AI

Artificial Intelligence

8.2.6.
Advanced Search (Ch 4) Part 2

Outline

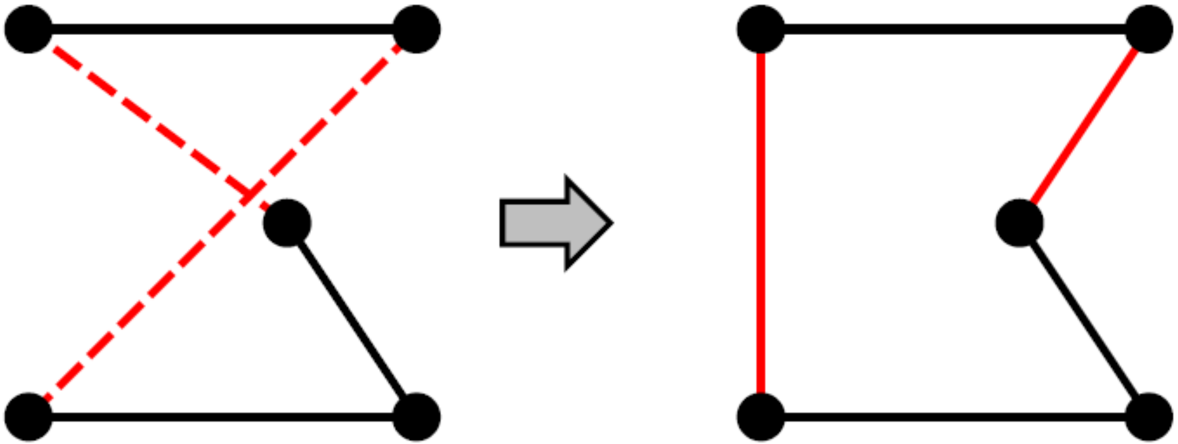
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
- Genetic algorithms

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
  - Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
  - keep a single "current" state, try to improve it

Example: Travelling Salesperson Problem

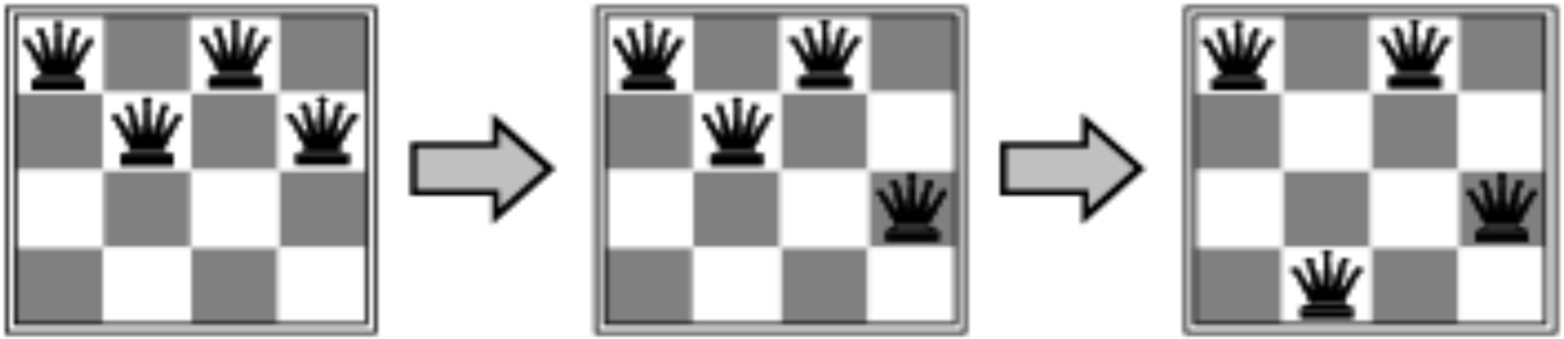
Start with any complete tour, perform pairwise exchanges



Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: *n*-queens

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



Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

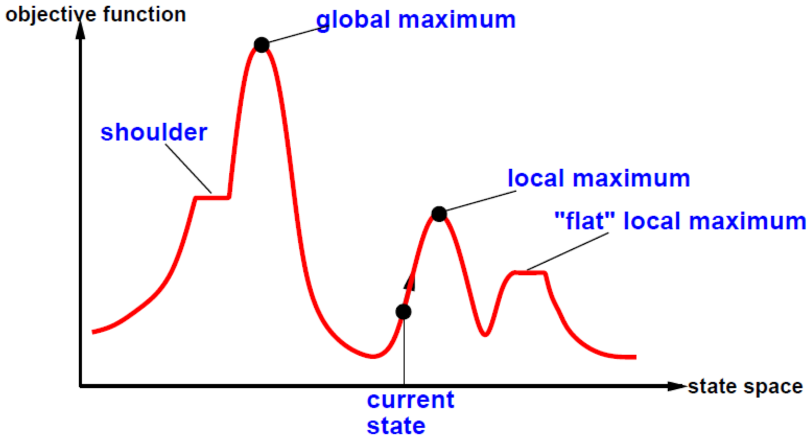
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima

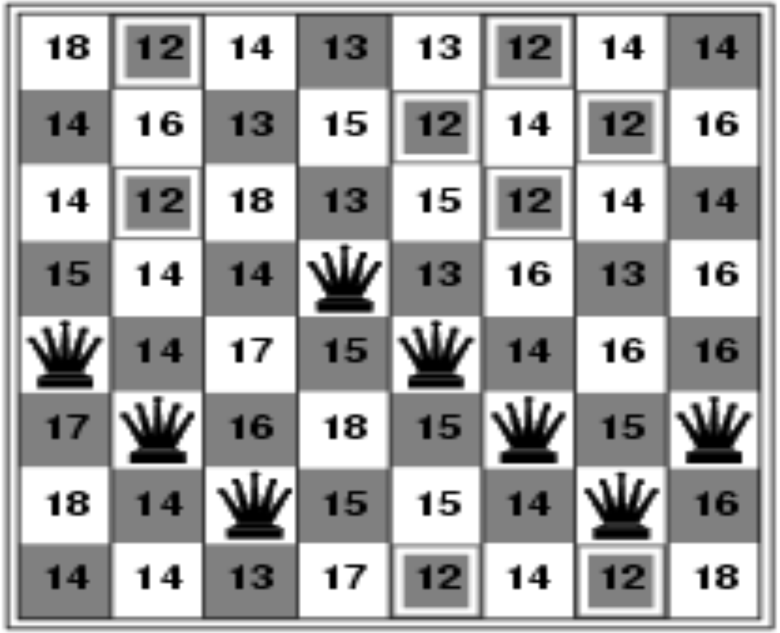
Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

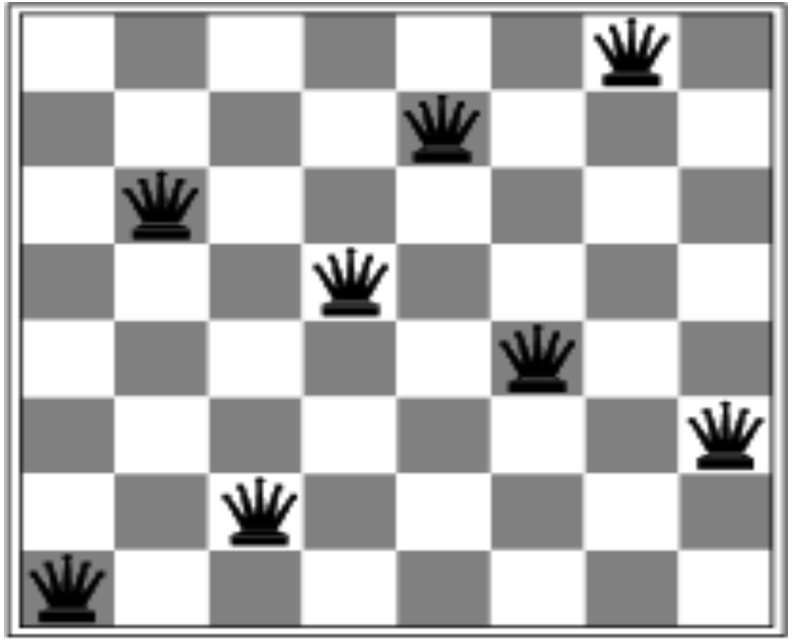
Random sideways moves 😊escape from shoulders ☹loop on flat maxima

Hill-climbing search: 8-queens



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

Hill-climbing search: 8-queens



- A local minimum with $h = 1$

Demos

- http://eightqueen.becher-sundstroem.de/
- http://cliplab.org/~jfran/ptojs/queens_ui/queens_ui.html

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
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 ← 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^(Δ E/T)
```

Properties of simulated annealing search

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^*
because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Local Beam Search

- Keep track of *k* states rather than just one
- *k* is called the beam width
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

Genetic algorithms

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation.

Genetic algorithms contd.

- What is the fitness function?
- How is an individual represented?
  - Using a string over a finite alphabet.
  - Each element of the string is a gene.
- How are individuals selected?
  - Randomly, with probability of selection proportional to fitness
  - Usually, selection is done with replacement.
- How do individuals reproduce?
  - Through crossover and mutation

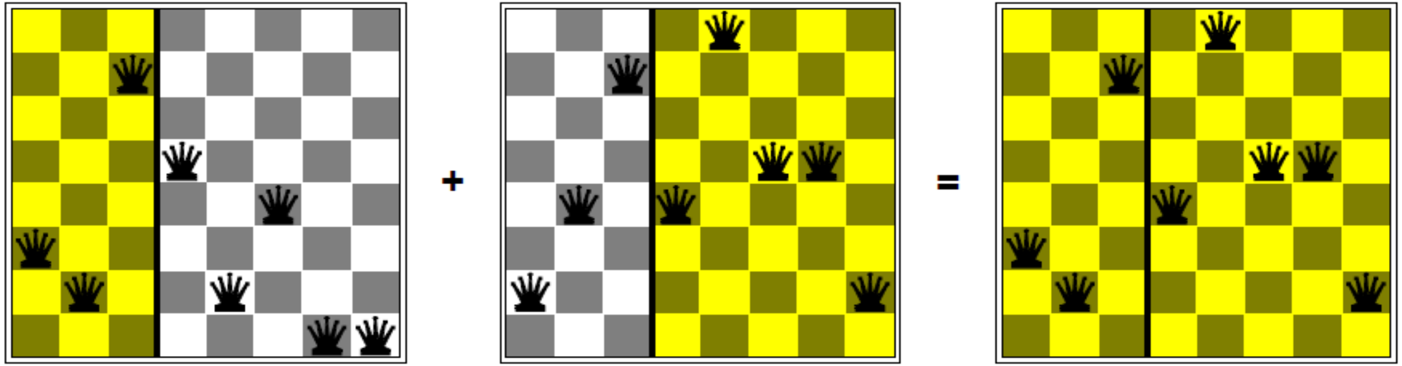
GA Pseudocode

- Choose initial population (usually random)
- Repeat (until terminated)
  - Evaluate each individual's fitness
  - Select pairs to mate
  - Replenish population (next-generation)
    - Apply crossover
    - Apply mutation
  - Check for termination criteria

Genetic Algorithms

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



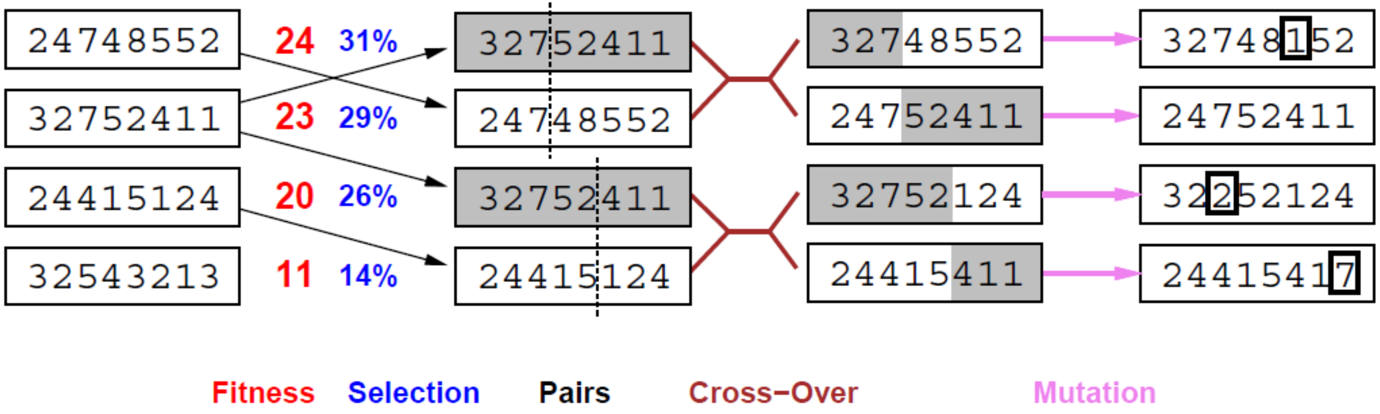
GAs $\neq$ evolution: e.g., real genes encode replication machinery!

3	2	7	5	2	4	1	1	+	2	4	7	4	8	5	5	2	

=	3	2	7	4	8	5	5	2

Genetic algorithms

= stochastic local beam search + generate successors from **pairs** of states



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc.

Replacement

- Simple or generational GAs replace entire population
- Steady state or online GAs use different replacement schemes:
  - Replace worst
  - Replace best
  - Replace parent
  - Replace random
  - Replace most similar

Demo

```
% python ga.py 10 4  | more
```

Initial Population

individual bitarray('010011001000') fitness 4
individual bitarray('000001001000') fitness 2
individual bitarray('001000000000') fitness 2
individual bitarray('000010000011') fitness 4
individual bitarray('010011001001') fitness 3
individual bitarray('011010001001') fitness 2
individual bitarray('001001011010') fitness 3
individual bitarray('000010000011') fitness 4
individual bitarray('011010010001') fitness 3
individual bitarray('010000010000') fitness 4

average 3.1

…

average 3.7

…

average 3.9

…

average 4.2

…

parent_x bitarray('001011000010')
parent_y bitarray('011011000010')
crossover_pt 2
child   bitarray('001011000010')

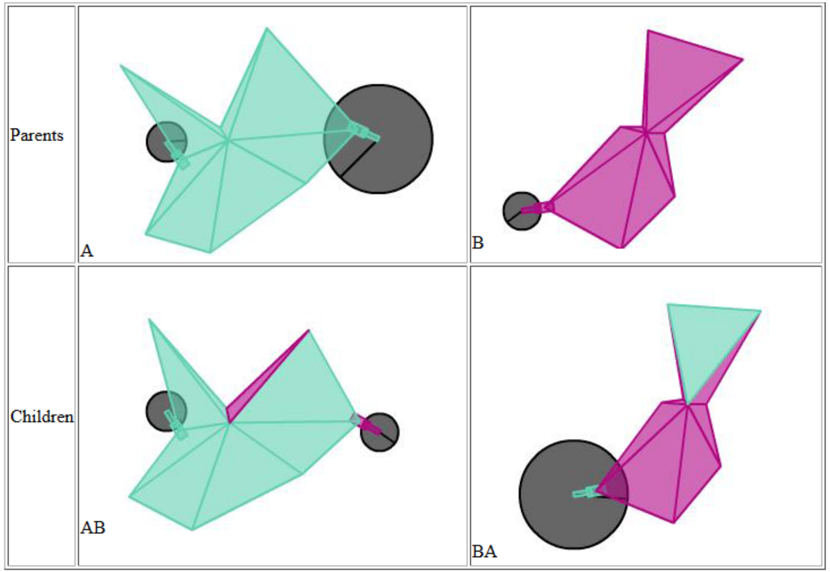
individual bitarray('001011000010') fitness 6

2 4 1 3

Car Design Simulation

- http://www.boxcar2d.com/
- http://www.boxcar2d.com/about.html

Crossover



Here are the associated chromosomes for the cars shown above:

Car	Angle0	Mag0	Angle1	Mag1	...	...	...	...	...	...	...	...	...	...	...	...	WheelVertex0	AxleAngle0	WheelRadius0	WheelVertex1	AxleAngle1	WheelRadius1
A	0.769	2.614	0.584	0.319	0.278	2.883	0.666	1.13	0.305	2.752	0.376	2.507	0.814	1.963	0.392	2.872	3	5.284	0.434	7	2.625	1.191
B	0.535	2.682	0.732	2.256	0.422	0.149	0.676	0.578	0.709	2.774	0.592	2.623	0.519	1.531	0.924	0.404	-1	0.704	0.122	4	0.167	0.409
AB	0.535	2.682	0.584	0.319	0.278	2.883	0.666	1.13	0.305	2.752	0.376	2.507	0.814	1.963	0.392	2.872	3	5.284	0.434	7	2.625	0.409
BA	0.769	2.614	0.732	2.256	0.422	0.149	0.676	0.578	0.709	2.774	0.592	2.623	0.519	1.531	0.924	0.404	-1	0.704	0.122	4	0.167	1.191

Genetic Algorithms Links

- http://math.hws.edu/eck/jsdemo/jsGeneticAlgorithm.html
- http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6
- http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5
- http://www.theprojectspot.com/tutorials/page/2
- http://genetic-algorithms-explained.appspot.com/

AI