Local Search

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Fall 2022

Soleymani

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

Outline

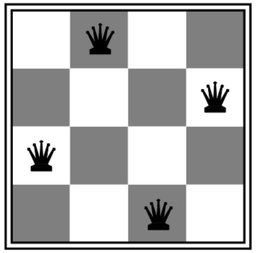
- Local search & optimization algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

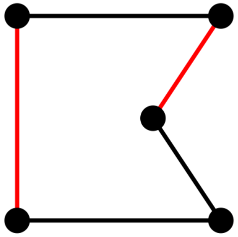
Local search algorithms

- In many optimization problems, ***path*** is irrelevant; the goal state ***is*** the solution

state space = set of "complete" configurations

find ***configuration satisfying constraints***, e.g., n-queens problem; or, find ***optimal configuration***, e.g., travelling salesperson problem



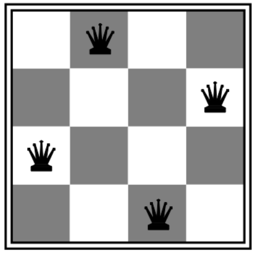


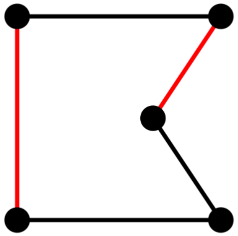
Local search algorithms

- In many optimization problems, ***path*** is irrelevant; the goal state ***is*** the solution

  state space = set of "complete" configurations

find ***configuration satisfying constraints***, e.g., n-queens problem; or, find ***optimal configuration***, e.g., travelling salesperson problem





- In such cases, can 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

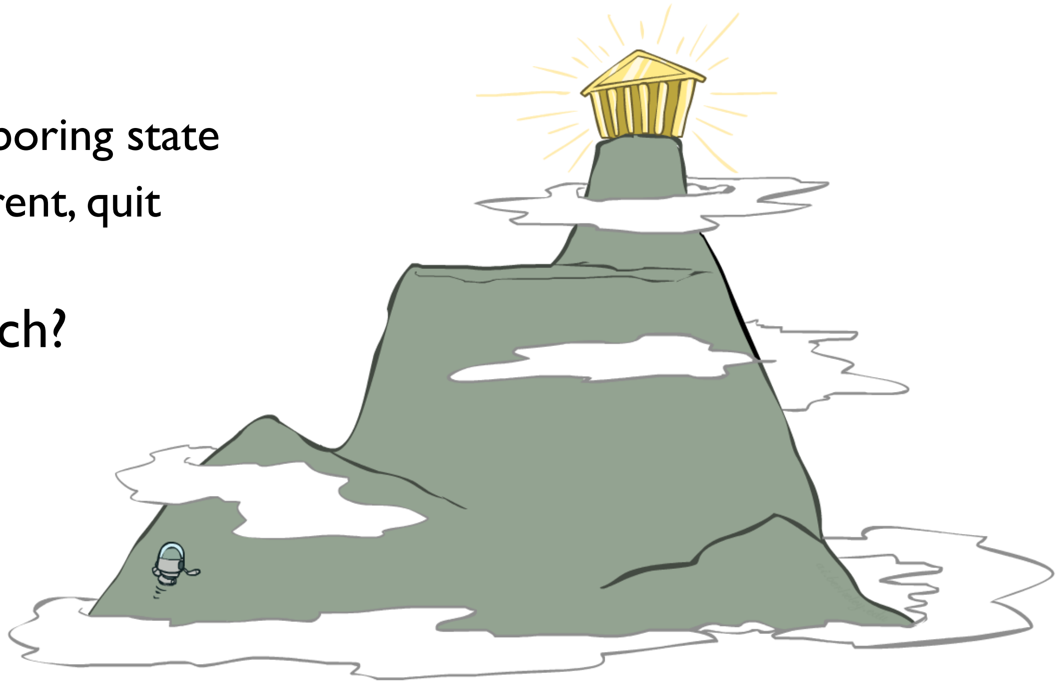
- Tree search keeps unexplored alternatives on the frontier (ensures completeness)

- Local search: improve a single option (no frontier)
  - 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 bad about this approach?
  - Complete?
  - Optimal?
- What's good about it?



Hill-climbing algorithm

Node only 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

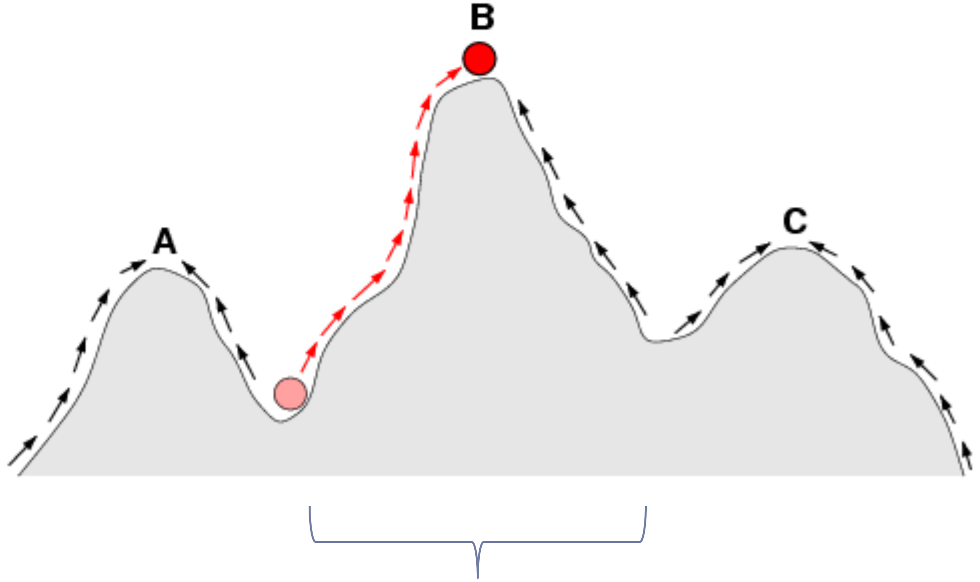
```
function HILL-CLIMBING(problem) returns a state
  current ← make-node(problem.initial-state)
  loop do
      neighbor ← a highest-valued successor of current
      if neighbor.value ≤ current.value then
          return current.state
      current ← neighbor
```

Current node is replaced by the best successor (if it is better than current node)

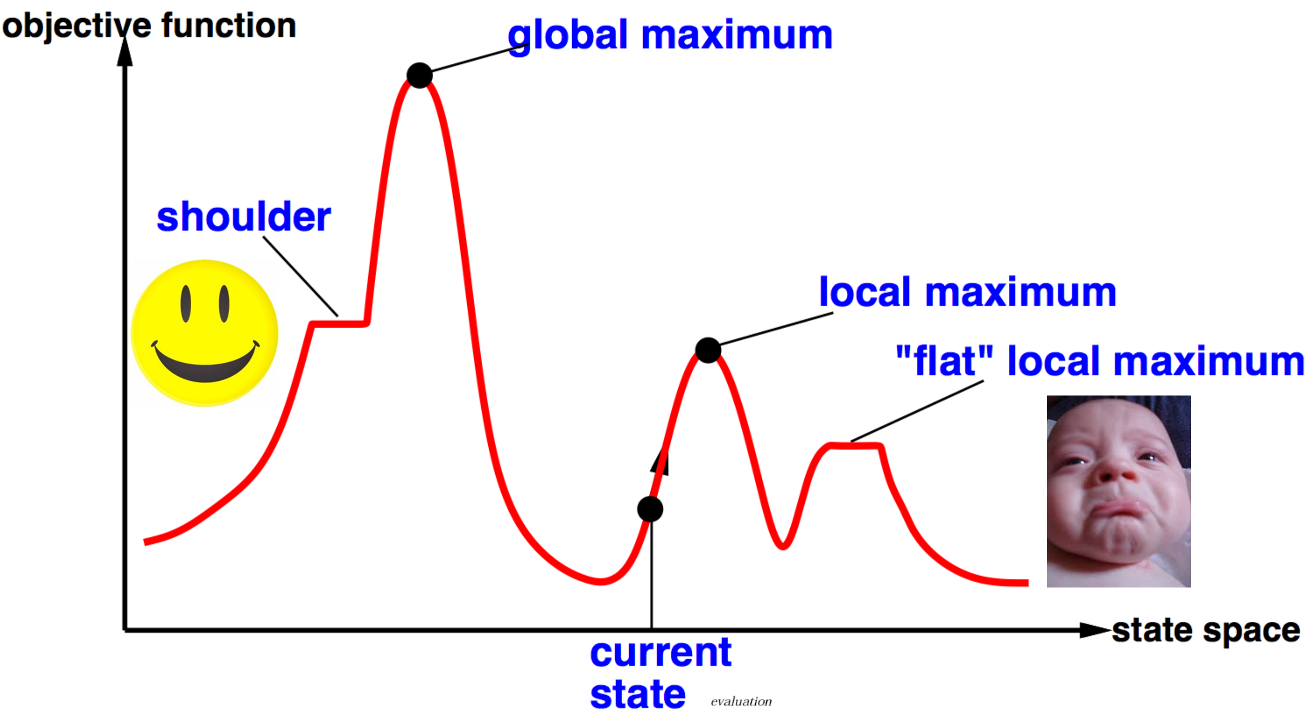
"Like climbing Everest in thick fog with amnesia"

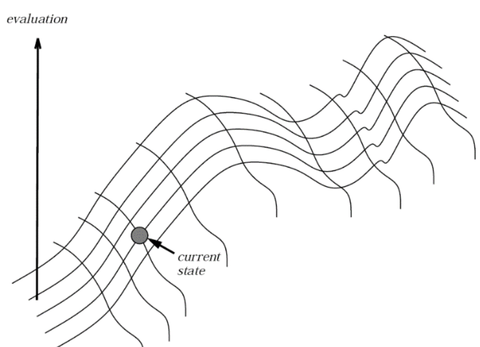
Hill-climbing search is greedy

- 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

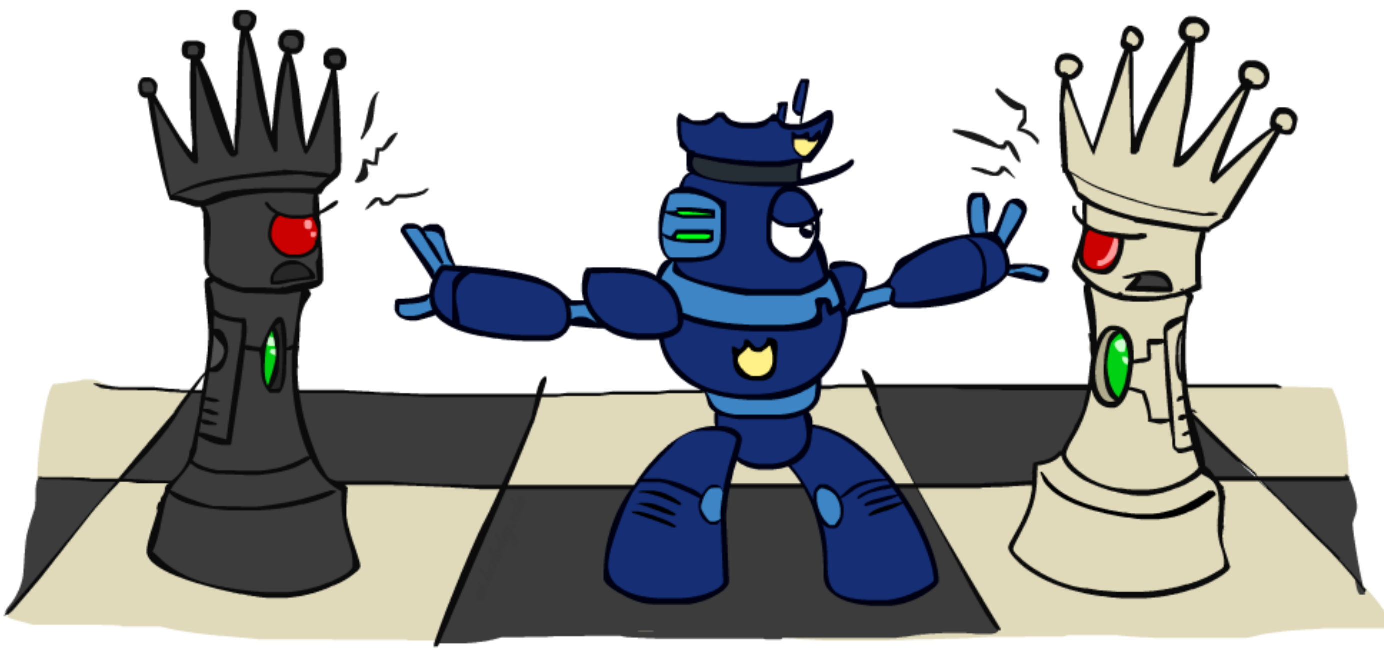




2-d state space

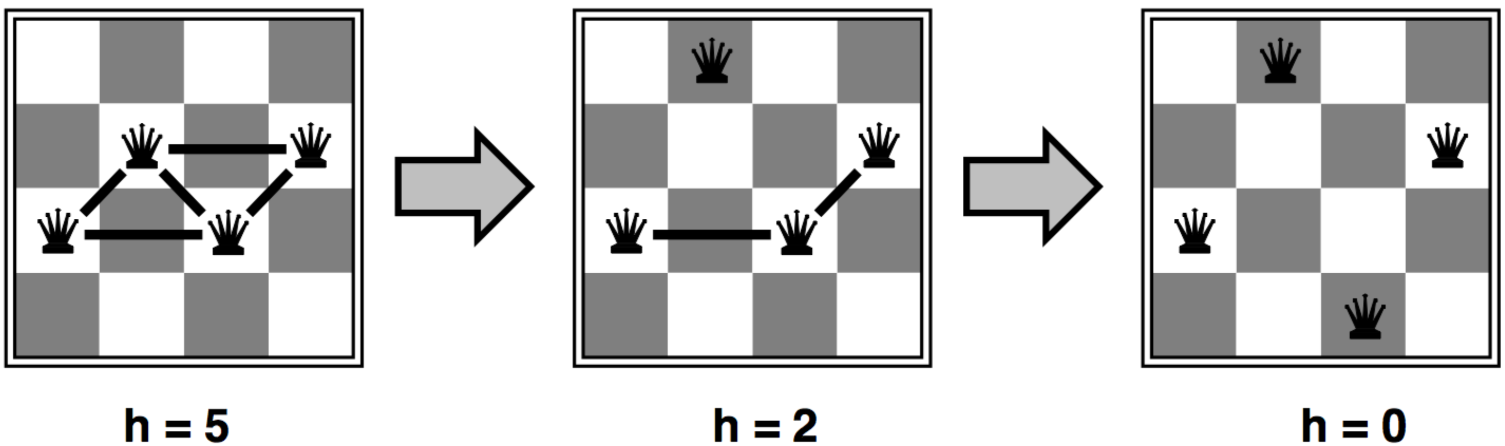
Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- What is state-space?
- What is objective function?



Heuristic for *n*-queens problem

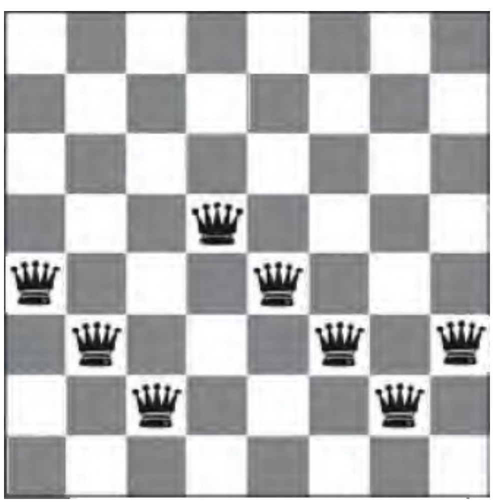
- Goal: n queens on board with no ***conflicts***, i.e., no queen attacking another
- States: n queens on board, one per column
- Successors: move a queen in its column
- Heuristic value function: number of conflicts



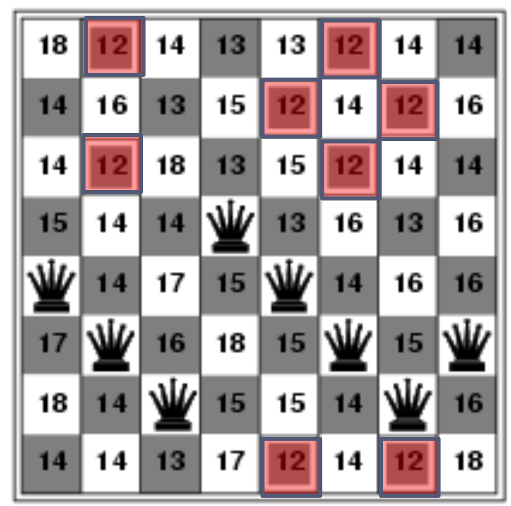
Local search: 8-queens problem

- 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: $h(s) = 0$

$h(s) = 17$

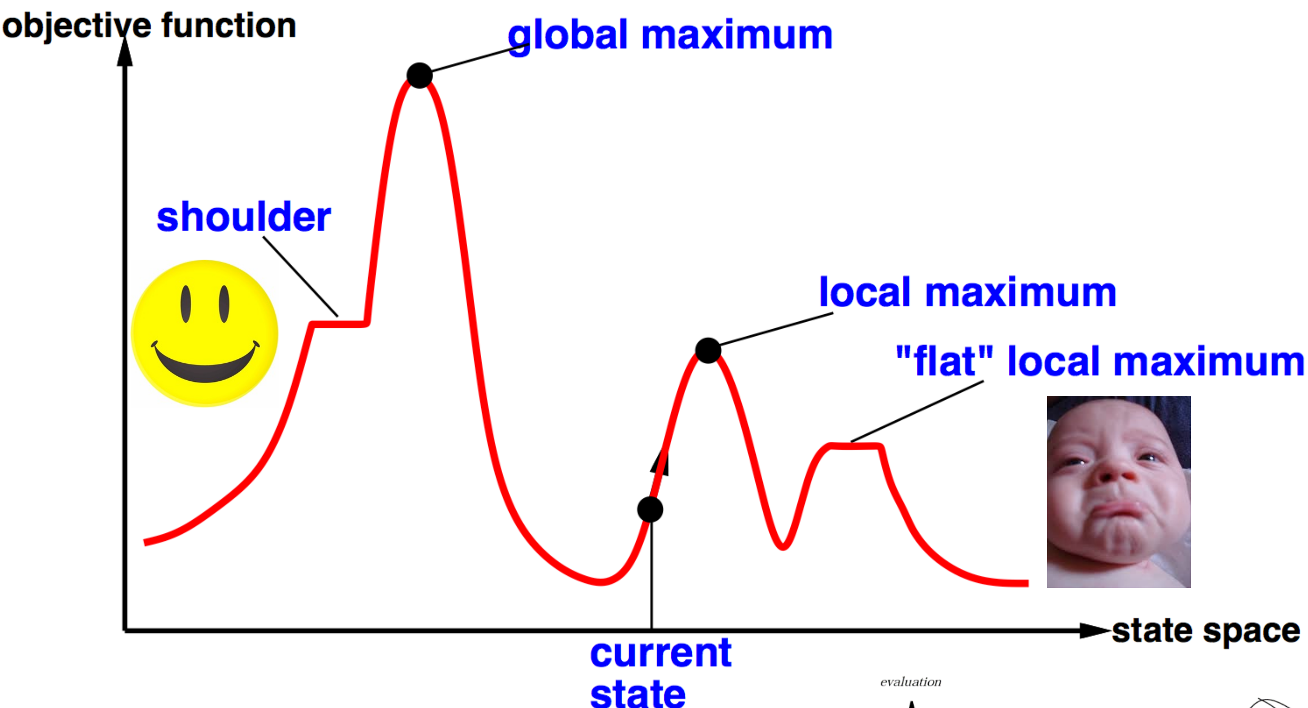


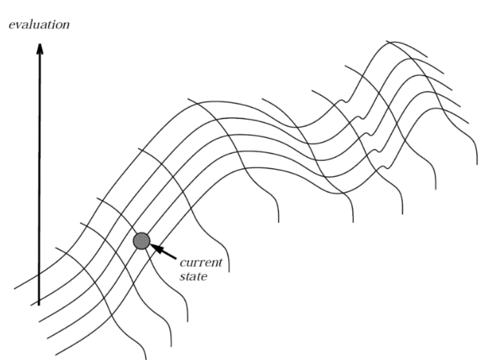
successors objective values



Red: best successors

Global and local maxima





2-d state space

Random restarts

- find global optimum
- duh

Random sideways moves

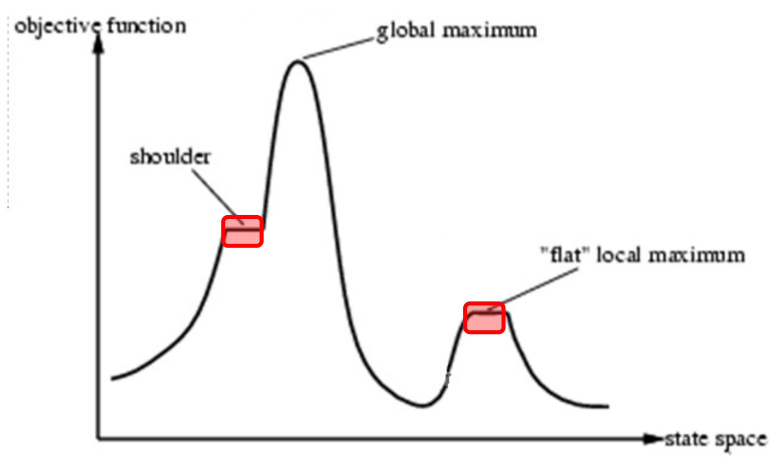
- Escape from shoulders
- Loop forever on flat local maxima

Stochastic hill climbing

- First-choice hill climbing

Sideways move

- 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
- Result on 8-queens:
  - Limit = 100 for consecutive sideways moves
    - 94% success instead of 14% success
      - on average, 21 steps when succeeding and 64 steps when failing



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

Random-restart hill climbing

- All previous versions are incomplete
  - Getting stuck on local max

- **while** state ≠ goal **do**

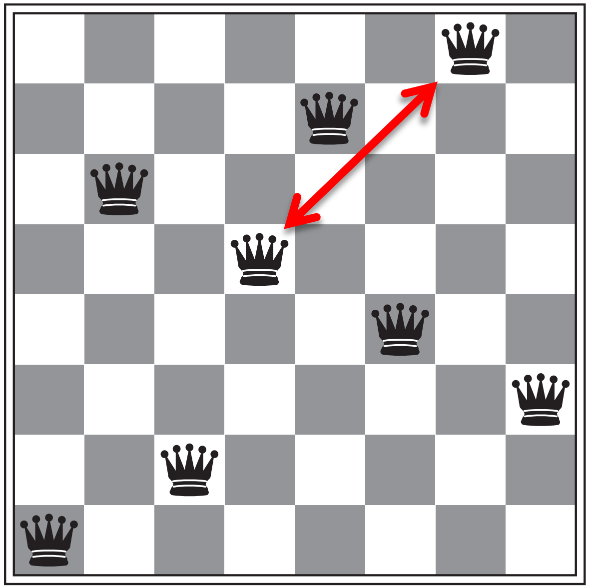
  run hill-climbing search from a random initial state

- 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
  - Although NP-Hard problems typically have an exponential number of local maxima

Hill-climbing on the 8-queens problem

- No sideways moves:
  - Succeeds w/ prob. 0.14
  - Average number of moves per trial:
    - 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

Moral: algorithms with knobs to twiddle are irritating

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 (SA) Search

- **Hill climbing**: move to a better state
  - Efficient, but incomplete (can stuck in local maxima)
- **Random walk**: move to a random successor
  - Asymptotically complete, but extremely inefficient

- Idea: Escape local maxima by allowing some "bad" moves but gradually decrease their frequency.
  - More exploration at start and gradually hill-climbing become more frequently selected strategy

Simulated annealing algorithm

function SIMULATED-ANNEALING(problem,schedule) **returns** a state

current ← problem.initial-state

for t = 1 **to** ∞ **do**

T ←schedule(t)

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

$T(t) = schedule[t]$ is a decreasing series

E(s): objective function

- Pick a random successor of the current state
- If it is better than the current state go to it
- Otherwise, accept the transition with a probability

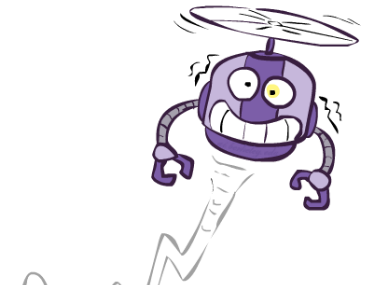
Probability of state transition

A successor of s

$$P(s, s', t) = \alpha \times \begin{cases} 1 & if\ E(s') > E(s) \\ e^{(E(s') - E(s))/T(t)} & o.w. \end{cases}$$

- Probability of "un-optimizing" ($\Delta E = E(s') - E(s) < 0$) random movements depends on <u>badness of move</u> and <u>temperature</u>
  - Badness of movement: worse movements get less probability
  - Temperature
    - High temperature at start: higher probability for bad random moves
    - Gradually reducing temperature: random bad movements become more unlikely and thus hill-climbing moves increase

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 hillclimbing 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

- Keep track of k states
  - Instead of just one in hill-climbing and simulated annealing

Start with k randomly generated states

Loop:

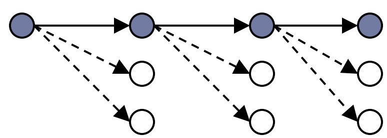
  All the successors of all *k* states are generated

  If any one is a goal state **then** stop

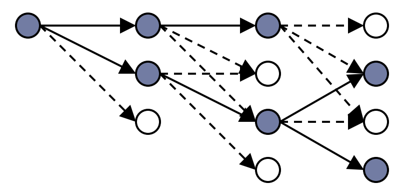
  else select the *k* best successors from the complete list of successors and repeat.

Local Beam Search

- Like greedy hillclimbing search, but keep K states at all times:



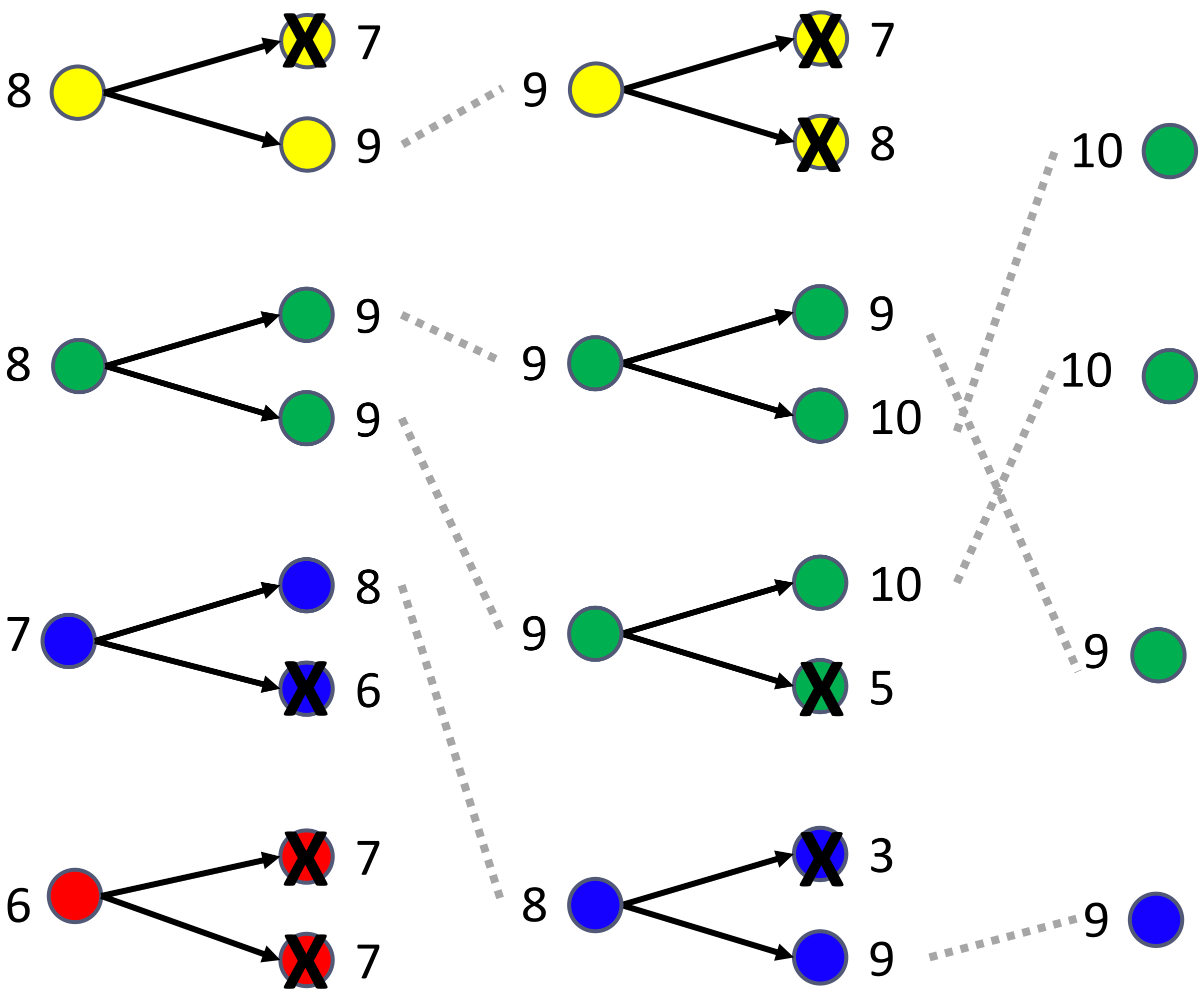
Greedy Search



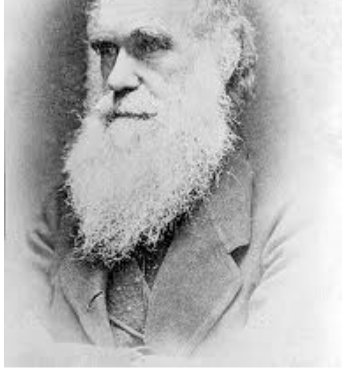
Beam Search

- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- <u>Problem</u>: Concentration in a small region after some iterations
  - <u>Solution</u>: Stochastic beam search
    - Choose k successors at random with probability that is an increasing function of their objective value

Beam search example (K=4)



Local beam search

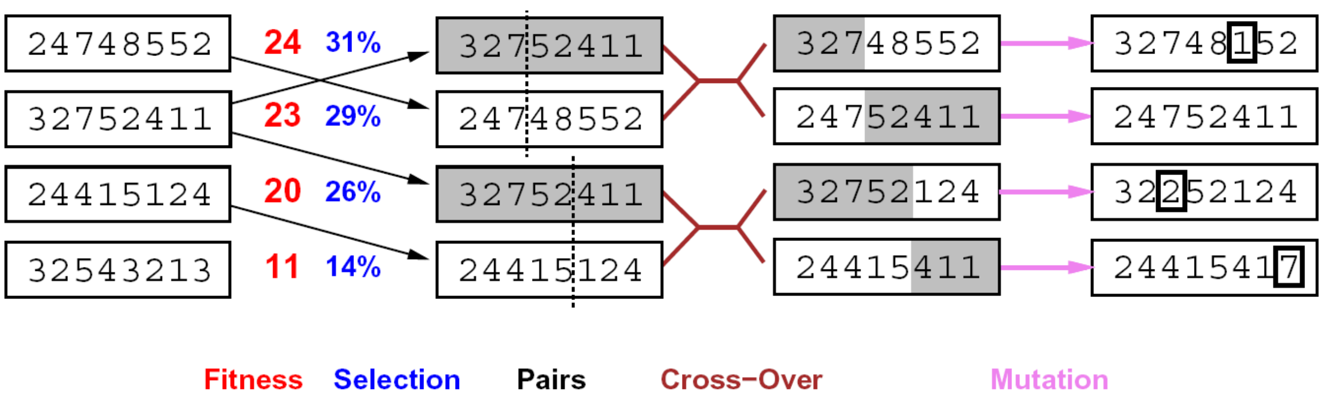


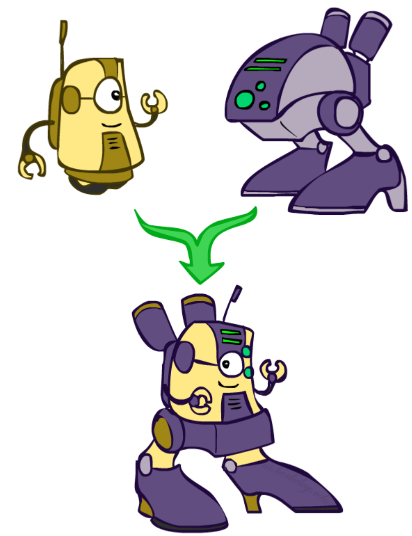
- Basic idea:
  - *K* copies of a local search algorithm, initialized randomly
  - For each iteration
    - Generate ALL successors from *K* current states
    - Choose best *K* of these to be the new current states

Or, K chosen randomly with a bias towards good ones

- Why is this different from *K* local searches in parallel?
  - The searches ***communicate***! "Come over here, the grass is greener!"
- What other well-known algorithm does this remind you of?
  - Evolution!

Genetic Algorithms (GAs)



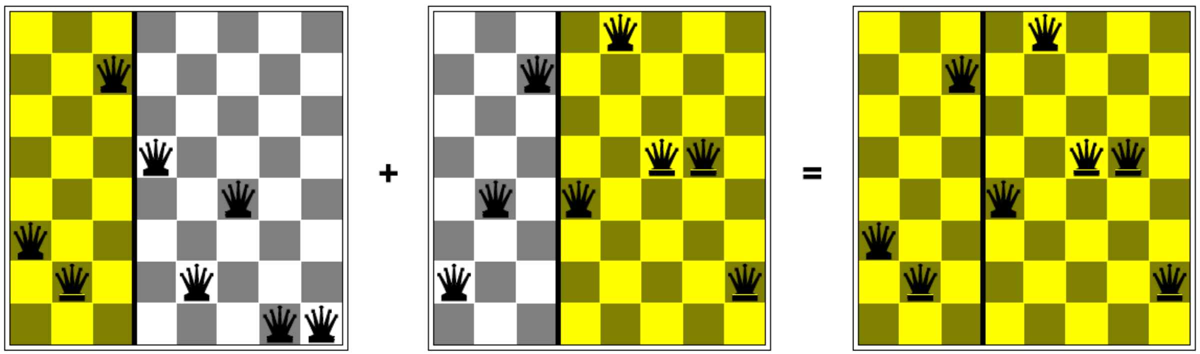


- Genetic algorithms use a natural selection metaphor
  - Resample *K* individuals at each step (selection) weighted by fitness function
  - Combine by pairwise crossover operators, plus mutation to give variety
- A variant of stochastic beam search
  - Successors can be generated by combining two parent states in addition to modifying a single state

Genetic Algorithm (GA)

- A state (solution) is represented as a string over a finite alphabet
  - Like a chromosome containing genes
- Start with *k* randomly generated states (population)
- Evaluation function to evaluate states (fitness function)
  - Higher values for better states
- Combining two parent states and getting offsprings (cross-over)
  - Cross-over point can be selected randomly
- Reproduced states can be slightly modified (mutation)
- The next generation of states is produced by selection (based on fitness function), crossover, and mutation

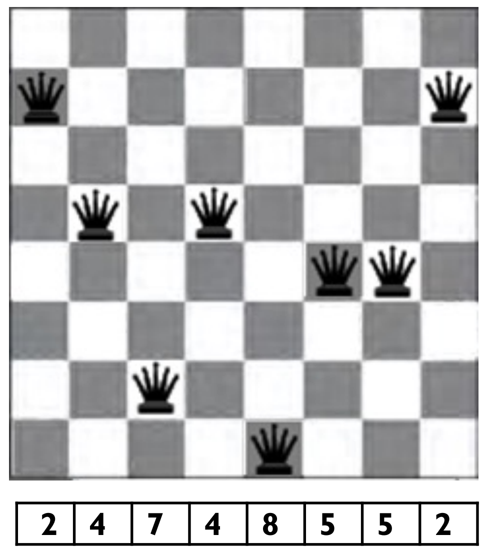
Example: N-Queens



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

Chromosome & fitness: 8-queens

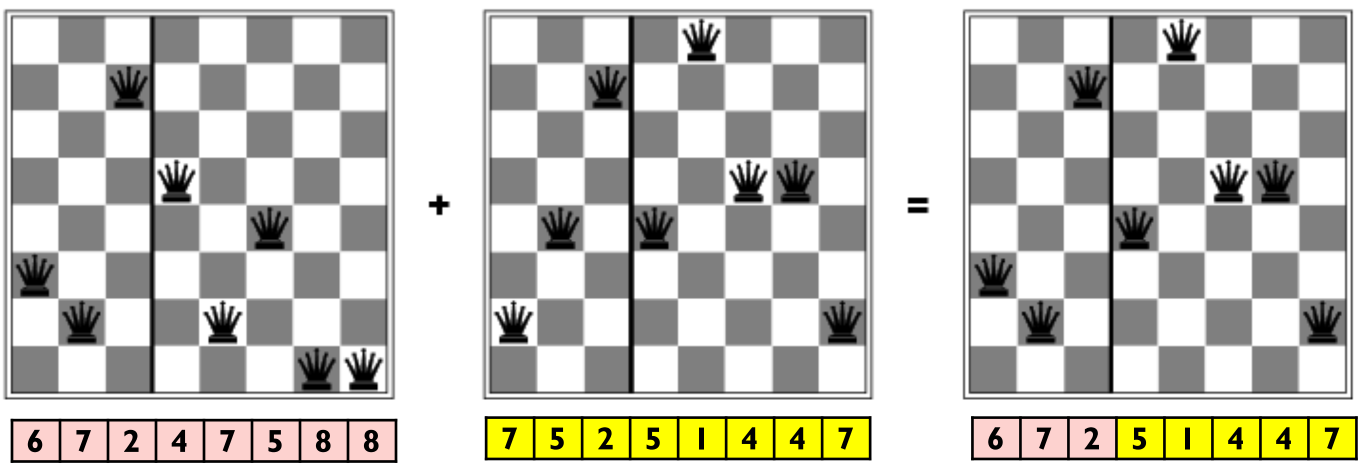
- Describe the individual (or state) as a string



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

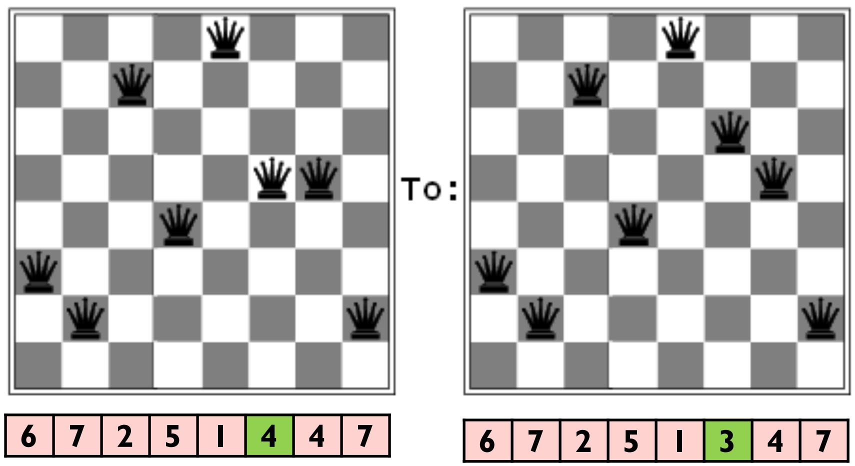
Genetic operators: 8-queens

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

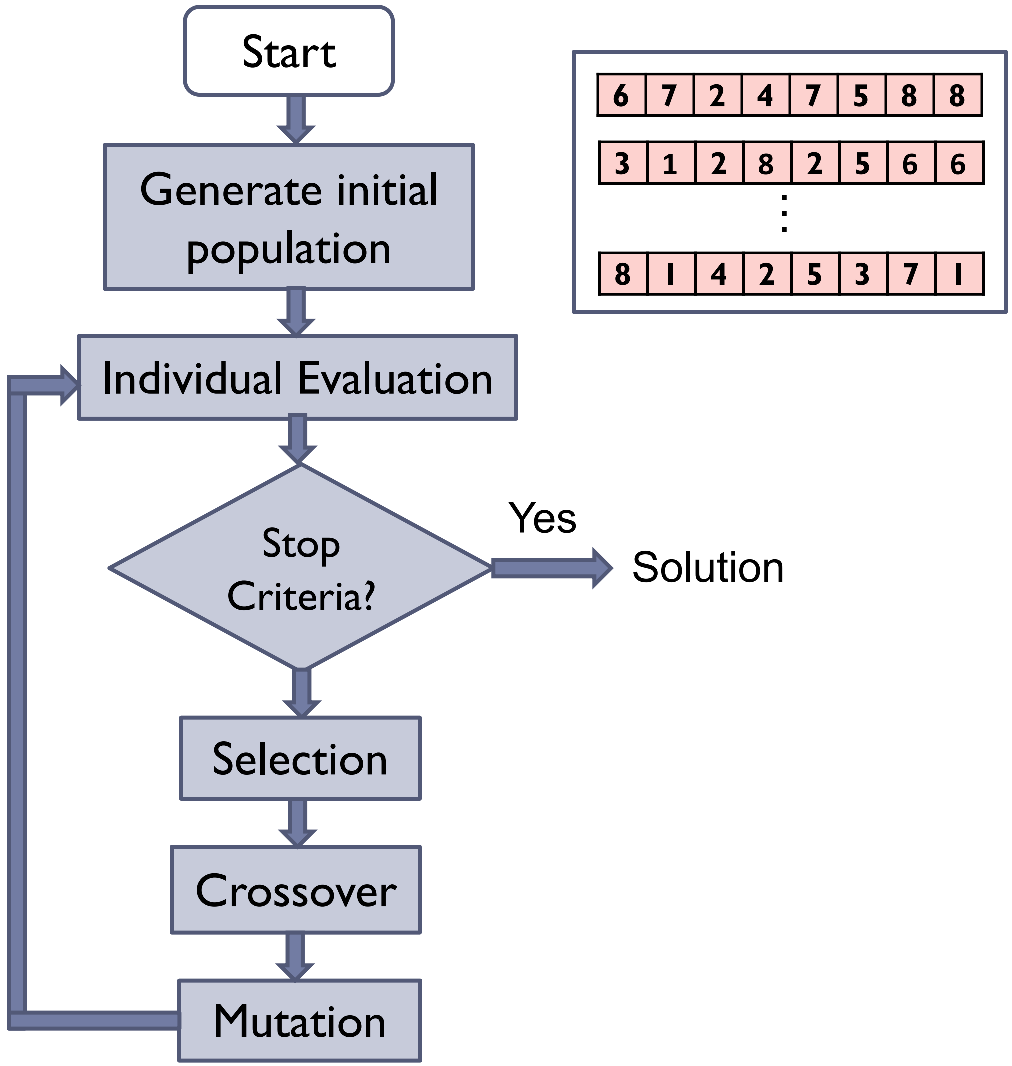


Genetic operators: 8-queens

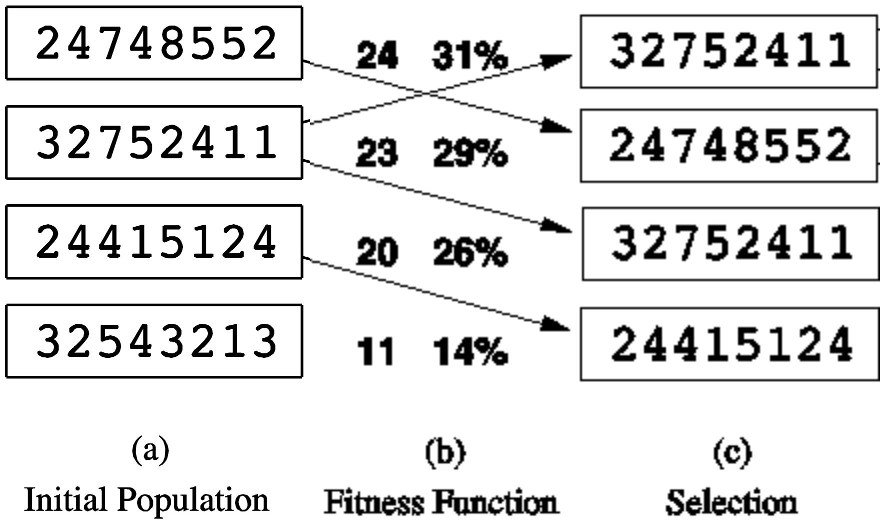
- Mutation: To change a small part of one state with a small probability.



A Genetic algorithm diagram

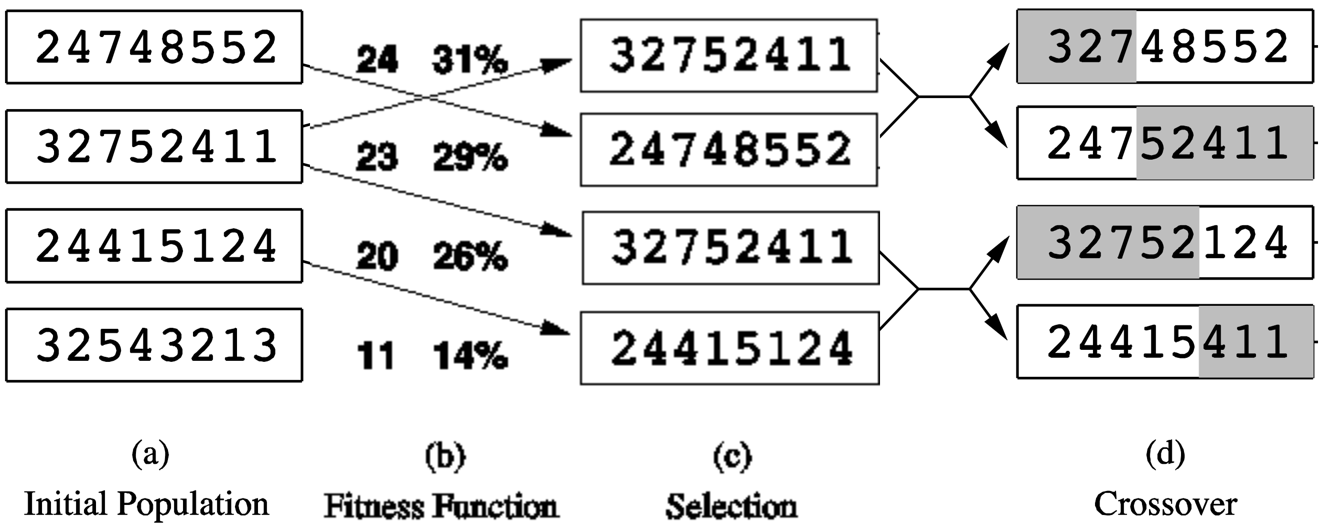


A variant of genetic algorithm: Selection

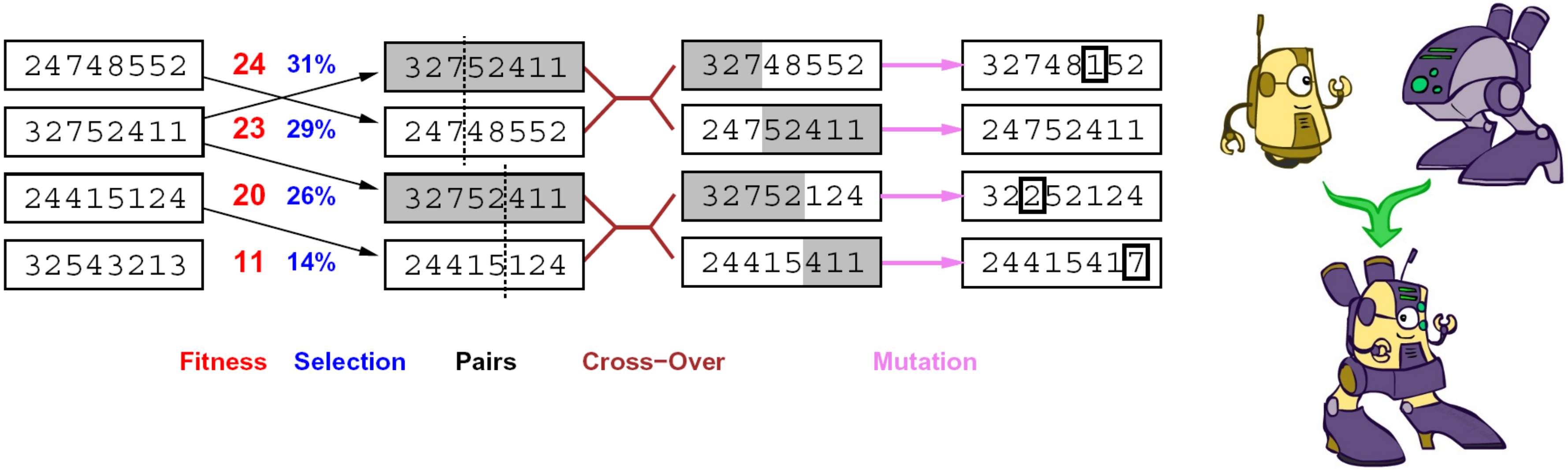


- Fitness function: number of non-attacking pairs of queens
  - min = 0, max = 8 × 7/2 = 28
  - Reproduction rate(*i*) = $fitness(i)/\sum_{k=1}^{n} fitness(k)$
    - e.g., 24/(24+23+20+11) = 31%

A variant of genetic algorithm: Crossover



Genetic Algorithm: Mutation



- Possibly the most misunderstood, misapplied (and even maligned) technique around

Genetic algorithm properties

- 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

- Cross-over as a distinction property of GA
  - Ability to combine large blocks of genes evolved independently
    - Representation has an important role in benefit of incorporating crossover operator in GA

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

Summary

- Many configuration and optimization problems can be formulated as local search
- General families of algorithms:
  - Hill-climbing, continuous optimization
  - Simulated annealing (and other stochastic methods)
  - Local beam search: multiple interaction searches
  - Genetic algorithms: break and recombine states

We will see local search algorithms for continuous spaces

Many machine learning algorithms are local searches