Lecture 2

Uninformed search

Reference: Preparing for week 1

- Reading:
  - Chapters 1 and 2.1, 2.2, 2.5, 3.1, 3.2, 3.3, 3.4, 3.5
- Assignment 1 has now been posted on the course LEARN site
  - Uses MATLAB (a tutorial is included)
  - Companion a1-student-stuff.zip file

Academic AI versus "Game AI"

- Academic AI is concerned with optimal performance.
- Game AI has been more about creating a compelling experience for the player, giving *illusion* of intelligence.
- In the past, techniques used for Game AI tended to differ from Academic AI. Game designers concerned with real-time constraints, tight schedules, fast advances, …
- But now Game AI is moving into serious games: health, business, education, …
- How will Game AI evolve?
- Course theme: Enormous potential for improved game AI.

.

Outline: Search topics

- Different search algorithms
  - review of breadth-first, depth-first = uninformed (“brute-force”) search algorithms
  - informed (“heuristic”) search
  - backtracking search for constraint satisfaction problems (CSP)
  - local search
- There are also different forms of representations
  - variable-based / Constraint Satisfaction Problem representations
  - predicates / STRIPS-rules representations

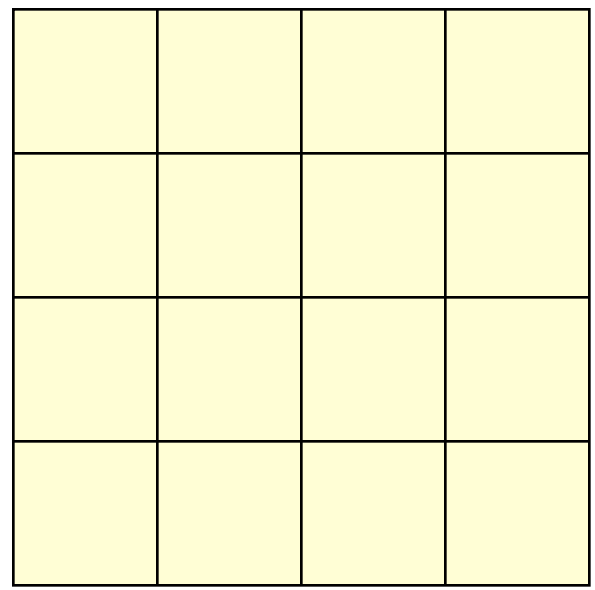
Definition: Problem-solving agent

Artificial intelligence as problem-solving in a search space:

Goal-based agents decide what to do by finding sequences of actions that lead to desirable states.

Example: *n*-queens

Place *n*-queens on an $n \times n$ board so that no pair of queens attacks each other.



Example: Sliding puzzles

Initial configuration

2		3
1	8	4
7	6	5

Goal configuration

1	2	3
8		4
7	6	5

Example: River crossing puzzle

A father, his two sons, and a boat are on one side of a river. The capacity of boat is 100 kg. The father weighs 100 kg and each son weighs 50 kg. How can they get across the river?

Example: Propositional satisfiability

Given a formula in propositional logic, determine if the Boolean variables can be assigned in such a way as to make the formula true.

$(\neg A \lor B) \land$

$(\neg B \lor \neg C \lor D) \land$

$(\neg D \lor G \lor \neg E) \land$

$(\neg D \lor G \lor \neg F) \land$

$A \land$

$C \land$

$\neg E$

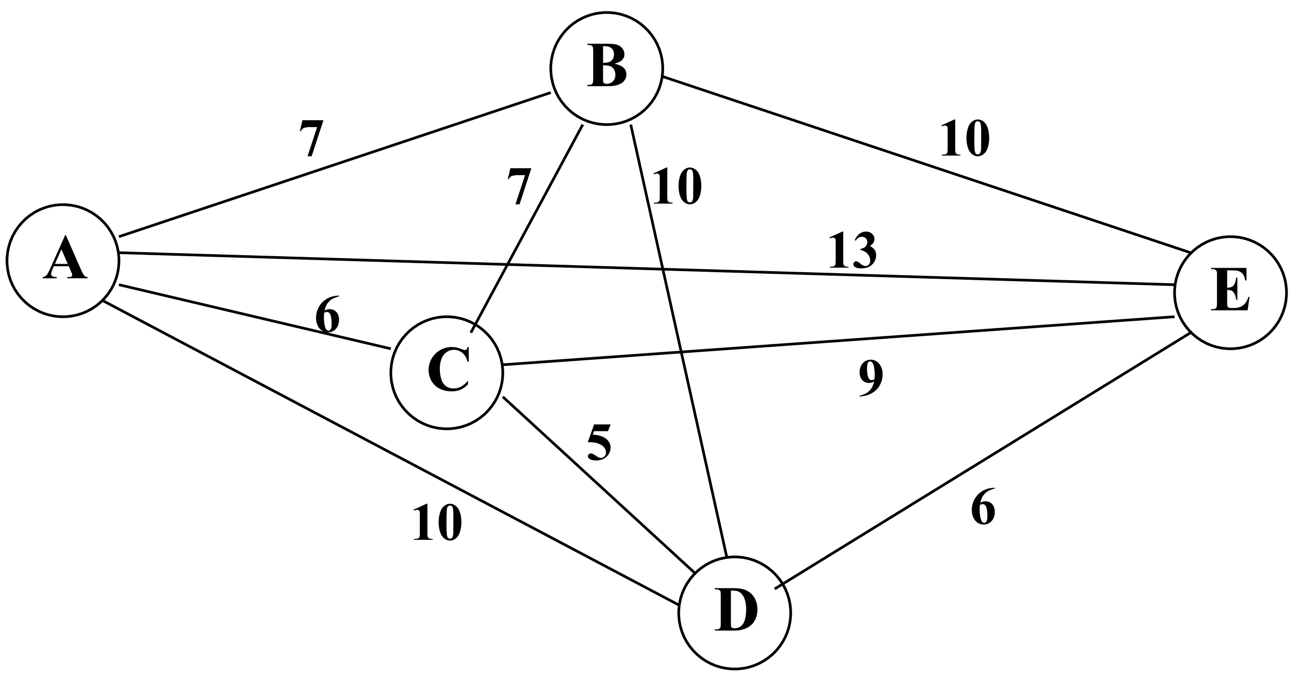
Example: Partition problem

Given a set of objects with weights, partition the objects into two sets U and V such that the total weights of U and V are as close as possible.

Object	a	b	c	d	e	f	g	h
Weight	5	7	10	10	11	15	16	16

Example: Travelling saleswoman problem

Starting at city A, find a route of minimal distance that visits each of the cities only once and returns to A.



Example: Set covering problem

Find a minimum size committee of people that together have the skills necessary to accomplish a task.

SkillsNeeded = {a, b, c, d, e, f, g, h, i, j, k, l}

People = $\{p_1, p_2, p_3, p_4, p_5, p_6\}$, where

p_1 has skills {a, b, e, f, i, j}

p_2 has skills {f, g, j, k}

p_3 has skills {a, b, c, d}

p_4 has skills {c, e, f, g, h}

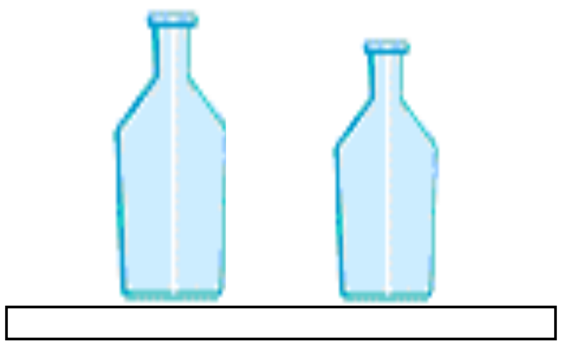
p_5 has skills {i, j, k, l}

p_6 has skills {d, h}

Example: Water jug problem

We are given two jugs: a 4 liter jug and a 3 liter jug. Neither has any measuring markers on it. There is a tap that can be used to fill the jugs with water. How can we get exactly 2 liters of water into the 4 liter jug?

- SOLUTION:
- Fill 3L jug
- Transfer all water in 3L jug into 4L jug
- Fill 3L jug
- Transfer water from 3L jug to 4L jug
  until 4L is full – now 2L water left in 3L jug
- Empty 4L jug onto ground
- Empty 3L jug into 4L jug
- There is now 2L water in 4L jug



Contrasts in problem types

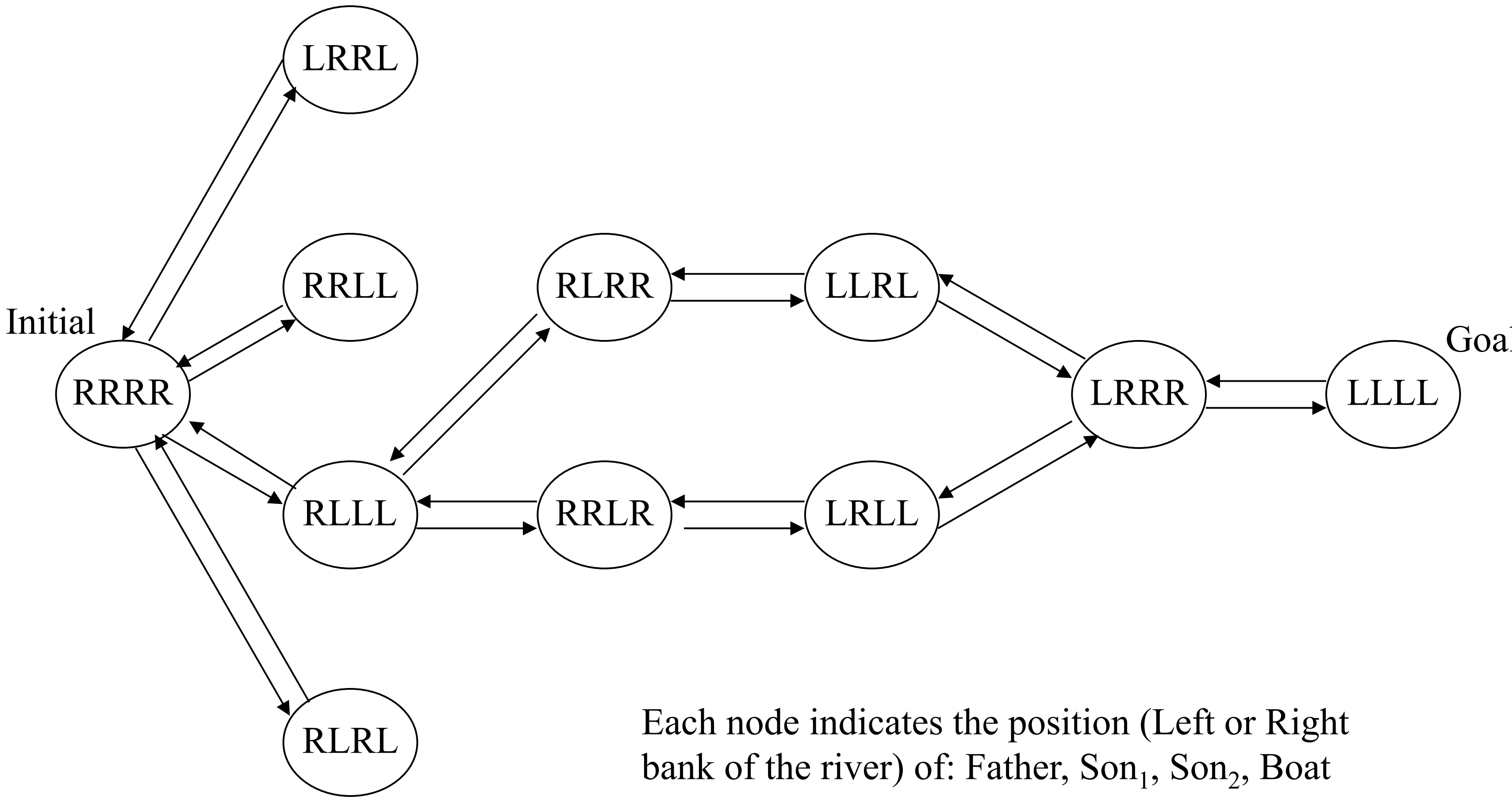
- Find a goal state, given **constraints** on the goal, not interested in sequence of actions
  - Any goal state
    - e.g., *n*-queens, crossword puzzles
  - *Optimal* goal state
    - e.g., traveling saleswoman problem, set covering problem
- Find a sequence of actions that lead to goal state
  - Any sequence
    - e.g., sliding puzzle, river crossing puzzle, water jug problem
  - *Optimal* sequence
    - e.g., sliding puzzle, …

Methodology

- Formulate problem solving as search on a graph
- Given a problem to solve:

1. **Create a Set of Nodes in a Graph:**
   - Specify representation of problem as a graph of nodes (states)
   - Specify initial and goal states ('distinguished' states)
2. **Define Arcs in Graph as Rules/Operators:**
   - Specify rules or operators (arcs) to move current representation of problem from one state to another
   - Also specify cost of each rule/operator
3. **Search to Solve Problem:**
   - Find a path in the graph from the initial state to a goal state

Search graph for River Crossing Puzzle



Each node indicates the position (Left or Right bank of the river) of: Father, Son_1, Son_2, Boat

General search algorithm

```
L ← [start nodes]
while L ≠ empty do
    select and remove a node from L, call it p
    if p is a goal node, return(success)
    generate all successor states of p, and add them to L
end while
return(fail)
```

FIFO queue gives Breadth-First Search (BFS)
LIFO queue gives Depth-First Search (DFS)
Priority queue gives informed search (greedy, A*)

What to do about repeated states?

0. Nothing
1. Don’t return to a state that you just came from
2. Do not create paths with cycles in them (look at ancestors of a node)
3. Do not generate any state that was ever generated before (keep a closed list using a hash table)

Uninformed search

- Uninformed, or brute-force, search uses no knowledge about a particular problem.
- Works the same for all problems.
- Examples: Breadth-first search, depth-first search.

Example: Breadth-first search on the 8-puzzle

Example: Depth-first search on the 8-puzzle

Reference: Breadth-first versus depth-first search

- Complete? (guaranteed to find a solution)
  - BFS: yes
  - DFS: no (graph may have infinite branches)
- Optimal? (guaranteed to find solution at least depth?)
  - BFS: yes (will find shortest)
  - DFS: no (may find leftmost, but not shortest)
- b = branching factor, d = depth of solution, m = max depth of tree
- Time: (worst-case analysis)
  - BFS = $O(b^d)$
  - DFS = $O(b^m)$
- Space:
  - BFS = $O(b^d)$ (always storing previous layer – suppose sol'n at bottom)
  - DFS = $O(bm)$ (all branches from path at each of m levels) (always storing successors)

Improving on brute-force: Iterative-deepening search

- Idea: Combine space efficiency of depth-first search with optimality of breadth-first.
- Make a breadth-first search into interative-deepening search:
  - Each iteration is a complete depth-first search with a cut-off (i.e., searches to a limited depth).
  - Can throw away previous computation each time and begin again.
- Eventually will find solution if one exists. Solution is guaranteed to have fewest arcs.
- Unnatural versus natural failure:
  - Depth limit is increased if DFS was truncated by reaching the depth limit. In this case, the search failed *unnaturally*.
  - The search failed *naturally* if the search did not prune any paths due to the depth limit. In this case, the program can stop and report no (more) paths.

Reference: DFS with cut-off (Iterative-deepening) versus DFS

- Complete? (guaranteed to find a solution)
  - Iterative-deepening: yes
  - DFS: no
- Optimal? (guaranteed to find solution at least depth?)
  - Iterative-deepening: yes
  - DFS: no
- b = branching factor, d = depth of solution, m = max depth of tree
- Time: (worst-case analysis)
  - Iterative-deepening = BFS = O (b^d)
- Space:
  - Iterative-deepening =similar to DFS = O (bd) (always storing previous layer but final layer dominates)
- Iterative=deepening search leads to very practical algorithms

Demo: Why computers can beat humans at chess

- Computers can use brute-force search to simulate moves ahead in chess game. Far more lookahead than humans can do!
- Applications: Chess, other alternate-player "zero-sum" games
  - "Zero-sum": Add up total player wins, subtract losses, sum is zero.
- From Text 10.3:
  - In the case where two agents are competing so that a positive reward for one is a negative reward for the other agent, we have a two-agent **zero-sum game**. The value of such a game can be characterized by a single number that one agent is trying to maximize and the other agent is trying to minimize. Having a single value for a two-agent zero-sum game leads to a **minimax** strategy. Each node is either a MAX node, if it is controlled by the agent trying to maximize, or is a MIN node if it is controlled by the agent trying to minimize.
- (continued)

Demo: Why computers can beat humans at chess (cont)

(comtinued)

- Can use clever method **(alpha-beta pruning**) to reduce the number of nodes that are searched. Stop evaluating a move if at least one possibility has been found that makes this move worse than a previously evaluated move. Thus, whole branches of the search tree can be avoided.
- In best case (best moves always searched first) search goes twice as deep with same amount of computation.
- Can also use heuristics to improve pruning, e.g., examine moves that take pieces before moves that do not.
- Demo: http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html
- Better demo but VERY slow:

  http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning