Local Search

Lecture 6

What search algorithms arise if we relax our assumptions – instead of systemically searching alternative paths, evaluate/modify one or more current states?

Agenda

- Local search
  - Optimization
- Objective functions
- Hill Climbing
- Simulated Annealing
- Local Beam Search
- Genetic Algorithms
- Continuous state spaces

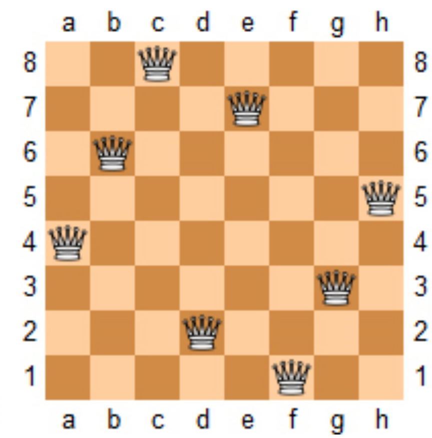
Motivation

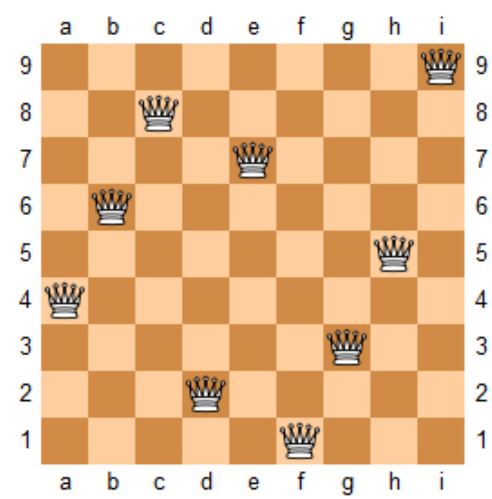
- We have studied algorithms that explore search spaces via paths, keeping alternatives in memory
  - This gets expensive!
- In many problems, path to goal is irrelevant – the final state is all that matters (called **complete-state** formulations, vs. **partial-state**)
- **Local search** algorithms operate using a **current node**, moving only to neighbors, typically…
  - require little memory (often constant)
  - find reasonable solutions (possibly random restarts)

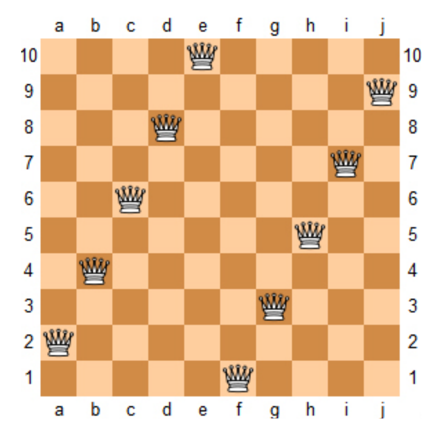
Example: N-Queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
  - How many states?

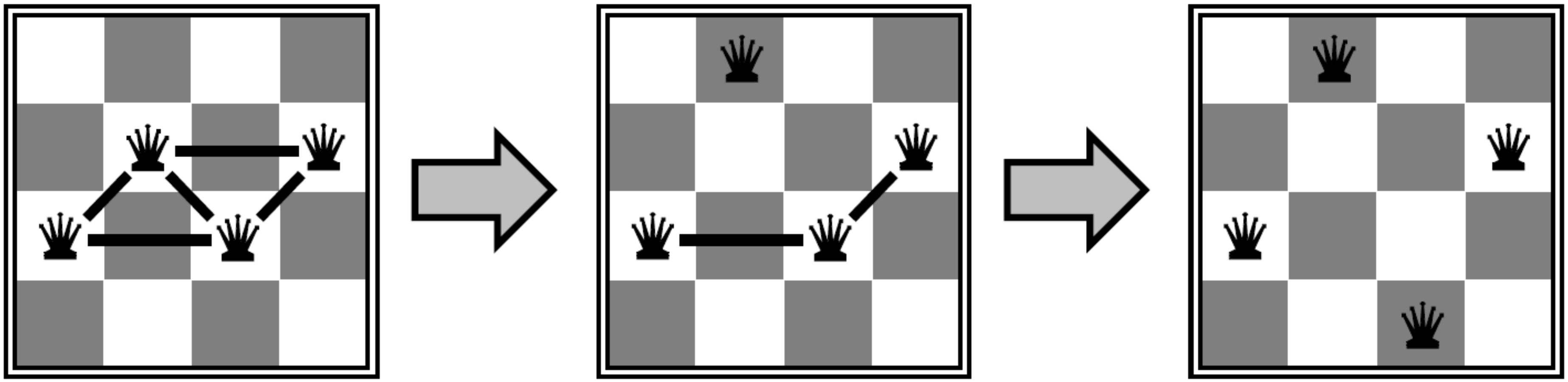
- Notice "path" to solution doesn't matter, just final placement







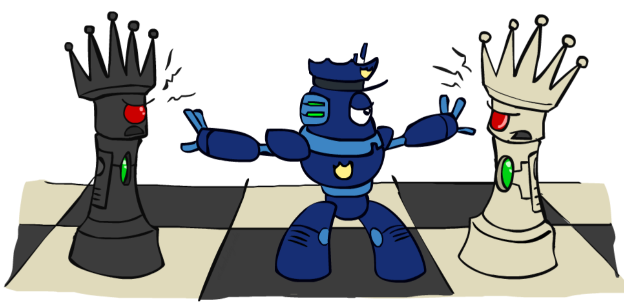
Iterative Improvement



h = 5 h = 2 h = 0

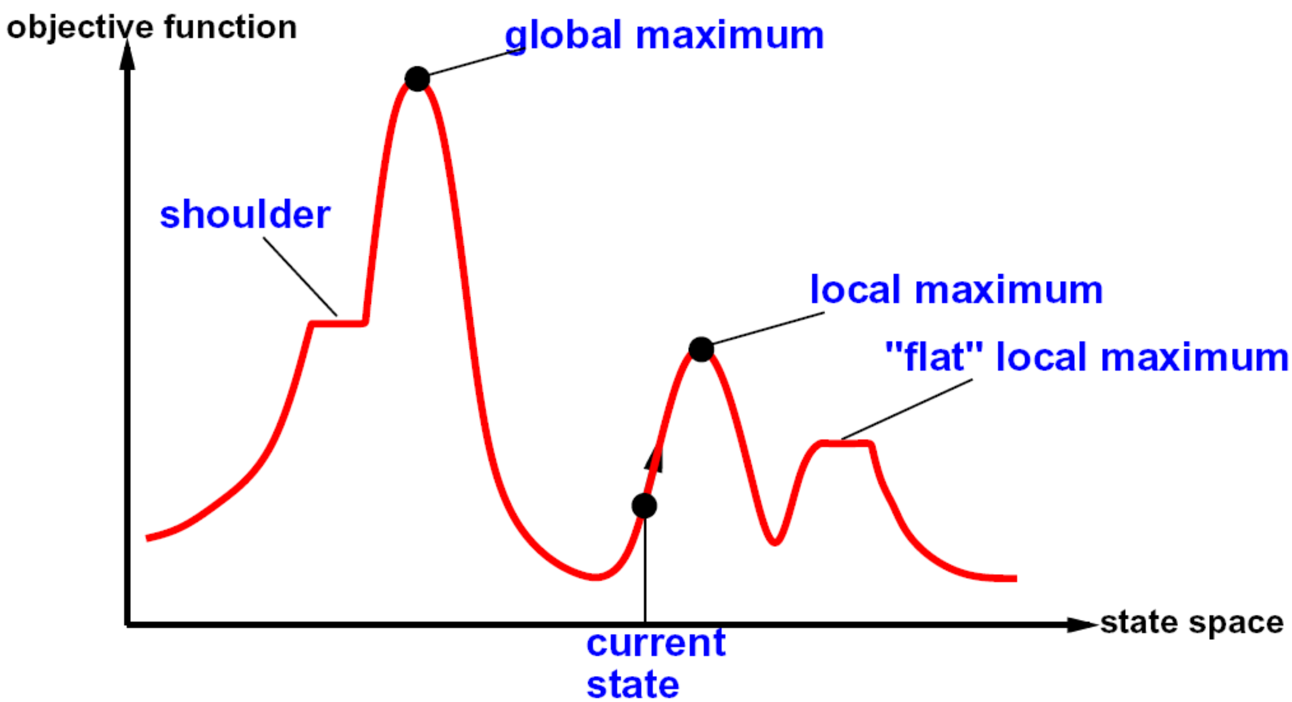
Can typically solve even large problems quickly via incremental moves

N-Queens



Solving Optimization Problems

In addition to search, local search algorithms are useful for solving **optimization problems**, in which the aim is to find the best state according to an **objective function**



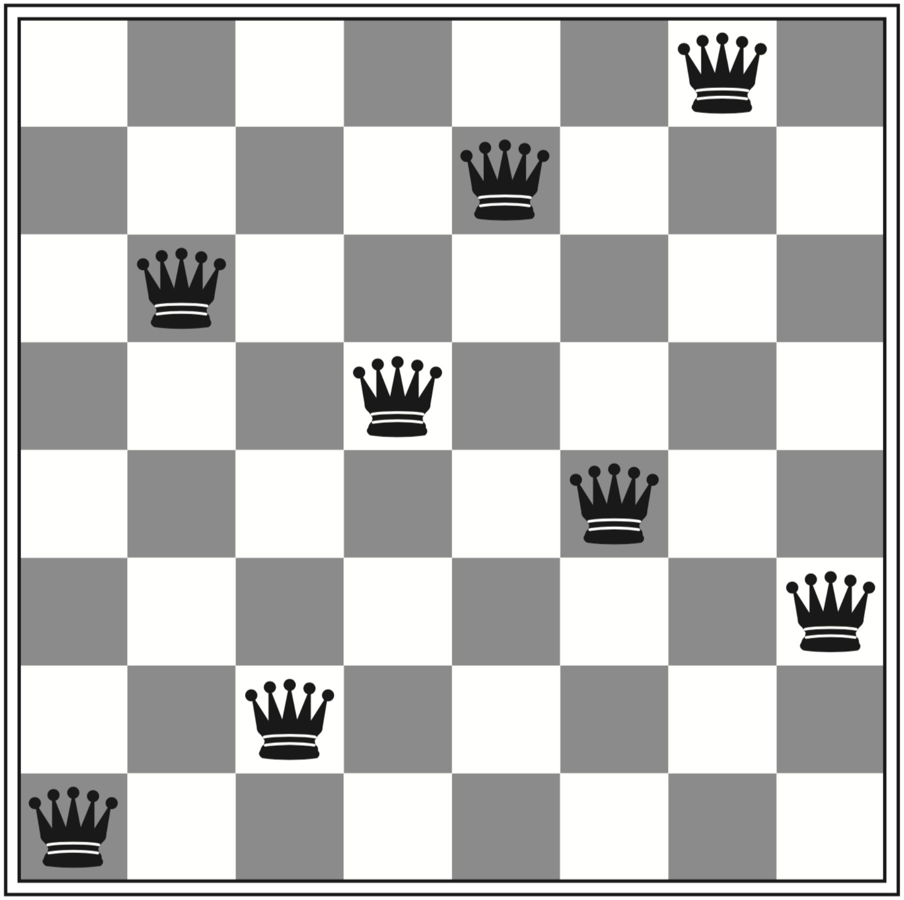
Hill Climbing

Algorithm

- Start wherever
- Repeat: move to the best neighboring state
- If no neighbors better than current, quit

Many variants exist

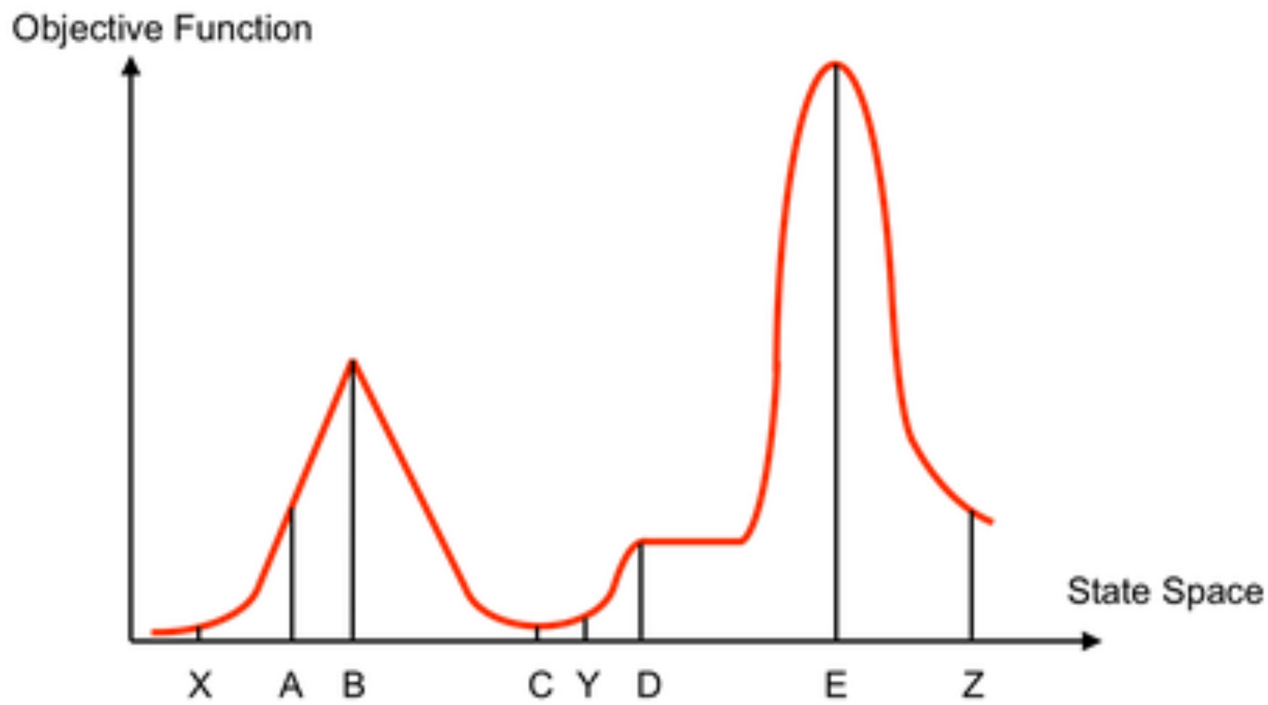
Checkup



- Complete?
- Random restarts fixes this trivially, why?

Checkup

- Where do you end up if you start from…
  - X
  - Y
  - Z
- Optimal?



Simulated Annealing (1)

- Basic idea: escape local maxima by allowing downhill moves
- But make them rarer as time goes on
- Theory: if slow enough, will converge to optimal state!

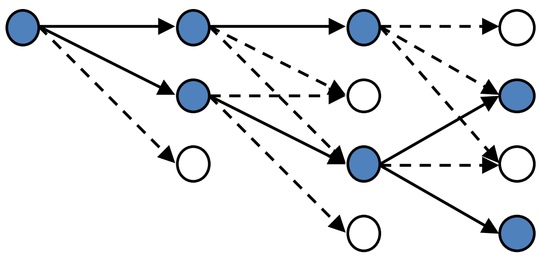
Simulated Annealing (2)

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

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

Local Beam Search

- Start: k randomly generated states
- Loop: generate successors, select k best
  - If any are goal, stop

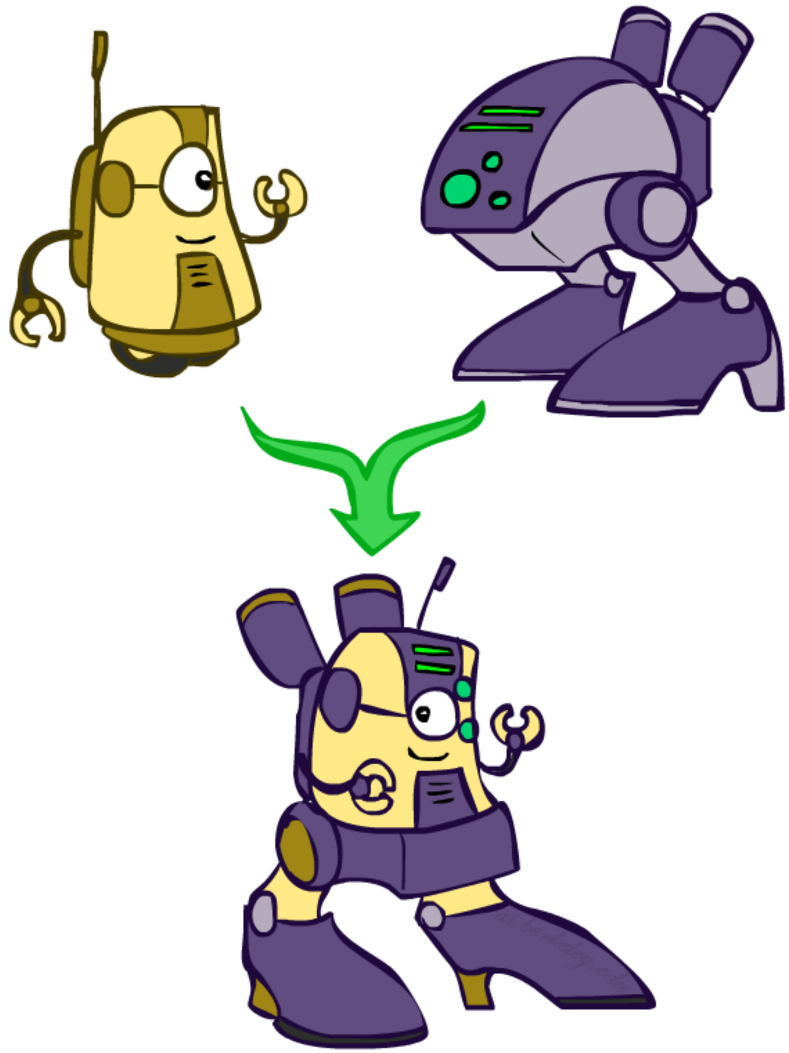


- Variant: **stochastic beam search**
  - Choose k at random, with probability being an increasing function of the value

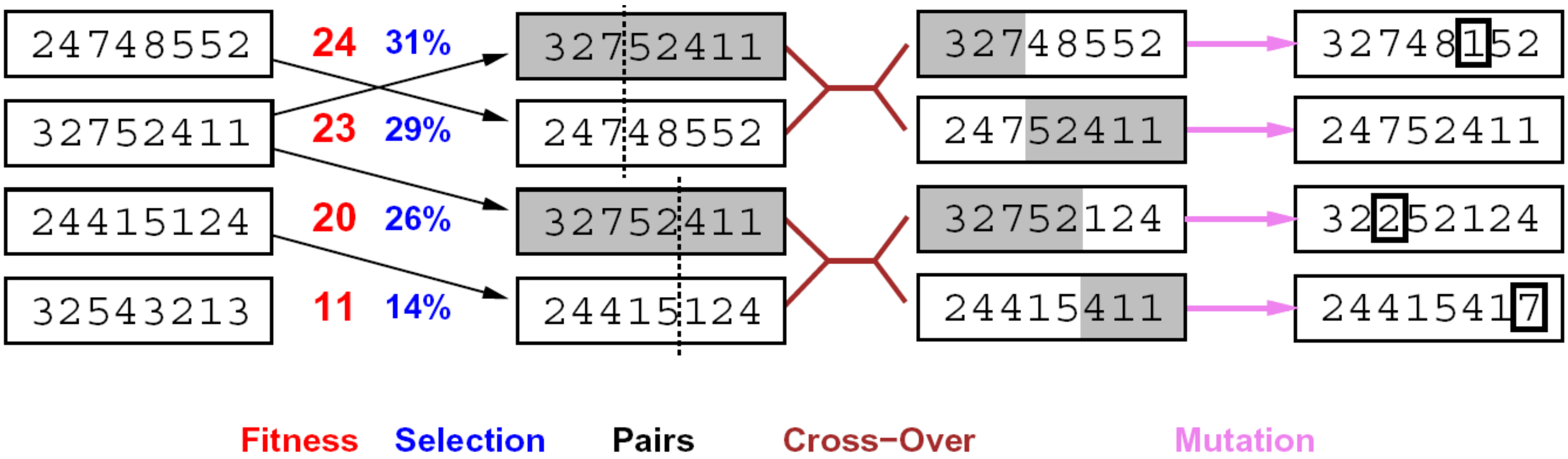
Genetic Algorithms (1)

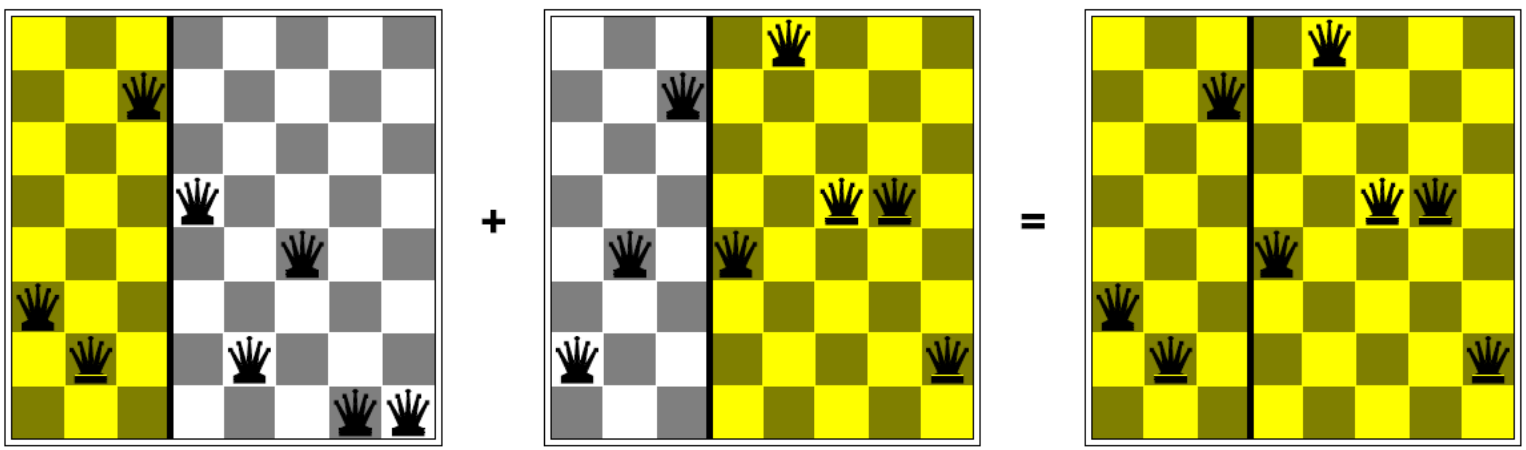
- Basic idea:
  - stochastic beam search + generate successors from *pairs* of states

- Possibly the most misunderstood, misapplied technique



Genetic Algorithms (2)



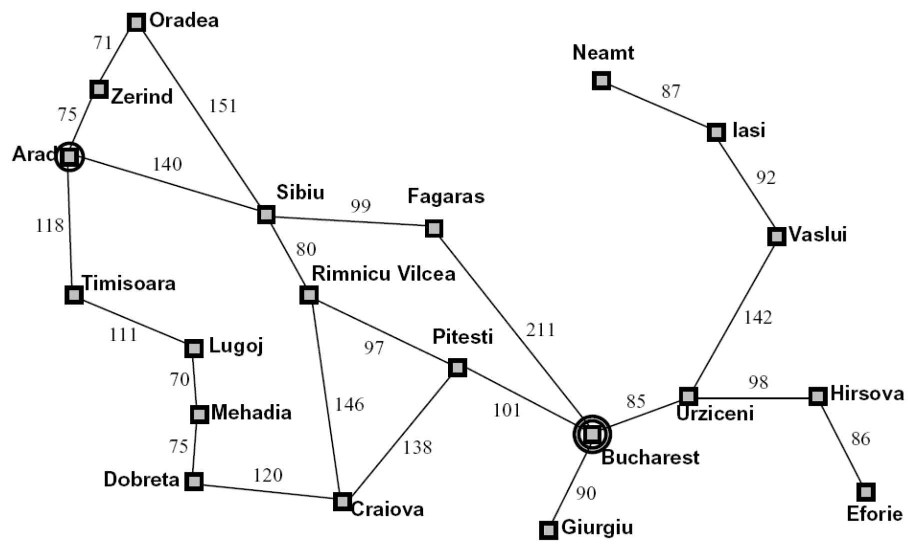


Local Search in Continuous Spaces

- Most of the algorithms we have mentioned thus far work only in discrete state spaces
  - Infinite branching factor!
- Sometimes we can take a continuous problem and discretize the neighborhood of each state
- But how to perform truly continuous search?
  - Long topic, here's a flavor

Example

- Where to locate three airports in Romania?
- Objective: minimize the sum of squared distances to each city on the map



$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C} (x_i - x_c)^2 + (y_i - y_c)^2$$

Using the Gradient

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C} (x_i - x_c)$$

- Finding the magnitude and direction of steepest slope could be used to find the minimum via **gradient methods**
- Often cannot be solved in closed form, so the **empirical gradient** is determined via evaluating the response $\pm\delta$

Hill Climbing

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \nabla f(\boldsymbol{x})$$

The step size (α) is a small constant

- Variety of methods for choosing/updating the value

Newton's Method

- A method for finding successively better approximations to the roots of a function

  $f(x) = 0$ via $x \leftarrow x - \frac{f(x)}{f'(x)}$

- In this case, finding extrema via…

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{H}_f^{-1}(\boldsymbol{x}) \nabla f(\boldsymbol{x})$$

where H is the Hessian

Summary

- **Local search** methods operate on **complete-state** formulations and keep only a small number of nodes in memory
- **Hill-climbing** methods can get stuck in **local optima** and stochastic methods, such as **simulated annealing**, can return optimal solutions under certain conditions
- A **genetic algorithm** is a stochastic hill-climbing search operating over a large population in which new states are generated by mutation and crossover
- Local search in continuous spaces often involves evaluating the **gradient** of the objective function