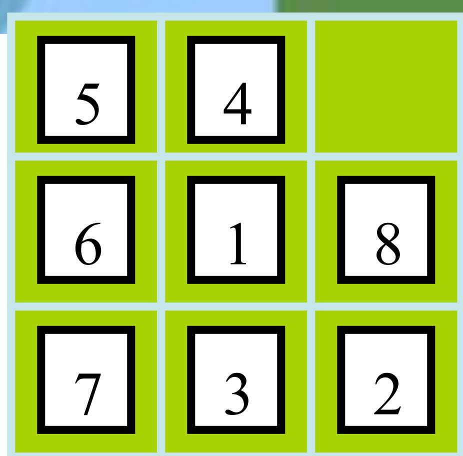
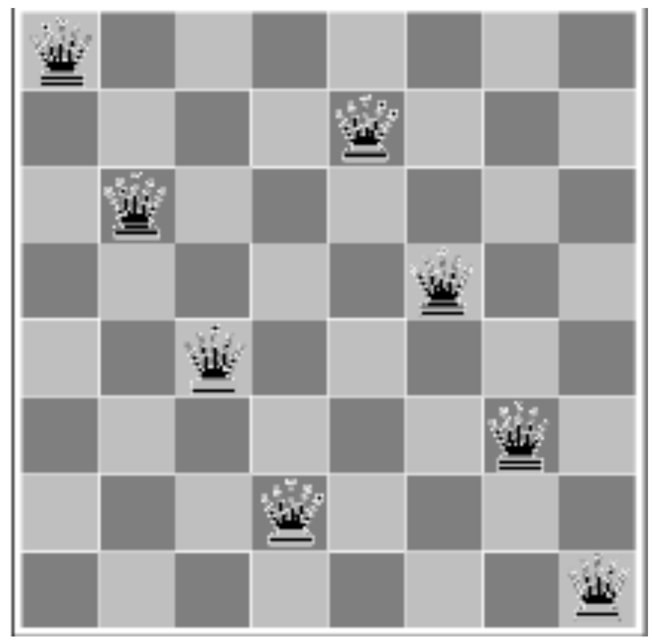
Problem Solving by Search





Readings for this class

- Chapter 3

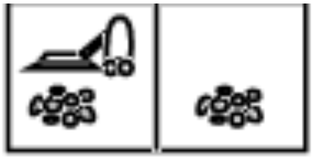
Learning Objectives

- understand how to formulate a problem in AI terms
- review basics of blind search methods
- recognize benefits of iterative deepening
- know how to design heuristics and apply them in A* search

Problem Formulation

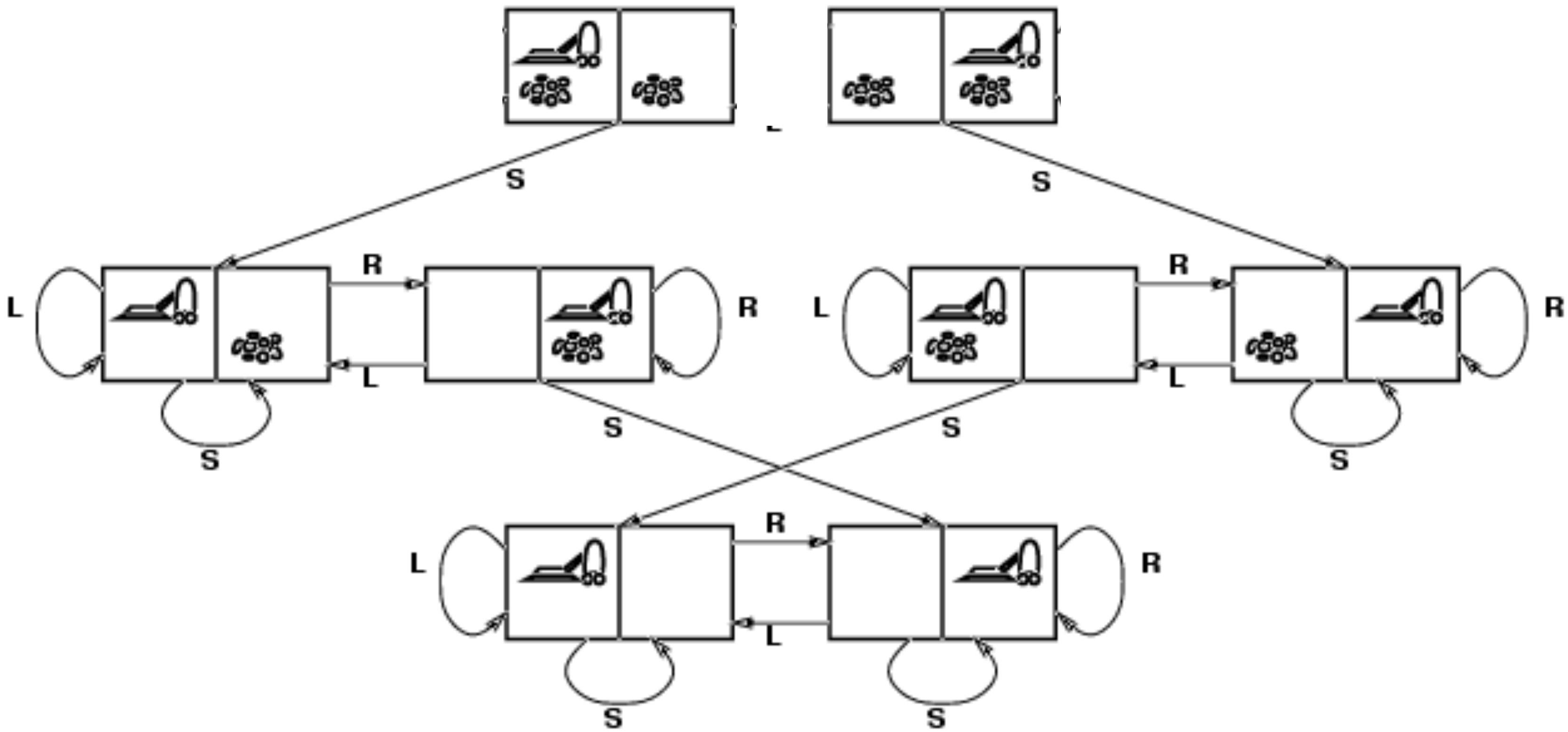
- states: description of “world of interest”
- initial state
- successor function: generates set of legal next states from available actions
- goal test: how do we know we’re done?
- path cost: way of choosing between multiple solutions (e.g., shortest route)

Vacuum World Problem



- successor function:
  - move left (L), move right (R), suck (S)
- goal test:
  - no dirt left in any square
- path cost:
  - each step costs 1

State Tree for the Vacuum World



Missionaries & Cannibals

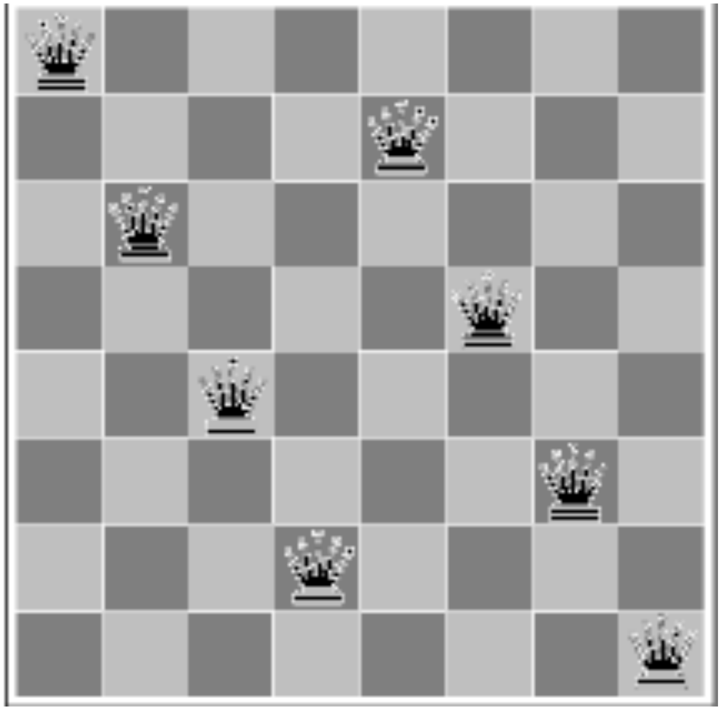
- 3 missionaries and 3 cannibals need to cross crocodile-infested river
- boat can hold 1 or 2 people
- can’t leave any missionaries outnumbered by cannibals

Missionaries & Cannibals: Formulation

- states: (# missionaries, # cannibals, # of boats) on left bank of river
- initial state: (3,3,1)
- successor function: move (# missionaries, # cannibals) from one bank to other
- goal test: (0,0,0)
- path cost: # of river crossings

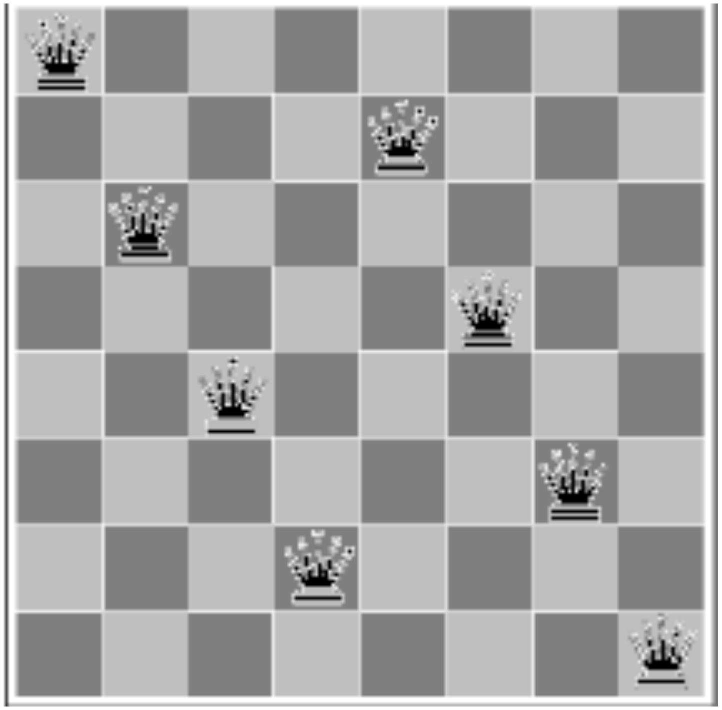
8-Queens problem

- arrange 8 queens on a chessboard so that no two queens are on the same row, column or diagonal (i.e., attack each other)
- applications to parallel memory storage, VLSI testing, traffic control, and deadlock prevention



Naïve approach

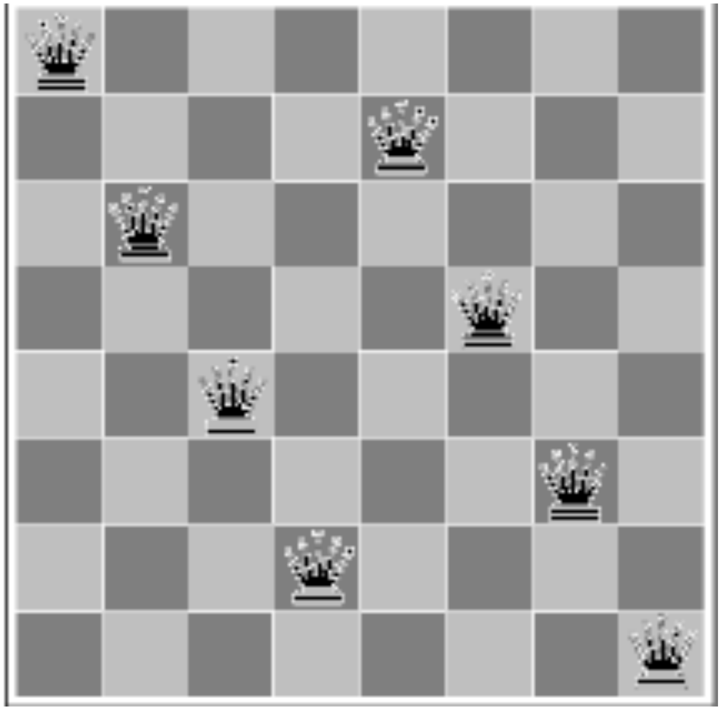
- state: any arrangement of [0,8] queens on the board
- successor function: add a queen to any empty square



State space: 3×10^{14}

Better approach

- state: any arrangement of *n=[0,8]* queens, one per column in the leftmost *n* columns, with no queen attacking another
- successor function: add a queen to any square in the leftmost empty column such that it is not attacked by any other queen



State space: 2057

Search Methods

- use to explore state space for solution to a problem
- can be uninformed (blind) or use some reasonable knowledge (heuristics) to guide search

Uninformed Search

- breadth-first
  - expand shallowest nodes first (FIFO)
- depth-first
  - expand deepest nodes first (LIFO)
- depth-limited search
  - depth-first with cutoff
- iterative-deepening
  - combines benefits of BFS and DFS
- bidirectional
  - applicable when operators are reversible

Breadth-first search

- Expand ***shallowest*** unexpanded node
- Put successors at end of FIFO queue

Breadth-first search

Complete?	Yes (if b is finite)
Time complexity	$1+b+b^2+b^3+\ldots +b^d = O(b^d)$
Space complexity	$O(b^d)$ (every node kept in memory)
Optimal?	Yes (if cost = 1 per step)

b: maximum branching factor of search tree
d: depth of the least cost solution

Exponential time/memory requirements make breadth-first search unsuitable for large problems

Depth-first search

- Expand ***deepest*** unexpanded node
- Put successors at end of LIFO queue (or push on stack)

Depth-first search

Complete?	No (fails in infinite-depth spaces or spaces with loops)
Time complexity	$O(b^m)$ (bad if m >> d)
Space complexity	O(bm) (linear in space)
Optimal?	No

b: maximum branching factor of search tree
d: depth of the least cost solution
m: maximum depth of state space

How to get best of both worlds?

- i.e., how to combine completeness of breadth first & space complexity of depth-first search?
- start with depth-limited search
  - solves the infinite depth problem

Depth-limited search

- depth-first search with depth limit ℓ

Complete?	only if $\ell > d$
Time complexity	$O(b^{\ell})$
Space complexity	$O(b\ell)$
Optimal?	only if $\ell = d$

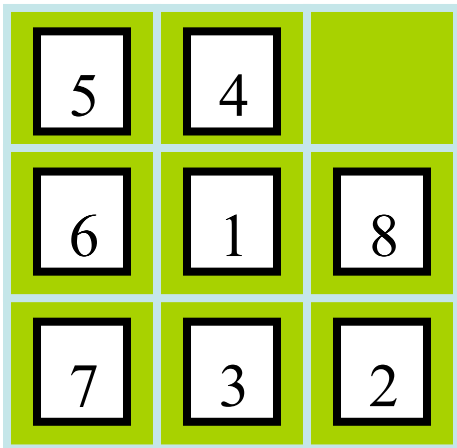
Iterative-deepening search

- use depth-limited search as subroutine with increasing ℓ
- is this efficient?

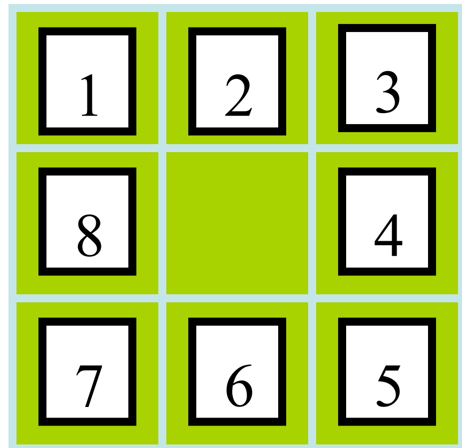
Complete?	Yes
Time complexity	$d+(d-1)b+(d-2)b^2+\ldots+b^d = O(b^d)$
Space complexity	$O(bd)$
Optimal?	Yes (if cost = 1 per step)

8-squares problem

What's a good state description and successor function?



initial state



goal state

Informed Search: Greedy Search

- minimize estimated cost to goal, h(n)
- start by expanding minimal cost node

Informed Search: A* search

- minimize estimated cost to goal: $f(n) = g(n) + h(n)$
  - $g(n)$ = cost of solution from start to *n*
  - $h(n)$ is estimated cost of cheapest solution from *n* to goal
- A* uses a best-first search: chooses least-cost path from initial state to goal state

Definitions

- h(n) is **admissible** or **valid** if it never over-estimates true cost to reach goal
- h(n) is **consistent** or **monotonic** if f(n) never increases as one follows a path from a node through its successors, toward the goal
- a consistent heuristic is also admissible

Optimality of A* search

- If h(n) is admissible, A* is optimal:
  - no optimal algorithm employing the same heuristic will expand fewer nodes than A*

ECE Linux machines

- general purpose Linux machines
  - tr5130gu-<#>.ece.mcgill.ca
    where <#> in (1..15)
- 3 Debian machines in front of TR 5107
  - tr5130oa-0<#>
    where <#> in (1..3)
- simple AI installed here:
  `/opt/linux64/simpleai`

Hello World Simple AI Exercise – Part 1

- the source file can be found under `samples/search/hello_world.py`
- modify the code to test BFS and DFS
- how do these other search techniques perform? why?

Hello World Simple AI Exercise – Part 2

- Change the program to use 3 actions:
  - Insertion: can insert a single character anywhere in the string
  - Deletion: can remove any single character from the string
  - Replace: can replace any character in the string with another character

Hello World Simple AI Exercise – Part 3

- Run the modified code to search for "Hello World!" starting from "halo, word."
- What is the solution given by A*?
- Is the heuristic still admissible?

Homework

- 8-Queens simple AI exercise:
  - implement the 8-Queens problem in simple AI (start from missioners.py)
- read **before next class**:
  - Ch. 5-5.4