CS 1571 Introduction to AI
Lecture 7

Search for optimal configurations

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

Administration

- **PS-2: due today**
- **PS-3: out, due next week**
  - **Programming part:**
    - simulated annealing
    - genetic algorithms

Search for the optimal configuration

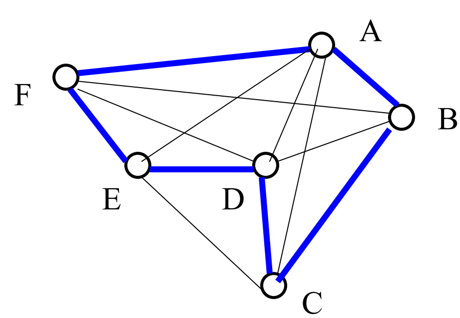
Configuration search problems:
- Are often enhanced with some **quality measure**

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

Goal
- find the configuration with the optimal quality

Example: Traveling salesman problem

Problem:
- A graph with distances

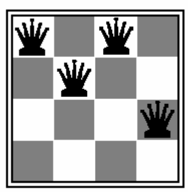


- **Goal:** find the shortest tour which visits every city once and returns to the start

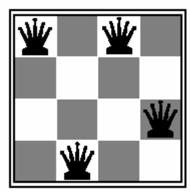
 An example of a valid tour: ABCDEF

Example: N queens

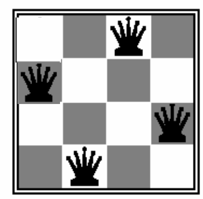
- **Some CSP problems do not have a quality measure**
- **The quality of a configuration in a CSP** can be measured by the number of constraints violated
- Solving corresponds to the minimization of the number of constraint violations



# of violations =3



# of violations =1



# of violations =0

Local search methods

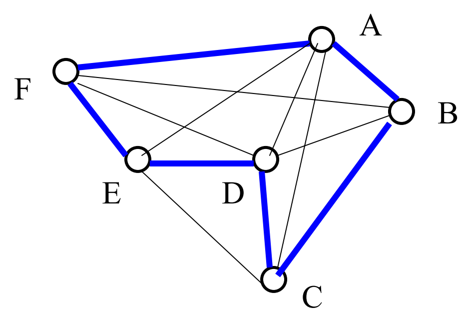
- Are often used to find solutions to large configuration search problems with an additional optimality measure

- **Properties of local search algorithms:**
  - **Search the space of "complete" configurations**
  - **Operators make "local" changes to "complete" configurations**
  - **Keep track of just one state (the current state), not a memory of past states**
    - **!!! No search tree is necessary !!!**

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: 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:

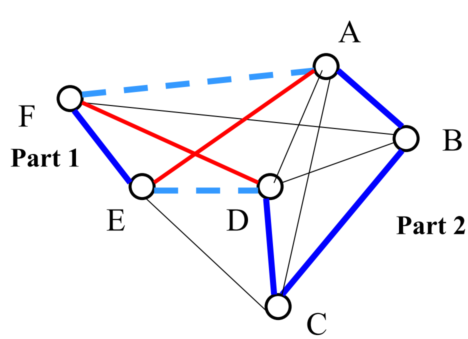
ABCDEF

⇩

ABCD | EF |

⇩

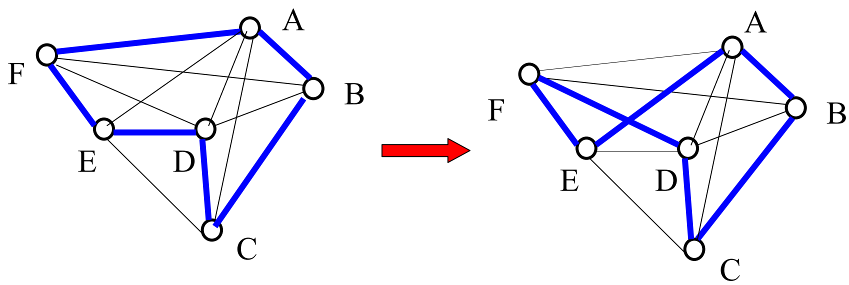
ABCDFE



Example: Traveling salesman problem

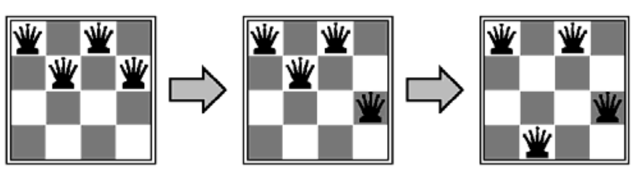
"Local" operator:

– generates the next configuration (state)



Example: N-queens

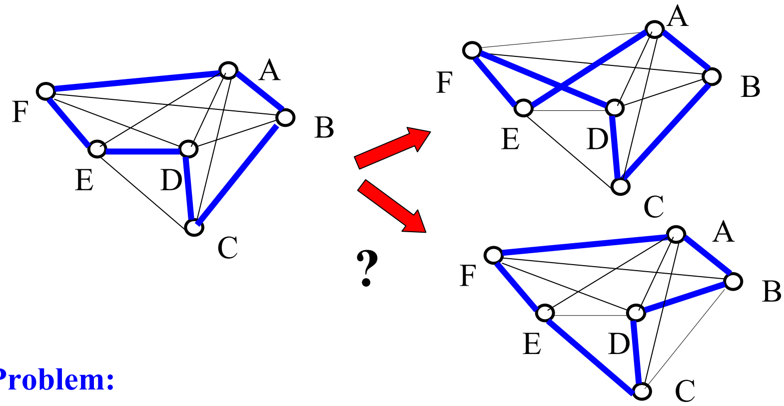
- **"Local" operators for generating the next state:**
  - Select a variable (a queen)
  - Reallocate its position



Searching configuration space

Local search algorithms

- keep only one configuration (the current configuration) active



Problem:

- How to decide about which operator to apply?

Local search algorithms

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

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

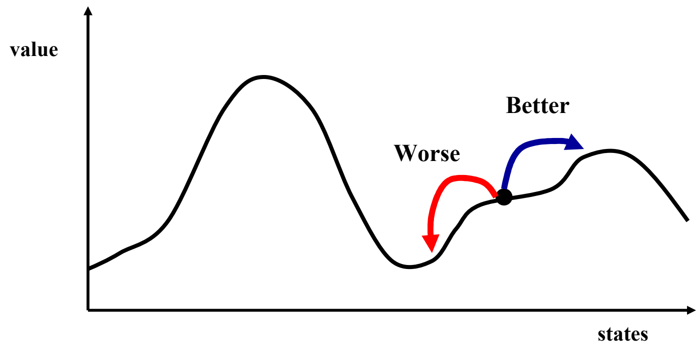
- Later: Extensions to multiple current states:
  - **Genetic algorithms**

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

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

Hill climbing

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



Hill climbing

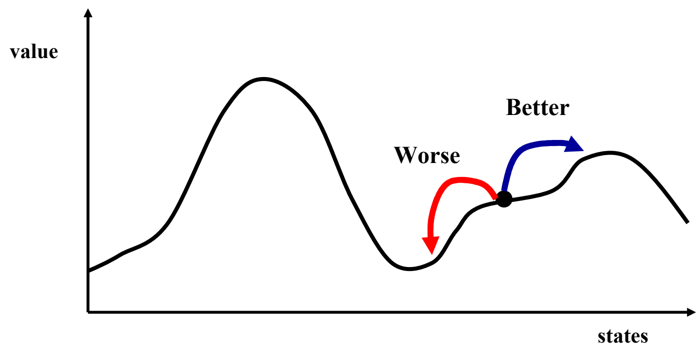
- Always choose the next best successor state
- Stop when no improvement possible

```
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

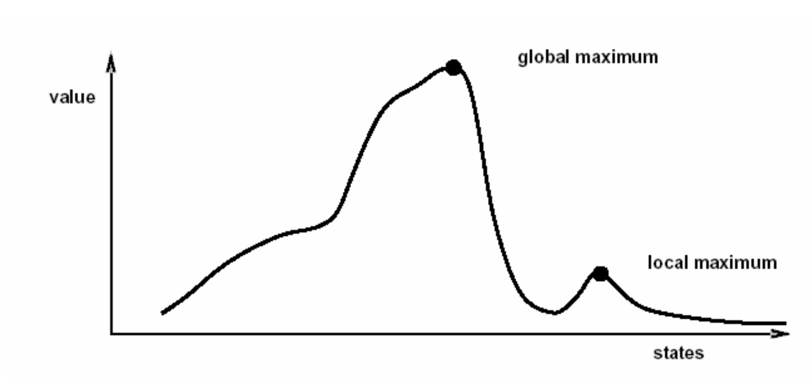
- Local improvement algorithm
- Look around at states in the local neighborhood and choose the one with the best value



- **What can go wrong?**

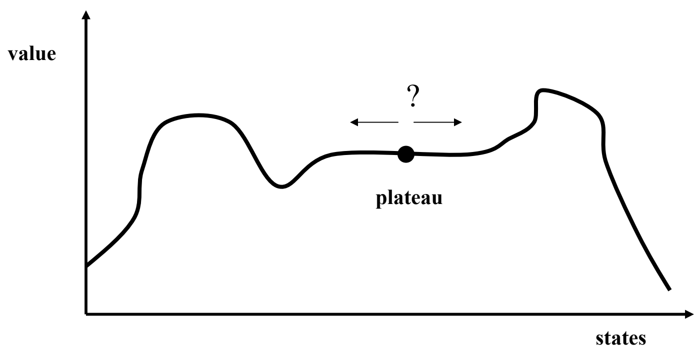
Hill climbing

- Hill climbing can get trapped in the local optimum



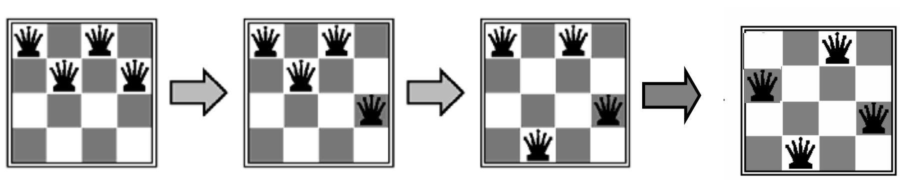
Hill climbing

- Hill climbing can get clueless on plateaus



Hill climbing and n-queens

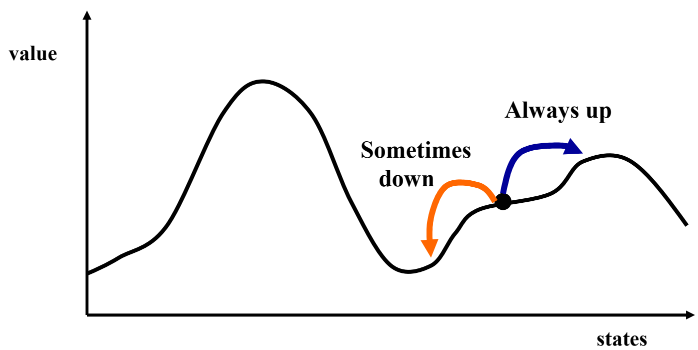
- The quality of a configuration given by the number of constraints violated
- **Then: Hill climbing** reduces the number of constraints
- **Min-conflict strategy (heuristic):**
  – Choose randomly a variable with conflicts
  – Choose its value such that it violates the fewest constraints



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

$$e^{\Delta E / T}$$

The probability is:

- **Proportional to the value (energy) difference** ΔE
- **Modulated through a temperature parameter T**:
  - for $T \rightarrow \infty$ the probability of any move approaches 1
  - for $T \rightarrow 0$ 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;
  - 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:

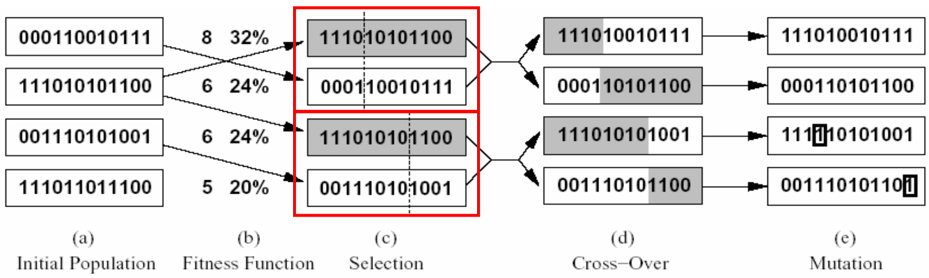
- **Create a population of random configurations**
- **Create a new population through:**
  - Biased selection of pairs of configurations from the previous population
  - Crossover (combination) of pairs
  - Mutation of resulting individuals
- **Evolve the population over multiple generation cycles**

- **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



Genetic algorithms

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     parents ← SELECTION(population, FITNESS-FN)
     population ← REPRODUCTION(parents)
  until some individual is fit enough
  return the best individual in population, according to FITNESS-FN
```

Fitness function = value function

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

Parametric optimization

- **Configuration search:**
  - Optimizes the measure of the configuration quality
  - Additional constraints are possible
- When state space we search is finite the search problem is called a **combinatorial optimization problem**
- When parameters we want to find are real-valued
  - **parametric optimization problem**

Parametric optimization:

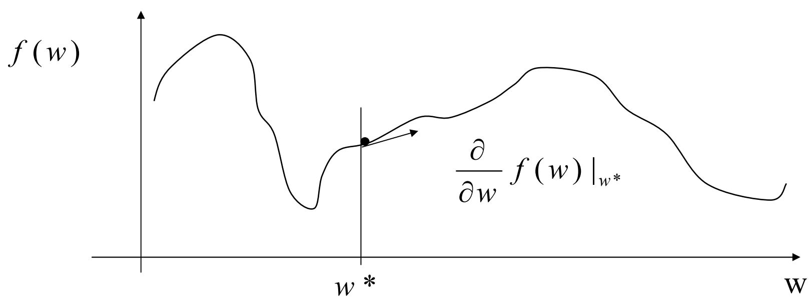
- Configurations are described by a vector of free parameters (variables) **w** with real-valued values
- **Goal:** find the set of parameters **w** that optimize the quality measure $f(\mathbf{w})$

Parametric optimization techniques

- **Special cases (with efficient solutions):**
  - **Linear programming**
  - **Quadratic programming**
- **First-order methods:**
  - **Gradient-ascent (descent)**
  - **Conjugate gradient**
- **Second-order methods:**
  - **Newton-Rhapson methods**
  - **Levenberg-Marquardt**
- **Constrained optimization:**
  - **Lagrange multipliers**

Gradient ascent method

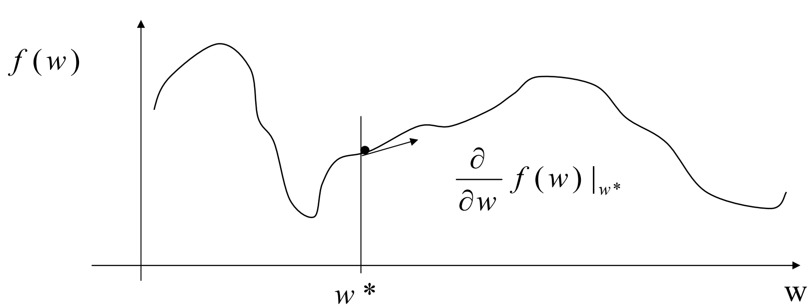
- **Gradient ascent:** the same as hill-climbing, but in the continuous parametric space **w**



- Change the parameter value of w according to the gradient

$$w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w)|_{w^*}$$

Gradient ascent method



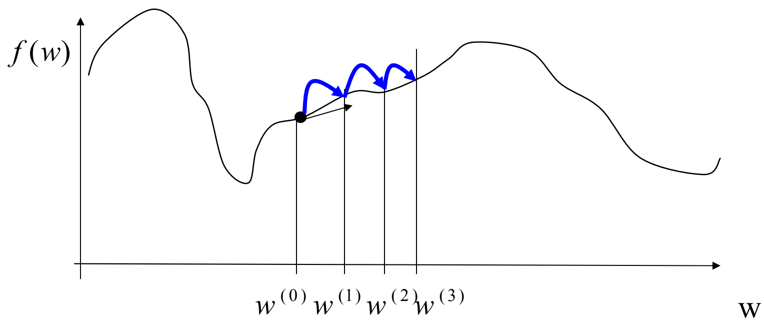
- New value of the parameter

$$w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w)|_{w^*}$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient ascent method

- To get to the function minimum repeat (iterate) the gradient based update few times



- **Problems:** local optima, saddle points, slow convergence
- More complex optimization techniques use additional information (e.g. second derivatives)