CS 1571 Introduction to AI
Lecture 9

Finding optimal configurations

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

Announcements

- **Homework assignment 2 due today**
- **Homework assignment 3 is out**
  – Programming and experiments
  – Simulated annealing + Genetic algorithm
  – Competition

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs1571/

Search for the optimal configuration

Constrain satisfaction problem:

Objective: **find a configuration that satisfies all constraints**

Optimal configuration problem:

Objective: **find the best configuration**

The quality of a configuration: is defined by some quality measure that reflects our **preference towards each configuration** (or state)

Search for the optimal configuration

Optimal configuration search:

- Configurations are described in terms of variables and their values
- Each configuration has a quality measure
- Goal: find the configuration with the best value

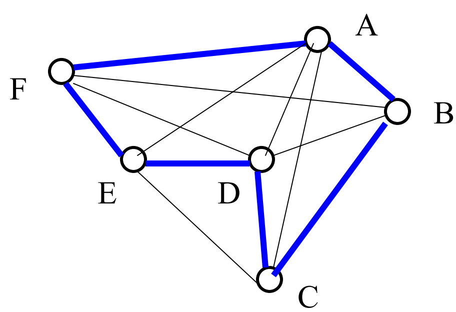
If the space of configurations we search among is

- **Discrete or finite**
  - **then it is a combinatorial optimization problem**
- **Continuous**
  - **then it is a parametric optimization problem**

Example: Traveling salesman problem

Problem:

- A graph with distances

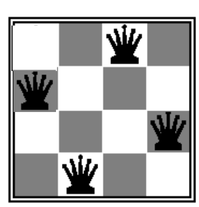


- **Goal:** find the shortest tour which visits every city once and returns to the start

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

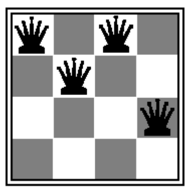
Example: N queens

- A CSP problem
- Is it possible to formulate the problem as an optimal configuration search problem ?

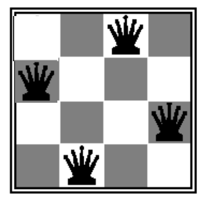


Example: N queens

- A CSP problem
- Is it possible to formulate the problem as an optimal configuration search problem ? **Yes.**
- **Constraints are mapped to the objective cost function that** counts the number of violated constraints



# of violations =3



# of violations =0

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:

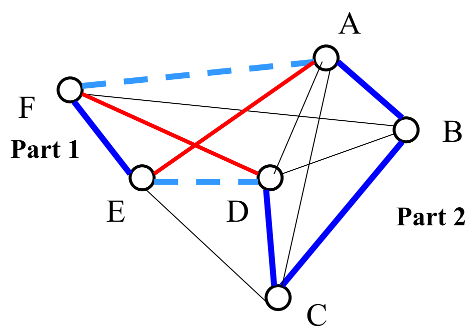
ABCDEF

⇩

ABCD | EF |

⇩

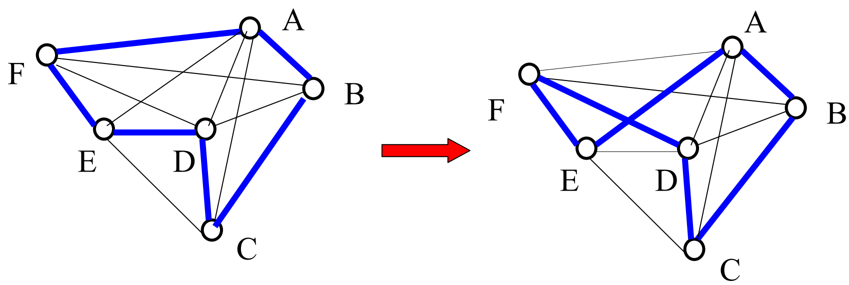
ABCDFE



Example: Traveling salesman problem

"Local" operator:

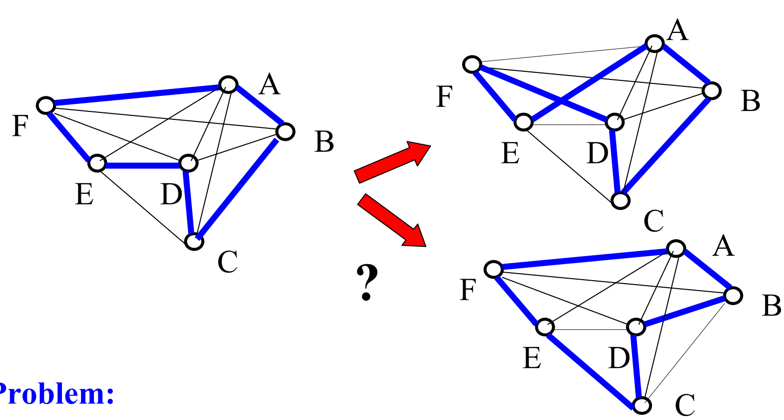
– generates the next configuration (state)



Searching the configuration space

Search algorithms

- keep only one configuration (the current configuration)



Problem:

- How to decide about which operator to apply?

Search algorithms

Strategies to choose the configuration (state) to be visited next:

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

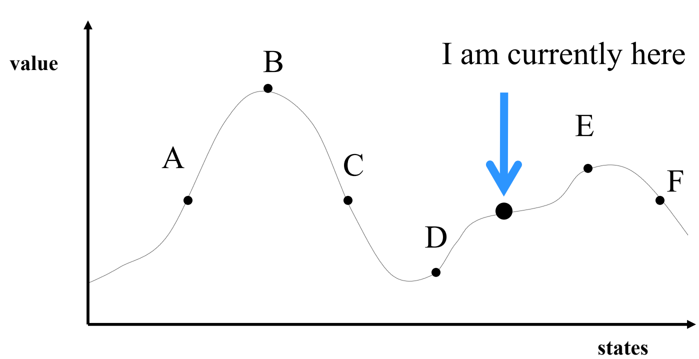
- Extensions to multiple current states:
  - **Genetic algorithms**
  - **Beam search**

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

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

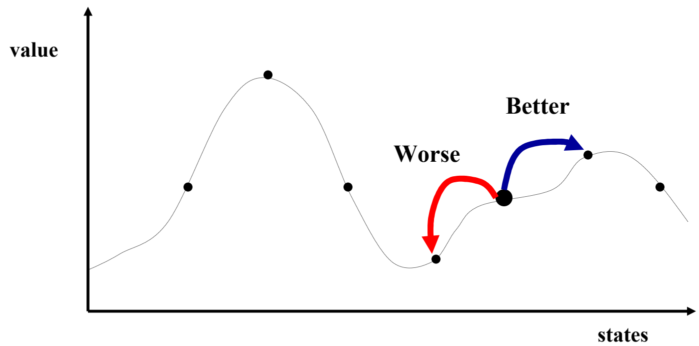
Hill climbing

- What configurations are considered next?
- What move the hill climbing makes?



Hill climbing

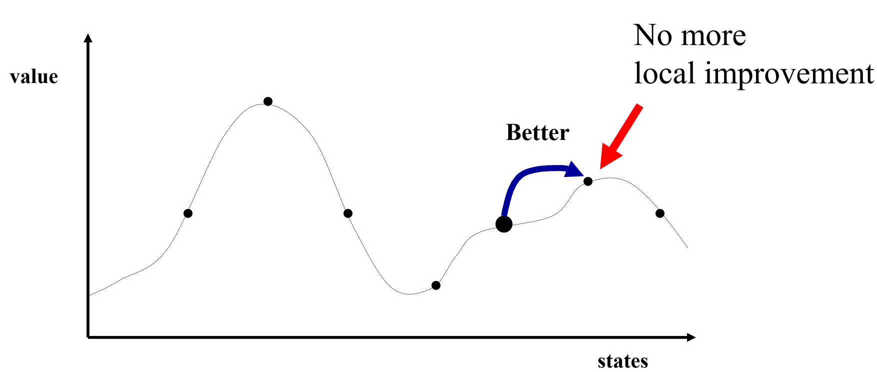
- Look at the local neighborhood and choose the one with the best value



- **What can go wrong?**

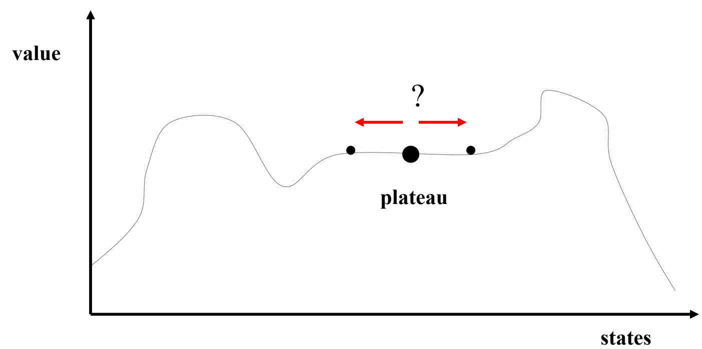
Hill climbing

- Hill climbing can get trapped in the local optimum



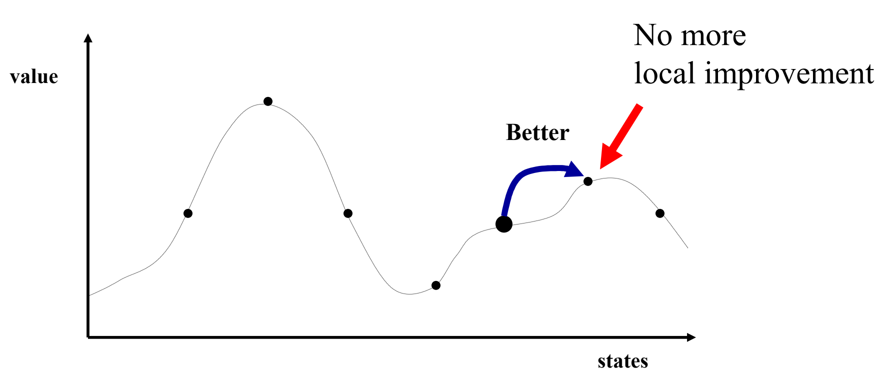
Hill climbing

- Hill climbing can get clueless on plateaus



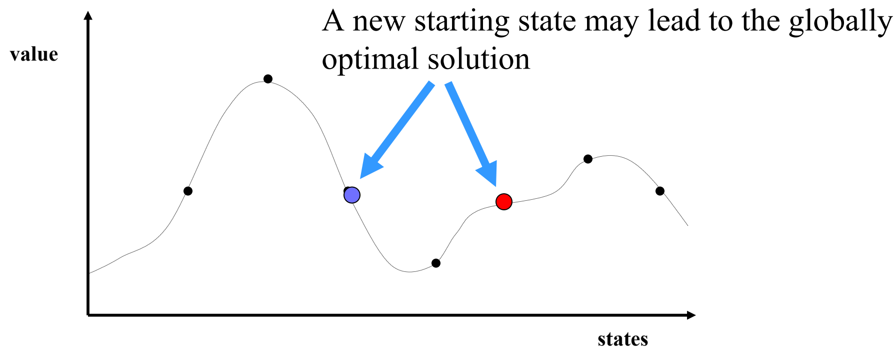
Hill climbing

- How to remedy the problem of local optima?



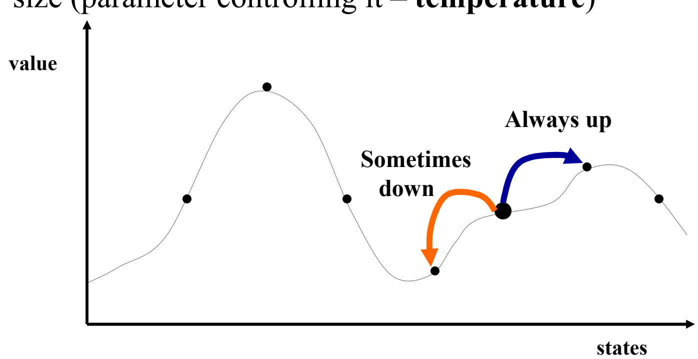
Hill climbing

- Multiple restarts of the hill climbing algorithms from different initial states.



Simulated annealing

- An alternative to solve the local optima problem
- Permits "bad" moves to states with a lower value hence lets us escape states that lead to a local optima
- **Gradually decreases** the frequency of such moves and their size (parameter controlling it – **temperature**)



Simulated annealing algorithm

Chooses uniformly at random one of the local neighbors of the current state – call it a candidate state

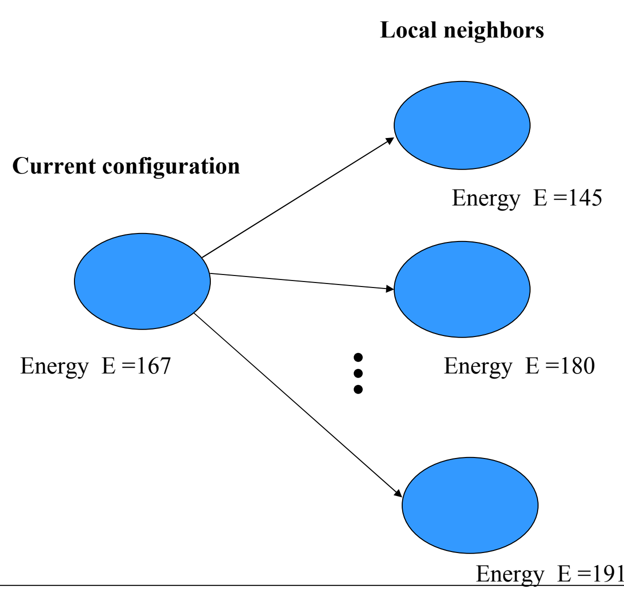
The probability of making a move into that candidate state:

- The probability of moving into a state with a higher objective function value is 1
- The probability of moving into a candidate state with a lower objective function value is
- Let E denotes the objective function value (also called energy).

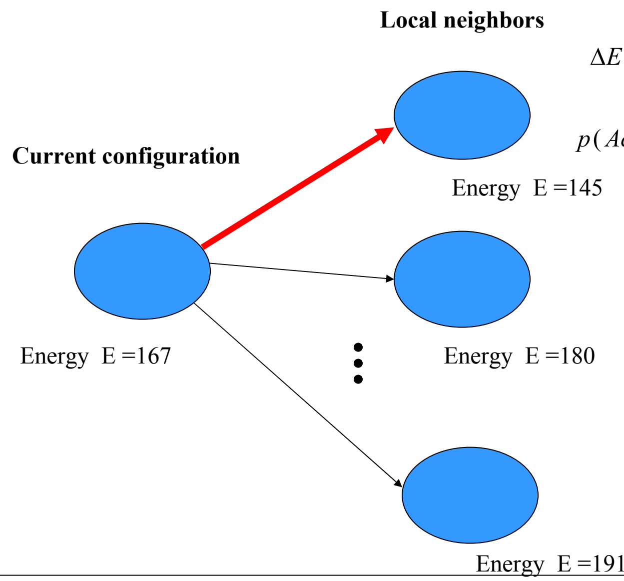
$$p(Accept\ NEXT) = e^{\Delta E / T} \quad \text{where} \quad \Delta E = E_{NEXT} - E_{CURRENT}, \quad T > 0$$

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

Simulated annealing algorithm



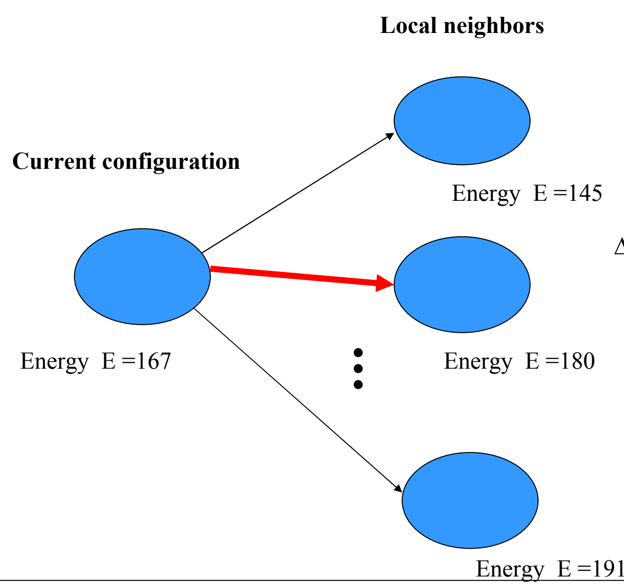
Simulated annealing algorithm



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

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 candidate solutions generated from combination of previous configuration candidates

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 selected pairs
  - Mutation of resulting individuals
- **Evolve the population over multiple generation cycles**

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