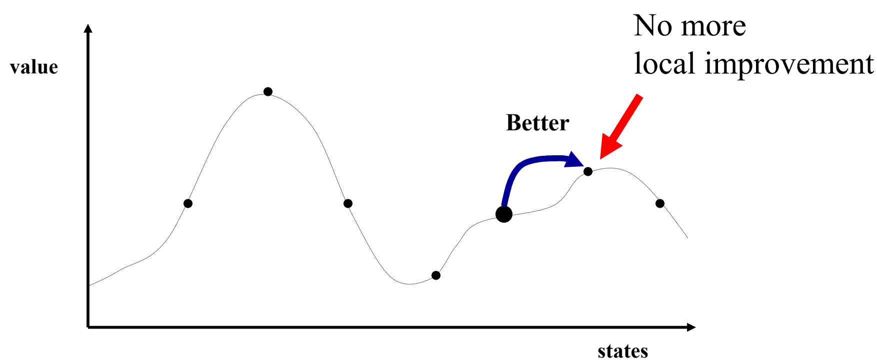
- Look around at states in the local neighborhood and choose the one with the best value



- **What can go wrong?**

Hill climbing

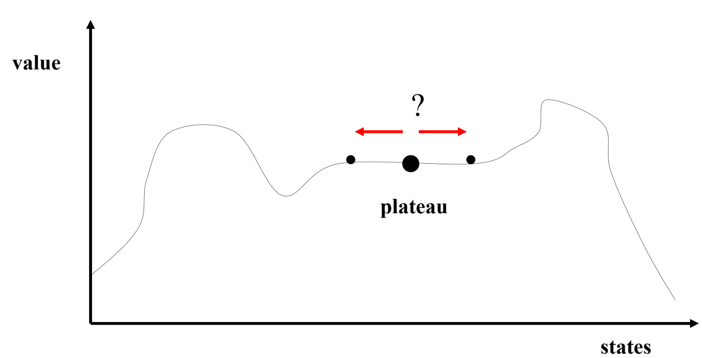
- Hill climbing can get trapped in the local optimum



- **What can go wrong?**

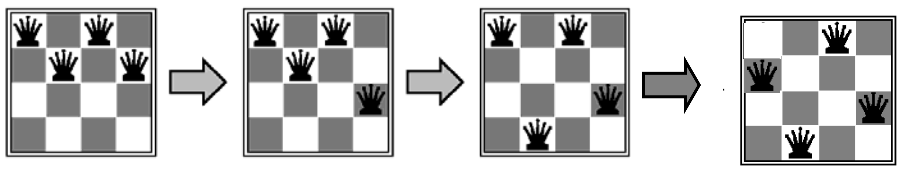
Hill climbing

- Hill climbing can get clueless on plateaus



Hill climbing and n-queens

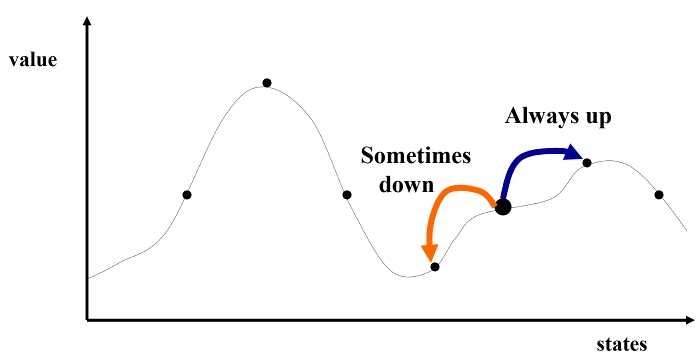
- The quality of a configuration is given by the number of constraints violated
- **Then: Hill climbing** reduces the number of constraints
- **Min-conflict strategy (heuristic):**
  - Choose randomly a variable with conflicts
  - Choose its value such that it violates the fewest constraints



Success !! But not always!!! The local optima problem!!!

Simulated annealing

- Permits "bad" moves to states with lower value, thus escape the local optima
- **Gradually decreases** the frequency of such moves and their size (parameter controlling it – **temperature**)



Simulated annealing algorithm

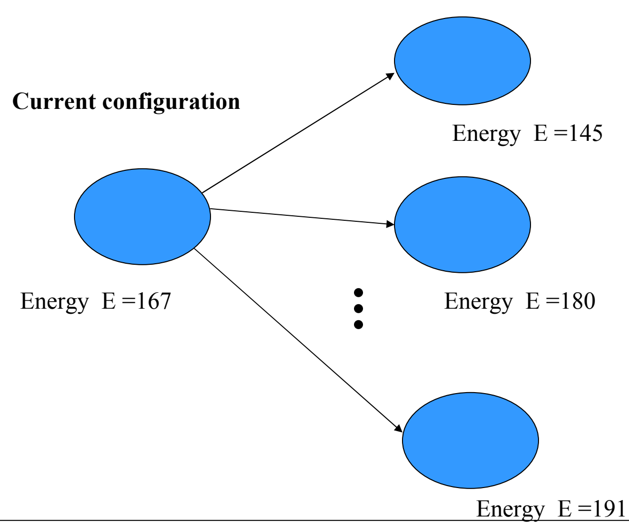
The probability of making a move:

- The probability of moving into a state with a higher value is 1
- The probability of moving into a state with a lower value is

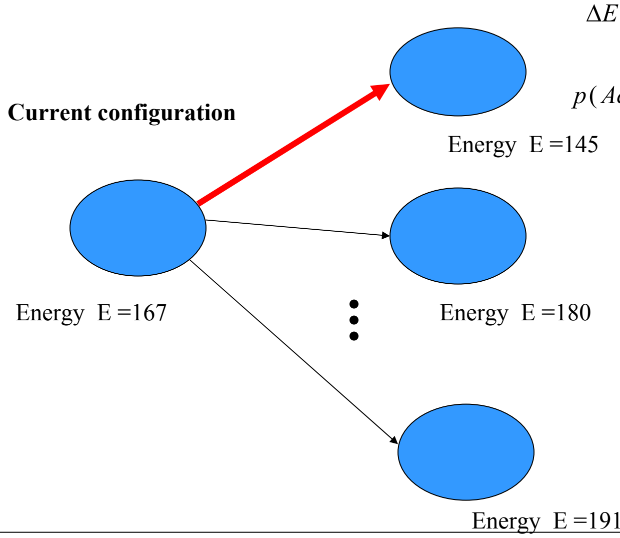
$$p(Accept\ NEXT) = e^{\Delta E / T} \quad \text{where} \quad \Delta E = E_{NEXT} - E_{CURRENT}$$

  - The probability is:
    - **Proportional to the energy difference**

Simulated annealing algorithm



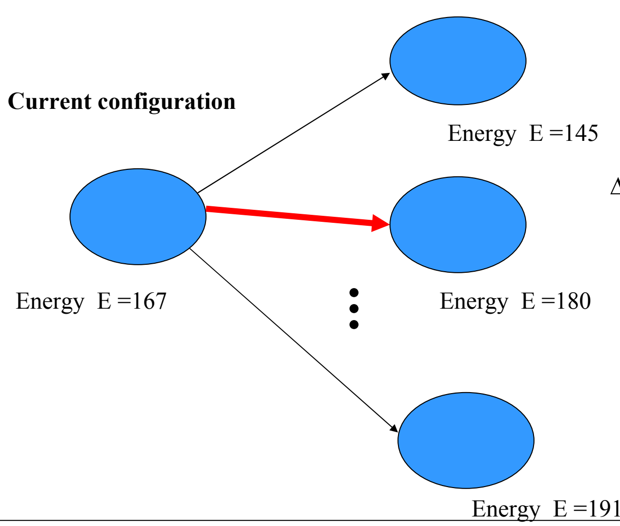
Simulated annealing algorithm



$$\Delta E = E_{NEXT} - E_{CURRENT}$$
$$= 145 - 167 = -22$$
$$p(Accept) = e^{\Delta E / T} = e^{-22/T}$$

Sometimes accept!

Simulated annealing algorithm



$$\Delta E = E_{NEXT} - E_{CURRENT}$$
$$= 180 - 167 > 0$$
$$p(Accept) = 1$$

Always accept!

Simulated annealing algorithm

The probability of moving into a state with a lower value is

$$p(Accept) = e^{\Delta E / T} \quad \text{where} \quad \Delta E = E_{NEXT} - E_{CURRENT}$$

The probability is:

- **Modulated through a temperature parameter T:**
  - for $T \to \infty$ the probability of any move approaches 1
  - for $T \to 0$ the probability that a state with smaller value is selected goes down and approaches 0
- **Cooling schedule:**
  - Schedule of changes of a parameter T over iteration steps

Simulated annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  static: current, a node
          next, a node
          T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
     T ← schedule[t]
     if T=0 then return current
     next ← a randomly selected successor of current
     ΔE ← VALUE[next] – VALUE[current]
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

Simulated annealing algorithm

- **Simulated annealing algorithm**
  - developed originally for modeling physical processes (Metropolis et al, 53)
- **Properties:**
  - **If T is decreased slowly enough the best configuration (state) is always reached**
- **Applications:**
  - VLSI design
  - airline scheduling

Simulated evolution and genetic algorithms

- Limitations of **simulated annealing**:
  - Pursues one state configuration at the time;
  - Changes to configurations are typically local

Can we do better?

- Assume we have two configurations with good values that are quite different
- We expect that the combination of the two individual configurations may lead to a configuration with higher value

(**Not guaranteed !!!)**

This is the idea behind **genetic algorithms** in which we grow a population of individual combinations

Genetic algorithms

Algorithm idea:

- **Create a population of random configurations**
- **Create a new population through:**
  - Biased selection of pairs of configurations from the previous population
  - Crossover (combination) of pairs
  - Mutation of resulting individuals
- **Evolve the population over multiple generation cycles**

- **Selection of configurations to be combined:**
  - **Fitness function = value function**

    measures the quality of an individual (a state) in the population

Reproduction process in GA

- Assume that a state configuration is defined by a set variables with two values, represented as 0 or 1