CSE 473

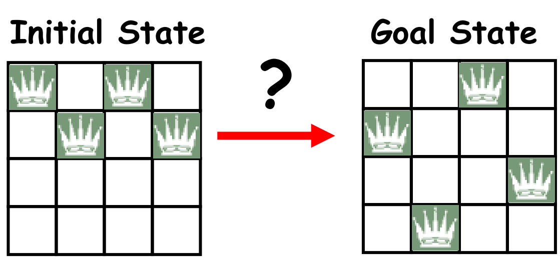
Chaps 4.1 & 5

Local Search and Games

Local search algorithms

- What if *path* to goal is irrelevant? Only interested in *finding* the goal state !

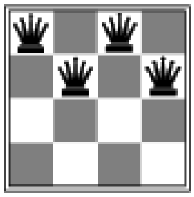
  E.g., N-queens: Put N queens on an $N \times N$ board with no two queens on the same row, column, or diagonal



2

Local Search

Not so good



- Local search algorithms: Keep only a single "current" state and try to improve it
  - Advantage: Very little memory required
  - Also works in infinite (continuous) state spaces

3

Hill-climbing search

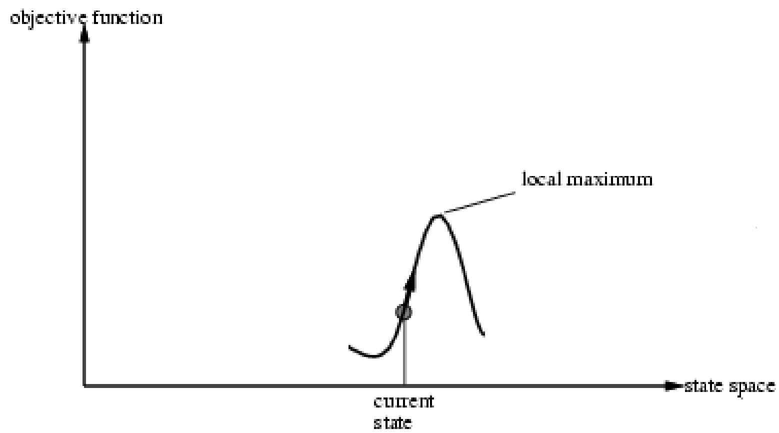
"Like climbing Mt. Rainier in thick fog with amnesia"

```
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
```

4

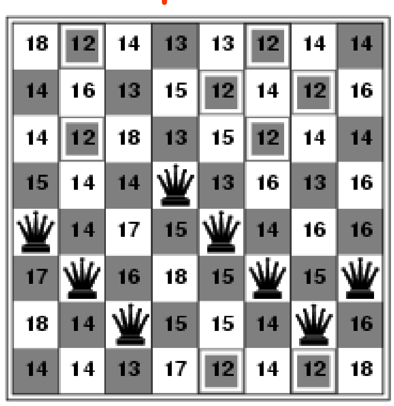
Hill-climbing search



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

Hill Climbing Example: 8-queens problem

Objective function *h*?



Here,

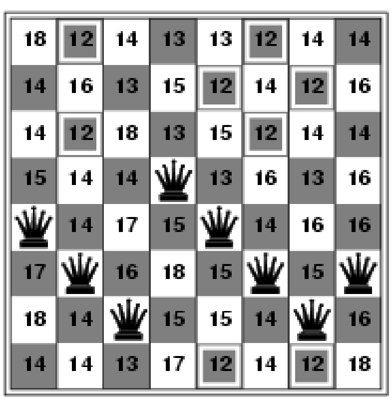
h = 17

Numbers denote h-values for available moves

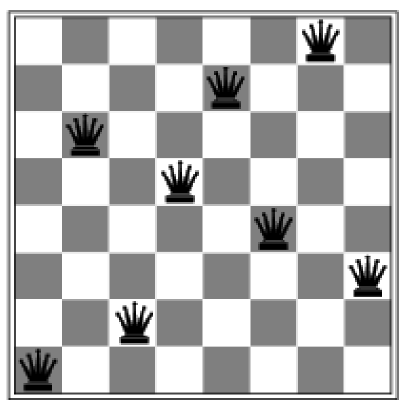
- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- Want to *minimize h*

Queens attacking each other? Most uncivilized. I prefer tea and crumpets.

Example: 8-queens problem



Hill climb (here, hill "descend")



- *A local minimum with $h = 1$.* Need $h = 0$
- *In general, how to find a global minimum or maximum?*

Simulated Annealing

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

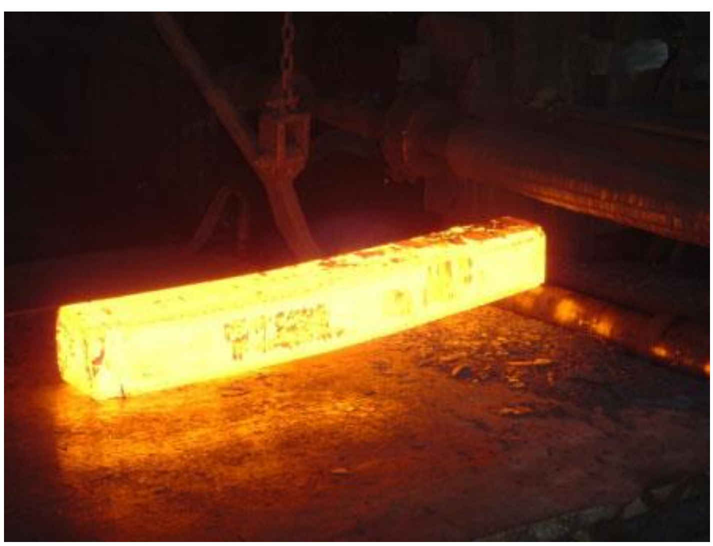
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward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

- Select random *next*
- Move to it for sure if it has higher value
- Otherwise move to it with some probability

9

Why "annealing"?



http://www.kumarsteels.in/process.htm

10

Properties of simulated annealing

- One can prove: If *T* decreases slowly enough, then simulated annealing will find a global optimum with probability approaching 1
- Simulated annealing is widely used for optimizing VLSI layout, airline scheduling, etc.

11

Instead of just one state, what if we keep *multiple states* (as we did in colonial times)?

12

Local Beam Search

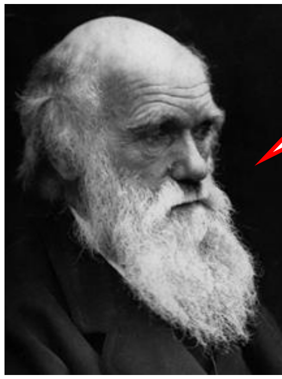
- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, generate all the successors of all *k* states
- If any one is a goal state, stop;
- Else select the *k* best successors from the complete list and repeat.

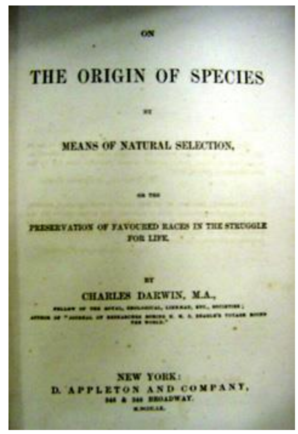
13

Hey, perhaps sex can improve search?

14

Sure, venerable lady
– check out my
yonder book.





ON

THE ORIGIN OF SPECIES

BY

MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

BY

CHARLES DARWIN, M.A.,

NEW YORK:
D. APPLETON AND COMPANY,

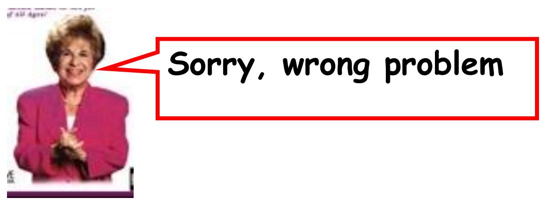
15

Genetic Algorithms

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

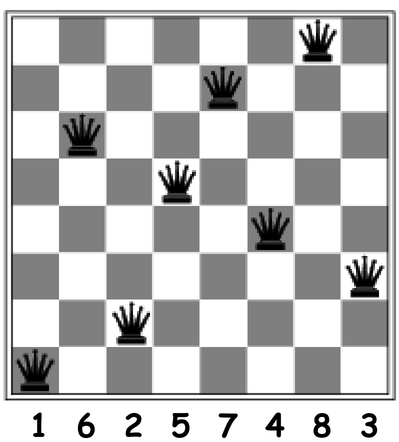
16

Example: Evolving 8 Queens



17

Example: Evolving 8 Queens

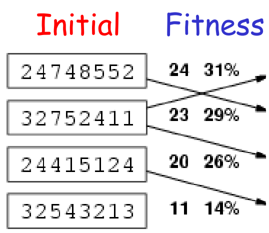


String representation of board (read from left to right column): 16257483

- Need a "fitness function": how "fit" or desirable (i.e., close to the solution) is a string
- Example: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)

18

One iteration of Genetic algorithm



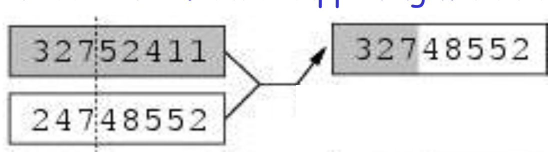
Fitness:

24/(24+23+20+11) = 31% probability of selection for reproduction
23/(24+23+20+11) = 29% etc.

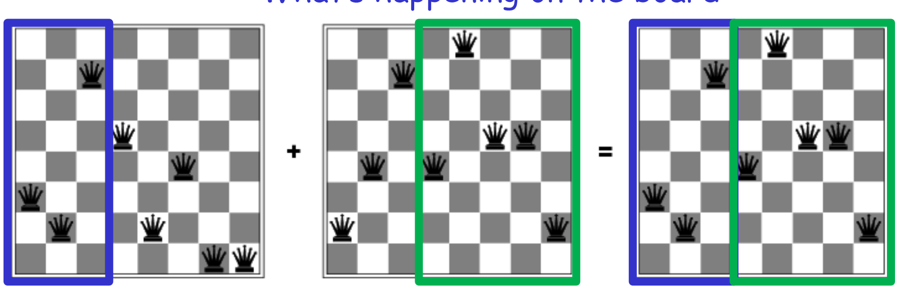
19

Queens crossing over

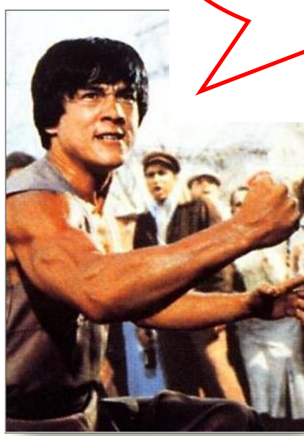
Crossover: What's happening with the strings



What's happening on the board



20



Adversarial Search

- Programs that can play competitive board games
- Minimax search

Games Overview

	deterministic	chance
Perfect Information (fully observable)	chess, checkers, go, othello	backgammon, monopoly
Imperfect Information (partially observable)	battleships	poker, bridge, scrabble

23

Games & Game Theory

- When there is *more than one agent*, the future is not easily predictable anymore for the agent
- In *competitive* environments (conflicting goals), adversarial search becomes necessary
- Class of games well-studied in AI:

  board games, which can be characterized as ***deterministic, turn-taking, two-player, zero-sum*** games with ***perfect information***

24

Games as Search

- Components:
  - States:
  - Initial state:
  - Successor function:
  - Terminal test:
  - Utility function:

Games as Search

- Components:
  - States: board configurations
  - Initial state: the board position and which player will move
  - Successor function: returns list of *(move, state)* pairs, each indicating a legal move and the resulting state
  - Terminal test: determines if the game is over
  - Utility function: gives a numeric value to terminal states (e.g., -1, 0, +1 in chess for loss, tie, win)

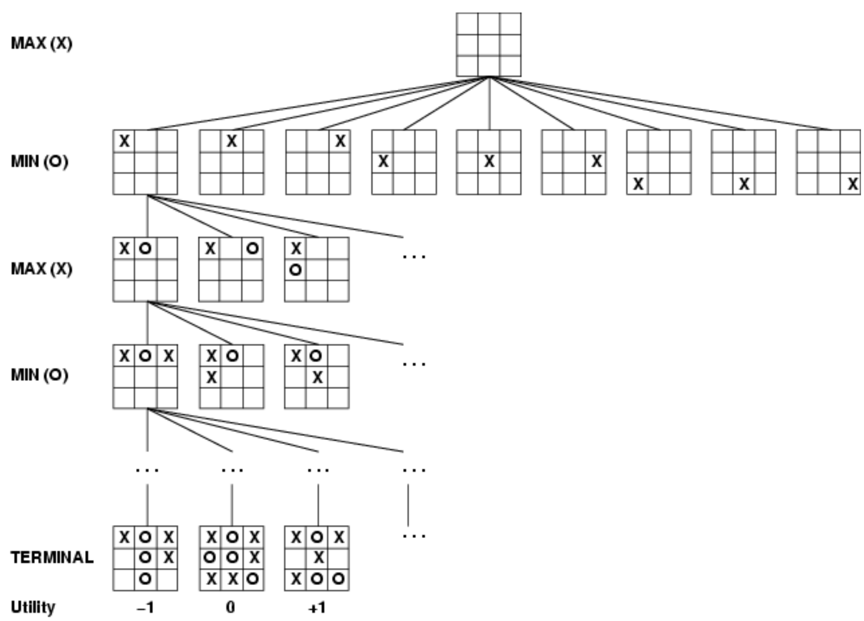
Games as Search

Convention: first player is MAX,
2nd player is MIN

- MAX moves first and they take turns until the game is over
- Winner gets reward, loser gets penalty
- Utility values are from MAX's perspective
- Initial state + legal moves define the *game tree*
- MAX uses game tree to determine next move

27

Game Tree for Tic-Tac-Toe



28

Optimal Strategy: Minimax Search

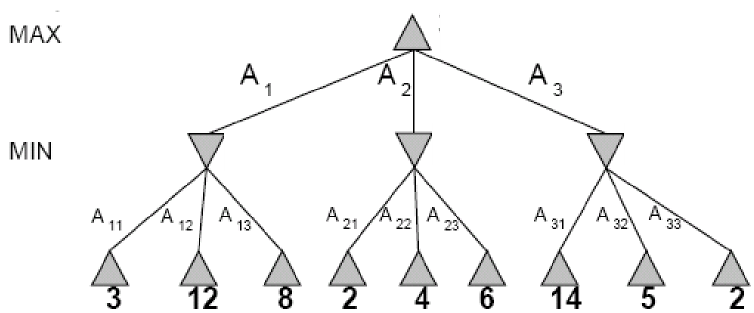
- Find the *best move* for MAX assuming MIN also chooses *its* best move
- Given game tree, optimal strategy determined by computing the *minimax* value of each node:

MINIMAX-VALUE(*n*)=

UTILITY(*n*)	if *n* is a terminal
$\max_{s \in succ(n)}$ MINIMAX-VALUE(*s*)	if *n* is a MAX node
$\min_{s \in succ(n)}$ MINIMAX-VALUE(*s*)	if *n* is a MIN node

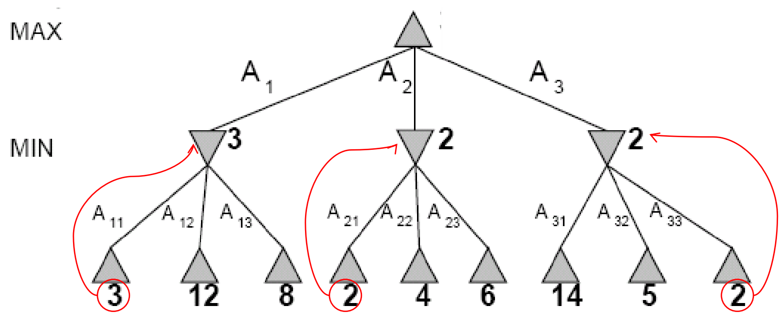
29

Two-Ply Game Tree



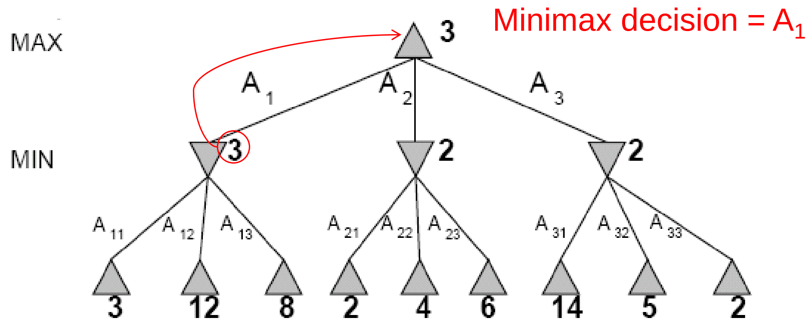
30

Two-Ply Game Tree



31

Two-Ply Game Tree



32

Next Time

- Alpha-beta pruning
- Heuristic evaluation functions
- Rolling the dice

You will find all the heuristic functions you need in my book!