Artificial Intelligence

Local search

Instructors: Fatemeh Mansoori

University of Isfahan

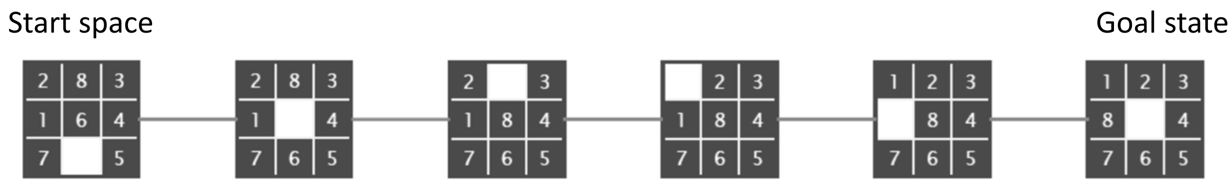
Artificial Intelligence: A Modern Approach", 3rd Edition, Chapter 4
Most slides have been adapted from CS188, UC Berkeley

Outline

- Local search & optimization algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

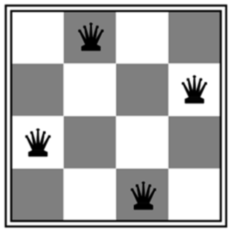
Review:Search

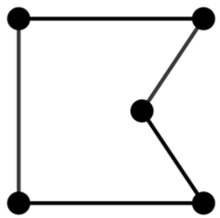
- Find a path with minimum cost from start state to goal state in state space graph
  - Systematic exploration of search space



Local search algorithms

- In many optimization problems, ***path*** is irrelevant; the goal state ***is*** the solution
  - state space = set of “complete” configurations;
  - For e.g. in solving n-queen by systematic search, each state is not a complete configuration
- find **configuration satisfying constraints**, e.g., n-queens problem;
- find ***optimal configuration***, e.g., travelling salesperson problem





Local search algorithms

- use **iterative improvement** algorithms: keep a single “current” state, try to improve it
- Constant space, suitable for online as well as offline search
- More or less unavoidable if the “state” is yourself (i.e., learning)

Sample problems for local & systematic search

- Path to goal is important
  - Theorem proving
  - Route finding
  - 8-Puzzle
  - Chess
- Goal state itself is important
  - 8 Queens
  - TSP
  - VLSI Layout
  - Job-Shop Scheduling
  - Automatic program generation

Local Search

- Tree search keeps unexplored alternatives on the frontier (ensures completeness)

- Local search: improve a single option until you can't make it better (no fringe!)
  - New successor function: local changes

- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing

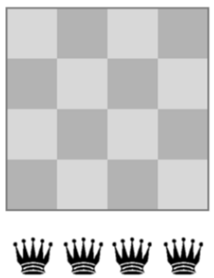
- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What's about this approach?
  - Complete?
  - Optimal?
  - Benefit ?

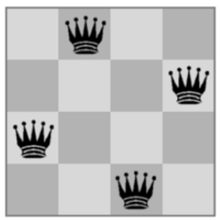
n-queens problem

- Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- Systematic search :
  - What is state-space?
  - What is the size of state space?

Start state



Goal state



Hill-climbing algorithm

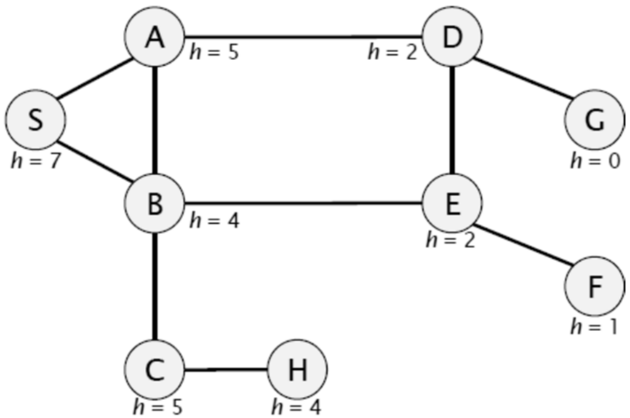
- Node contains the state and the value of objective function in that state (not path)
- Search strategy: steepest ascent among immediate neighbors until reaching a peak

function HILL-CLIMBING(problem) **returns** a state

```
current ← make-node(problem.initial-state)
loop do
    neighbor ← a highest-valued successor of current
    if neighbor.value ≤ current.value then
        return    current.state
    current ← neighbor
```

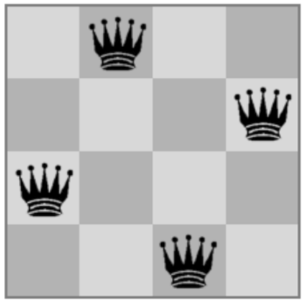
- Current node is replaced by the best successor (if it is better than current node)
- ***"Like climbing Everest in thick fog with amnesia"***

Run hill climbing algorithm



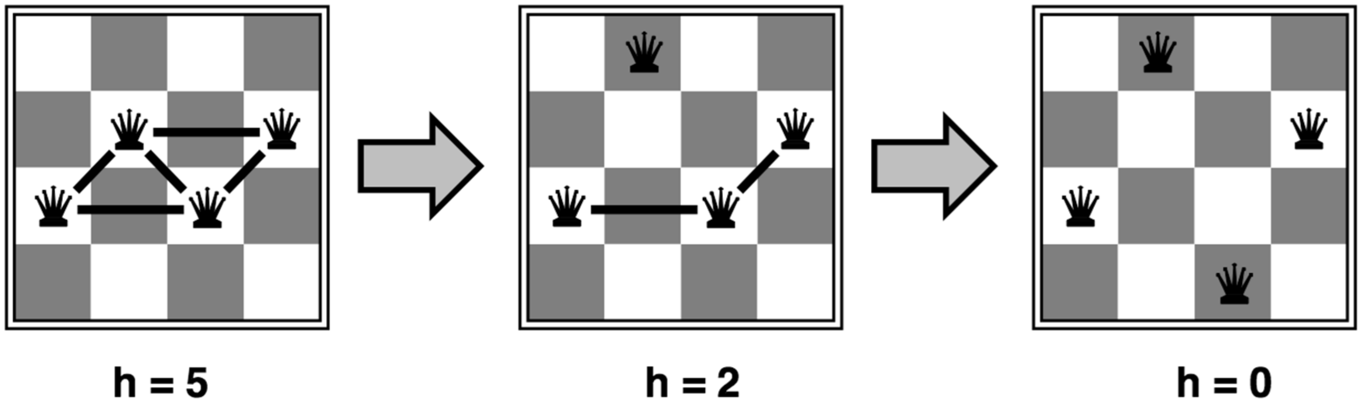
n-queens problem

- Modeling the state as a whole configuration
  - States: n queens on board, one per column
  - Successors: move a queen in its column
  - What is the branching factor ?
  - What is objective function?



Heuristic for *n*-queens problem

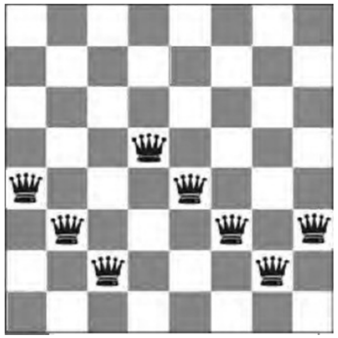
- Heuristic value function: number of conflicts



Local search: 8-queens problem

- States: 8 queens on the board, one per column ($8^8 \approx 17\,million$)
- Successors(s): all states resulted from s by moving a single queen to another square of the same column ($8{\times}7 = 56$)
- Cost function h(s): number of queen pairs that are attacking each other, directly or indirectly
- Global minimum : $h\,(s) = 0$

$h(s) = 17$



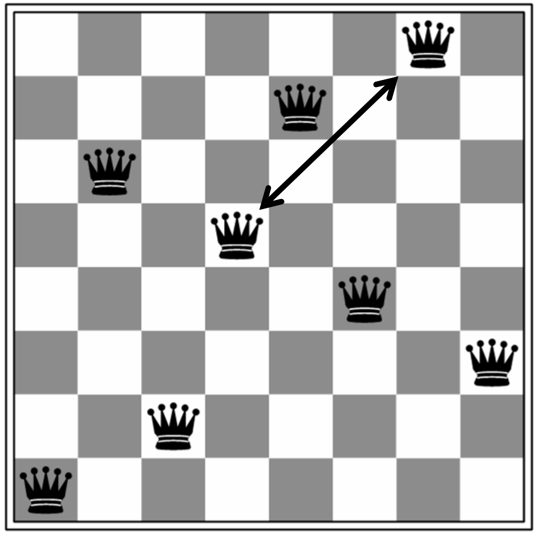
successors objective values

Red: best successors

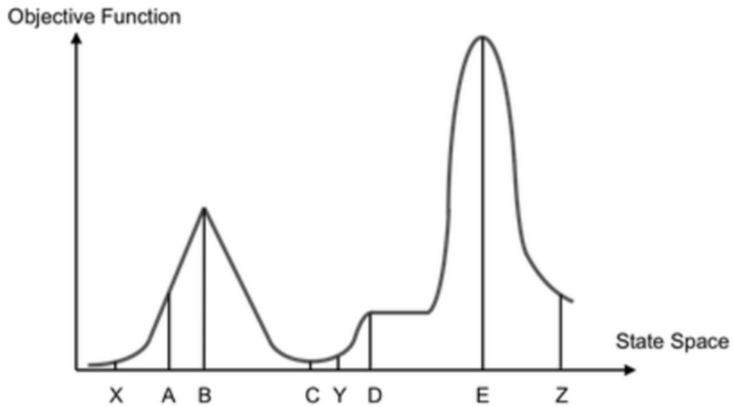
Hill-climbing search: 8-queens

- Result of hill-climbing in this case...

- A local minimum with $h = 1$



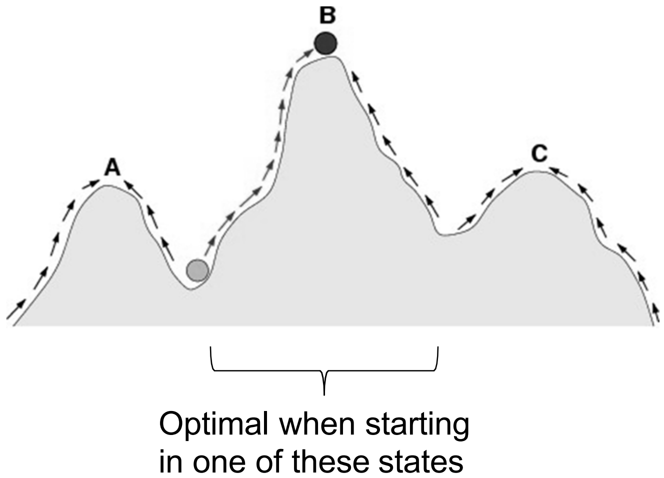
Hill Climbing Quiz



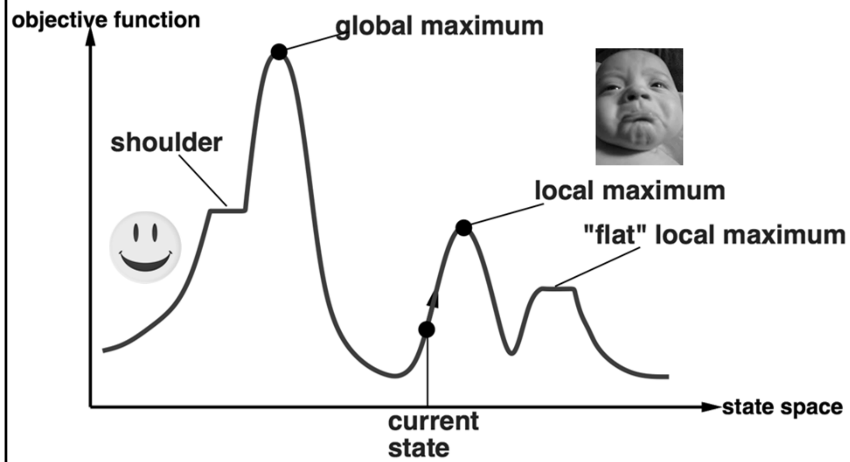
- Starting from X, where do you end up ?
- Starting from Y, where do you end up ?
- Starting from Z, where do you end up ?

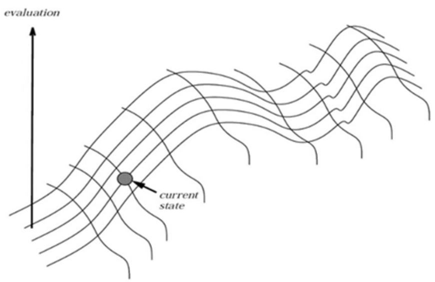
Hill-climbing search is greedy

- Greedy local search: considering only one step ahead and select the best successor state (steepest ascent)
- Rapid progress toward a solution
  - Usually quite easy to improve a bad solution



Global and local maxima





2-d state space

Random restarts

- find global optimum

Random sideways moves

- Escape from shoulders
- Loop forever on flat local maxima

Random-restart hill climbing

- Hill climbing is incomplete
  - Getting stuck on local max

- Hill climbing with random restart
  - **while** state ≠ goal **do**
    - run hill-climbing search from a random initial state

- p: probability of success in each hill-climbing search
  - Expected no of restarts = $1/p$

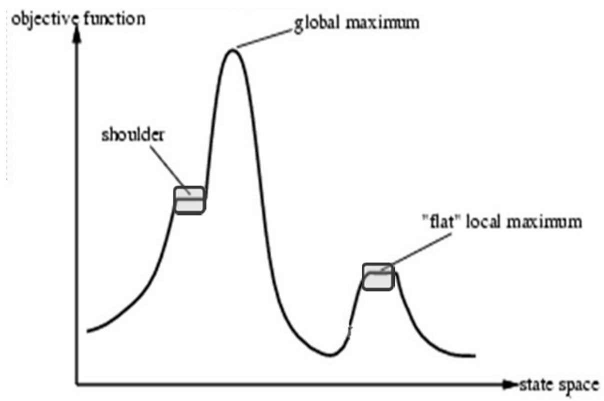
- Reasonable solution can be usually obtained after a small no of restarts

hill climbing variants

- Hill climbing : generate all successors and choose the best one
- Stochastic hill climbing : Randomly chooses among the available uphill moves according to the steepness of these moves
  - $P(S')$ is an increasing function of $h(s') - h(s)$
- First-choice hill climbing: generating successors randomly until one better than the current state is found
  - Good when number of successors is high

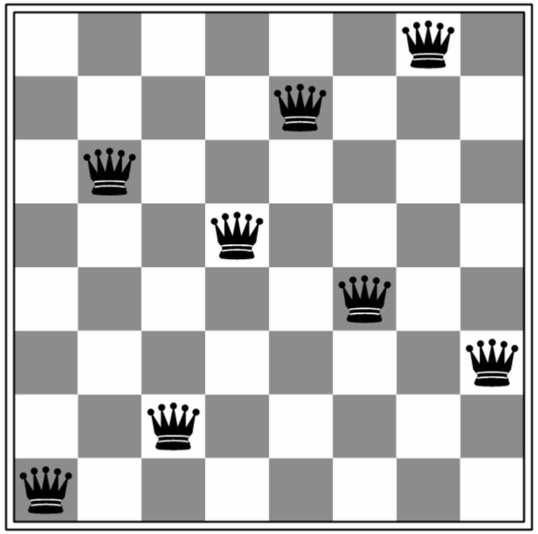
Sideways move

- plateau may be a shoulder so keep going sideways moves when there is no uphill move
  - Problem: infinite loop where flat local max
    - Solution: upper bound on the number of consecutive sideways moves



Hill-climbing on the 8-queens problem

- No sideways moves:
  - Succeeds w/ prob. 0.14
  - Expected number of restart to find the answer ? 1/p = 7
  - Average number of moves per trial (cost of finding solution):
    - 4 when succeeding, 3 when getting stuck
  - Expected total number of moves needed:
    3(1-p)/p + 4 =~ 22 moves
- Allowing 100 sideways moves:
  - Succeeds w/ prob. 0.94
  - Average number of moves per trial:
    - 21 when succeeding, 65 when getting stuck
  - Expected total number of moves needed:
    65(1-p)/p + 21 =~ 25 moves



Simulated annealing

- Resembles the annealing process used to cool metals slowly to reach an ordered (low-energy) state
- Basic idea:
  - Allow “bad” moves occasionally, depending on “temperature”
  - High temperature => more bad moves allowed, shake the system out of its local minimum
  - Gradually reduce temperature according to some schedule
  - Sounds pretty flaky, doesn’t it?

Simulated annealing algorithm

function SIMULATED-ANNEALING(problem,schedule) **returns** a state
inputs : *problem* : a problem
schedule : a mapping from time to "temperature"
current ← problem.initial-state
for t = 1 **to** ∞ **do**
T ←schedule(t) (T is a temperature, controlling probability of downward steps)
if T = 0 **then return** current
next ← a randomly selected successor of current
ΔE ← next.value – current.value
if ΔE > 0 **then** current ← next
else current ← next only with probability $e^{\Delta E/T}$

- Theoretical guarantee:
  - If T decreased slowly enough, will converge to optimal state
  - Convergence can be guaranteed if at each step, T drops no more quickly than C/log n, C=constant, n = # of steps so far

Simulated Annealing

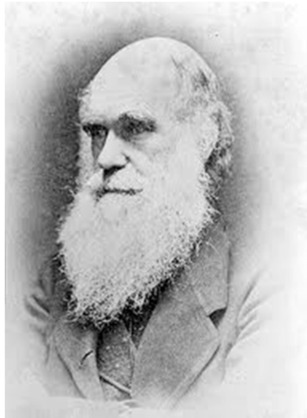
- Is this convergence an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - "Slowly enough" may mean exponentially slowly
  - Random restart hill climbing also converges to optimal state…
- Simulated annealing and its relatives are a key workhorse in VLSI layout and other optimal configuration problems

Local beam search

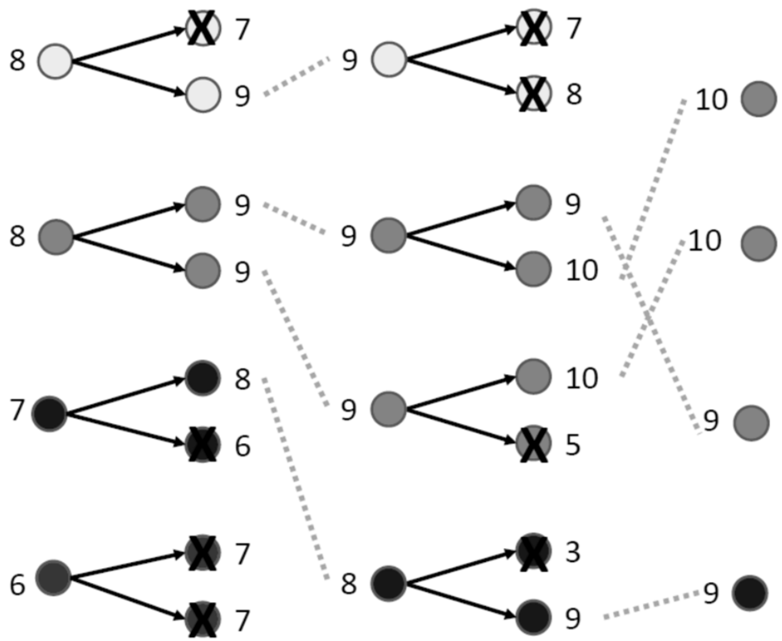
- Basic idea:
  - *K* copies of a local search algorithm, initialized randomly

Or, K chosen randomly with a bias towards good ones

- For each iteration
  - Generate ALL successors from *K* current states
  - If any of successors is goal -> finished
  - Choose best *K* of these to be the new current states



Local beam search example (k=4)



Local beam search

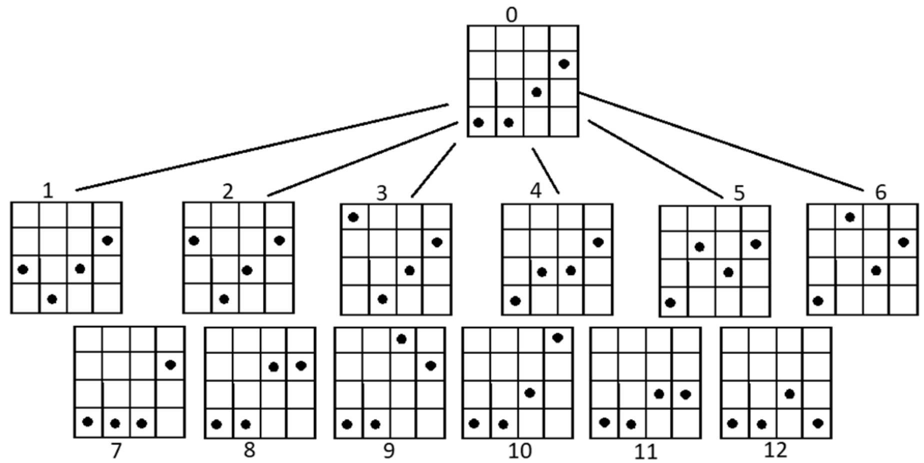
- Why is this different from *K* local searches in parallel?
  - The searches ***communicate***! “Come over here, the grass is greener!”
- Problem: quite often, all *k states end up on same local hill*
  - Idea: Stochastic beam search
    - Choose *k successors randomly, biased towards good ones*
- What other well-known algorithm does this remind you of?
  - Evolution!

الف) در صورت استفاده از first choice hill climbing به کدام همسایه می رویم. (ترتیب ساخته شدن همسایه به ترتیب شماره هاست)

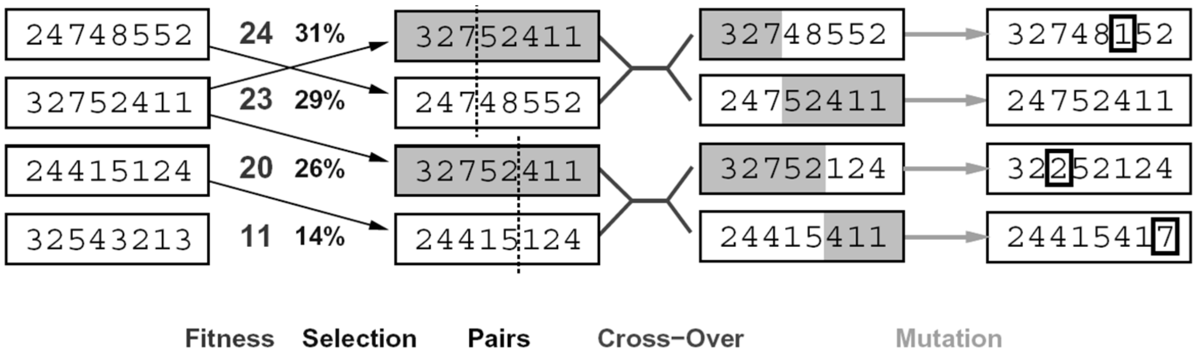
ب) در صورت استفاده از stochastic hill climbing کدام حالت ها ممکن است انتخاب شوند، و احتمال انتخاب هر کدام چقدر است؟

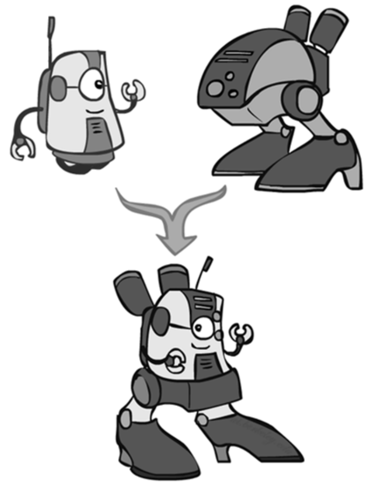
ج) در صورت استفاده از hill climbing معمولی کدام حالت انتخاب می شود؟ و آیا از این حالت می توانیم به هدف برسیم یا یک ماکزیمم محلی است؟

د) در صورت استفاده از الگوریتم beam search اگر k=3 باشد، کدام حالت یا حالت ها انتخاب می شوند؟



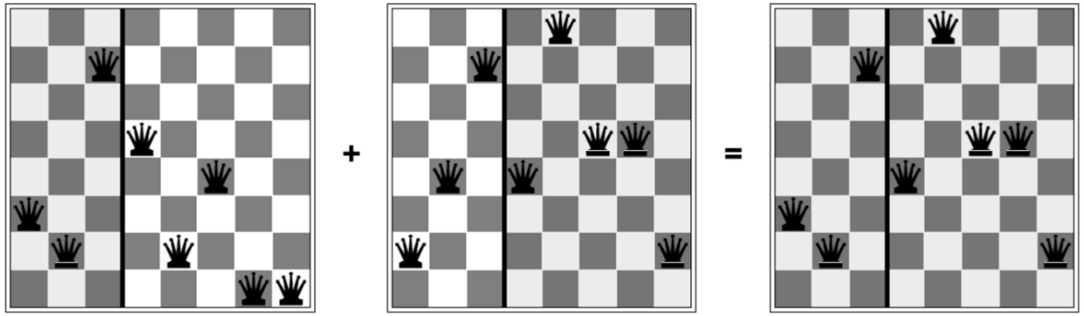
Genetic algorithms





- Genetic algorithms use a natural selection metaphor
  - Resample K individuals at each step (selection) weighted by fitness function
  - Combine by pairwise crossover operators, plus mutation to give variety

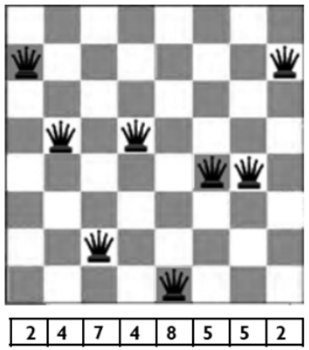
Example: N-Queens



- Does crossover make sense here?
- What would mutation be?
- What would a good fitness function be?

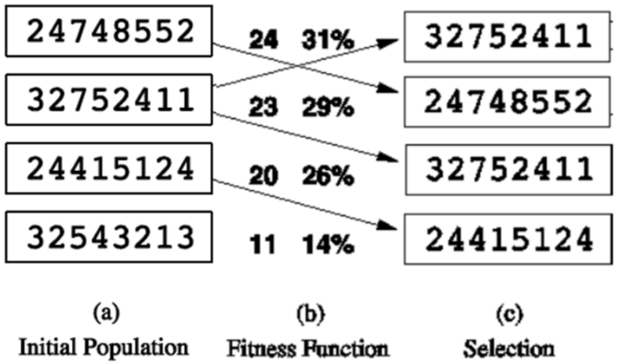
Chromosome & fitness: 8-queens

- Describe the individual (or state) as a string



- Fitness function: number of non-attacking pairs of queens
  - 24 for above figure

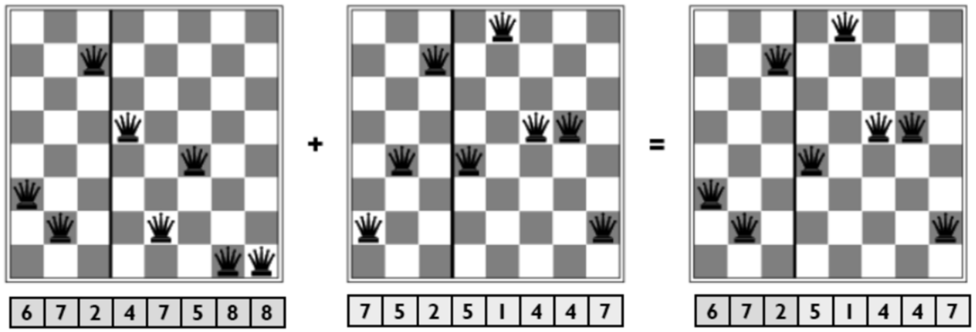
Genetic operators: Selection



- Fitness function: number of non-attacking pairs of queens
  - min = 0, max = 8 × 7/2 = 28
- Reproduction rate(*i*) = $fitness(i)/\sum_{k=1}^{n} fitness(k)$
  - e.g., 24/(24+23+20+11) = 31%

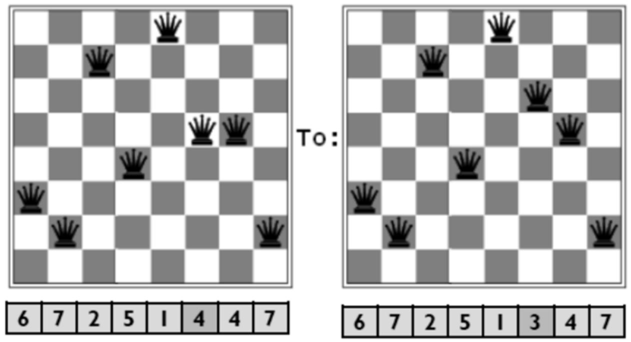
Genetic operators: Cross-over

- To select some part of the state from one parent and the rest from another.

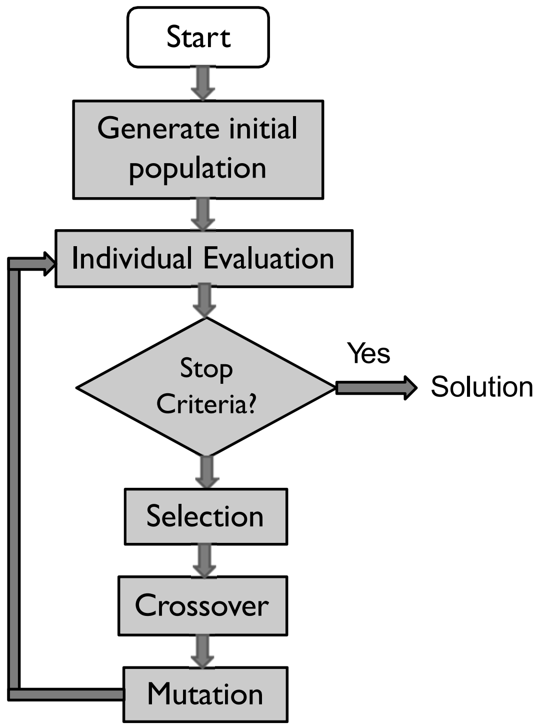


Genetic operators: Mutation

- change a small part of one state with a small probability



A Genetic algorithm diagram



Genetic algorithm properties

- Genetic algorithm is a variant of "stochastic beam search"
- Why does a genetic algorithm usually take large steps in earlier generations and smaller steps later?
  - Initially, population individuals are diverse
    - Cross-over operation on different parent states can produce a state long a way from both parents
  - More similar individuals gradually appear in the population

- Useful on some set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general

Local search vs. systematic search

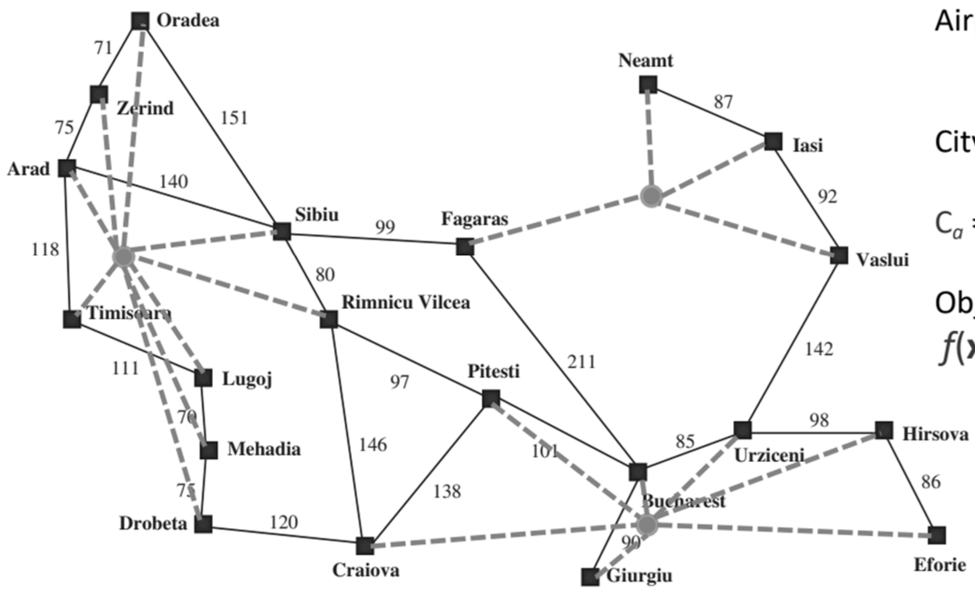
	Systematic search	Local search
Solution	Path from initial state to the goal	Solution state itself
Method	Systematically trying different paths from an initial state	Keeping a single or more "current" states and trying to improve them
State space	Usually incremental	Complete configuration
Memory	Usually very high	Usually very little (constant)
Time	Finding optimal solutions in small state spaces	Finding reasonable solutions in large or infinite (continuous) state spaces
Scope	Search	Search & optimization problems

38

Local search in continuous spaces

Example: Placing airports in Romania

Place 3 airports to minimize the sum of squared distances from each city to its nearest airport



Airport locations

$\mathbf{x} = (x_1,y_1), (x_2,y_2), (x_3,y_3)$

City locations (x_c,y_c)

C_a = cities closest to airport a

Objective: minimize

$f(\mathbf{x}) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$

Finding extrema in continuous space

- Gradient vector $\nabla f(\mathbf{x}) = (\partial f/\partial x_1, \partial f/\partial y_1, \partial f/\partial x_2, \ldots)^T$
- For the airports, $f(\mathbf{x}) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
- $\partial f/\partial x_1 = \Sigma_{c \in C_1} 2(x_1 - x_c)$
- At an extremum, $\nabla f(\mathbf{x}) = 0$
- Can sometimes solve in closed form: $x_1 = (\Sigma_{c \in C_1} x_c)/|C_1|$
  - Is this a local or global minimum of f?
- If we can't solve $\nabla f(\mathbf{x}) = 0$ in closed form…
  - Gradient descent: $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$
- Huge range of algorithms for finding extrema using gradients