Artificial Intelligence

CS 165A

May 2, 2023

Instructor: Prof. Yu-Xiang Wang

Today

→ Informed Search and Heuristics

→ Games and minimax search

Notes for Project 1

- Almost everyone has completed
  - Don't forget to submit the report and the leaderboard

- How to catch up if you missed the deadline?
  - 4 late days --- no question asked
  - 75% credits if submit before this Thursday (for the basic coding)
  - 50% credits if submit after this Thursday

- 9 students were able to beat the TA baseline!
  - Truly awesome! Keep it coming!

- Start Project 2 early!

Recap: Search agent and search algorithms

- Representing states, operators and costs
  - State-space diagram: What are the vertices, edges, edge weights?
  - Examples: Romania, Missionary and Cannibals, Pacman, 8-puzzle (and the MU puzzle from the quiz)
- Search algorithms
  - BFS, DFS, Depth-Limited, IDS, Bidirectional Search
- Four criteria to evaluate the search algorithms:
  - Completeness, Optimality, Space complexity, time complexity

This lecture

- Uniform cost search
- Informed search, aka Heuristic Search
- Admissible and consistent heuristics
- Tree search vs Graph Search
- (if time permits) Intro to games and adversarial search

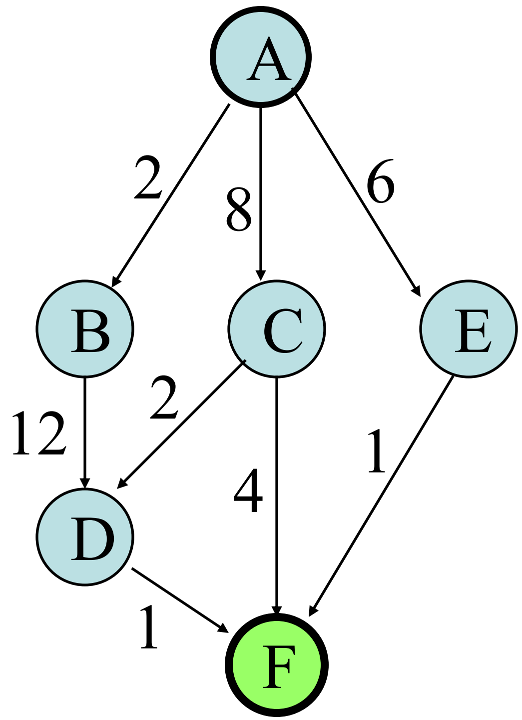
Uniform Cost Search

- Similar to breadth-first search, but always expands the lowest-cost node, as measured by the path cost function, *g(n)*
  - *g(n)* is (actual) cost of getting to node n
  - Breadth-first search is actually a special case of uniform cost search, where *g(n)* = DEPTH(*n*)
  - If the path cost is monotonically increasing, uniform cost search will find the optimal solution

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution or failure
  return **GENERAL-SEARCH**(*problem,* ENQUEUE-IN-COST-ORDER)

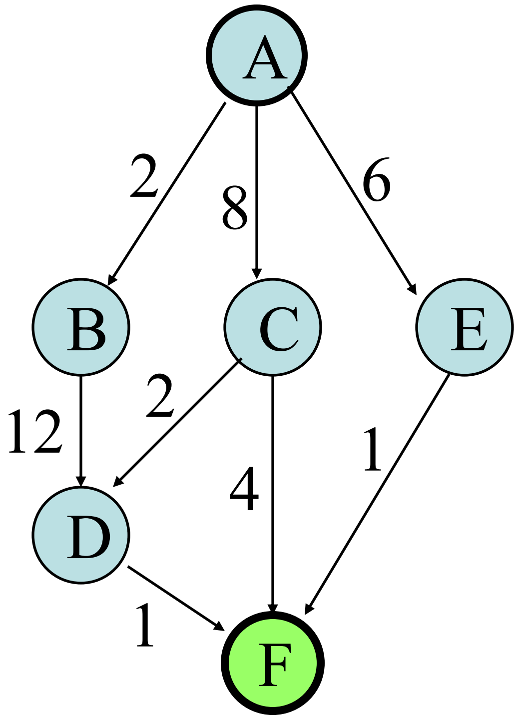
(Dijkstra's algorithm of an potentially infinite graph)

Example (3 min work)



Try breadth-first and uniform cost

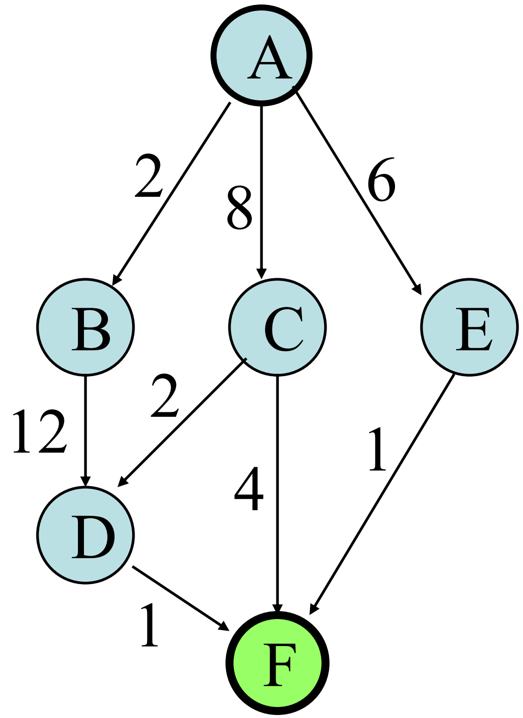
Example (3 min work): Breath-First Search



Node to expand:

Frontier:

Example (3 min work): Uniform Cost Search



Node to expand:

Frontier:

Uniform-Cost Search

C = optimal cost
ε = minimum step cost

- Complete? Yes, if $\varepsilon > 0$
- Optimal? Yes
- Time complexity? Exponential: $\mathbf{O}(\boldsymbol{b}^{\lfloor C/\varepsilon \rfloor})$
- Space complexity? Exponential: $\mathbf{O}(\boldsymbol{b}^{\lfloor C/\varepsilon \rfloor})$

Same as breadth-first if all edge costs are equal

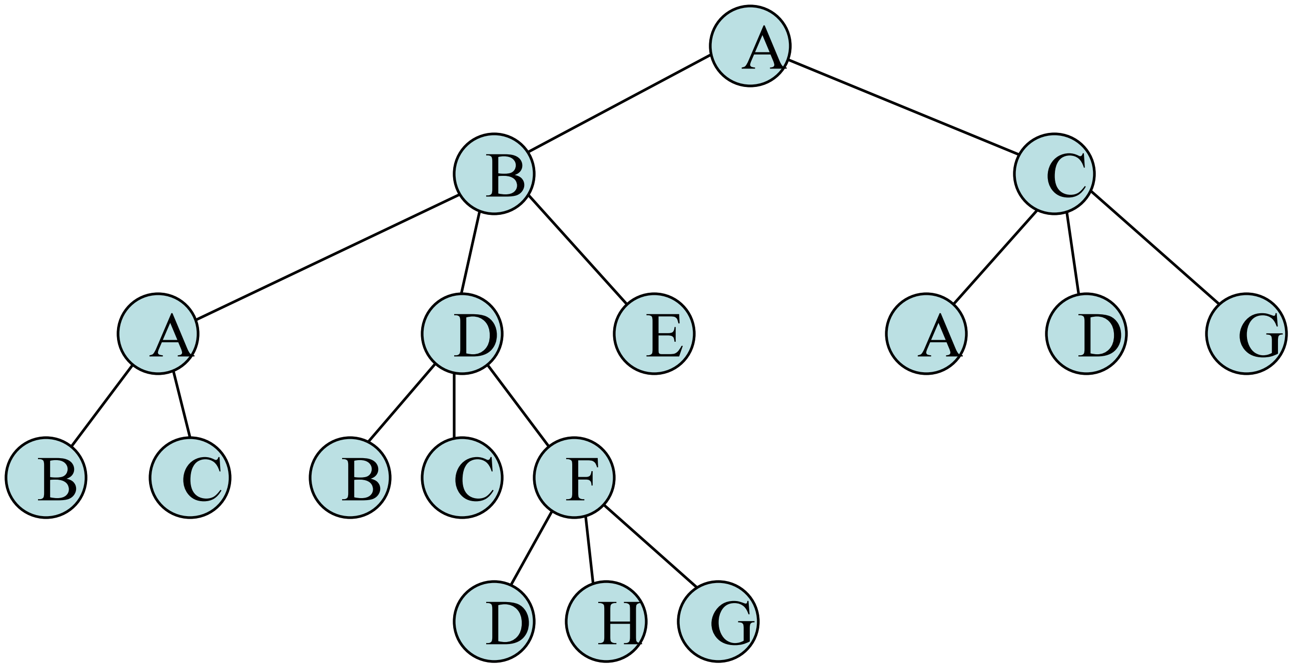
Can we do better than Tree Search?

- Sometimes.

- When the number of states are small
  - Dynamic programming (smart way of doing exhaustive search)

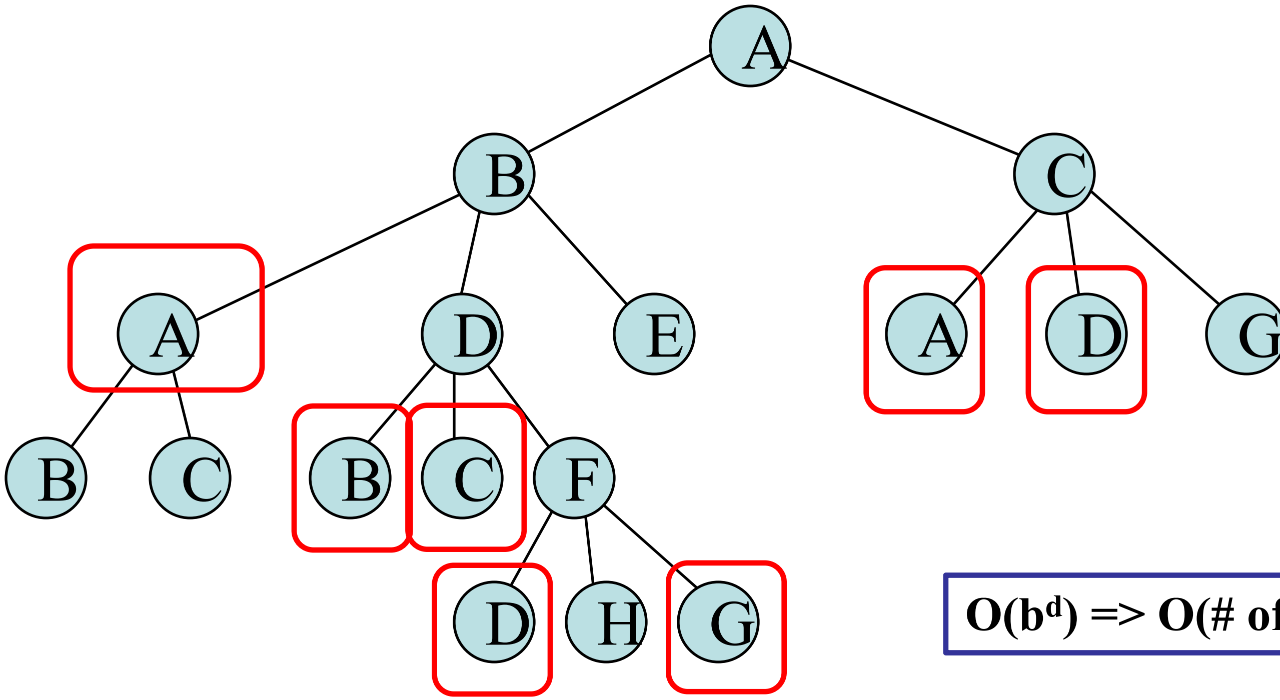
State Space vs. Search Tree (cont.)

Search tree (partially expanded)



Search Tree => Search Graph

Dynamic programming (with book keeping)



O(b^d) => O(# of states)

Graph Search vs Tree Search

- Tree Search
  - We might repeat some states
  - But we do not need to remember states

- Graph Search
  - We remember all the states that have been explored
  - But we do not repeat some states

Summary table of uninformed search

Criteria	BFS	Uniform-cost	DFS	Depth-limited	IDS	Bidirectional
Complete?	Yes#	Yes#&	No	No	Yes#	Yes#+
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/e \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/e \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes$	Yes	No	No	Yes$	Yes$+

b: Branching factor
d: Depth of the shallowest goal
l: Depth limit
m: Maximum depth of search tree
e: The lower bound of the step cost

(Section 3.4.6 in the AIMA book.)

#: Complete if *b* is finite
&: Complete if step cost >= *e*
$: Optimal if all step costs are identical
+: If both direction use BFS

Practical note about search algorithms

- The computer can't "see" the search graph like we can
  - No "bird's eye view" – make relevant information explicit!
- What information should you keep for a node in the search tree?
  - State
    - (1 2 0)
  - Parent node (or perhaps complete ancestry)
    - Node #3 (or, nodes 0, 2, 5, 11, 14)
  - Depth of the node
    - $d = 4$
  - Path cost up to (and including) the node
    - $g(node) = 12$
  - Operator that produced this node
    - Operator #1

Remainder of the lecture

- Informed search

- Some questions / desiderata
  1. Can we do better with some side information?
  2. We do not wish to make strong assumptions on the side information.
  3. If the side information is good, we hope to do better.
  4. If the side information is useless, we perform as well as an uninformed search method.

Best-First Search (with an Eval-Fn)

function BEST-FIRST-SEARCH(*problem,* EVAL-FN) **returns** a solution or failure
QUEUING-FN ← a function that orders nodes by EVAL-FN
return GENERAL-SEARCH(*problem,* QUEUING-FN)

- Uses a heuristic function, ***h(n)****,* as the EVAL-FN
- ***h(n)*** estimates the cost of the best path from state *n* to a goal state
  - *h(goal)* = 0

Greedy Best-First Search

- Greedy search – always expand the node that appears to be the closest to the goal (i.e., with the smallest ***h***)
  - Instant gratification, hence "greedy"

function GREEDY-SEARCH(*problem*, ***h***) **returns** a solution or failure
  return BEST-FIRST-SEARCH(*problem*, ***h***)

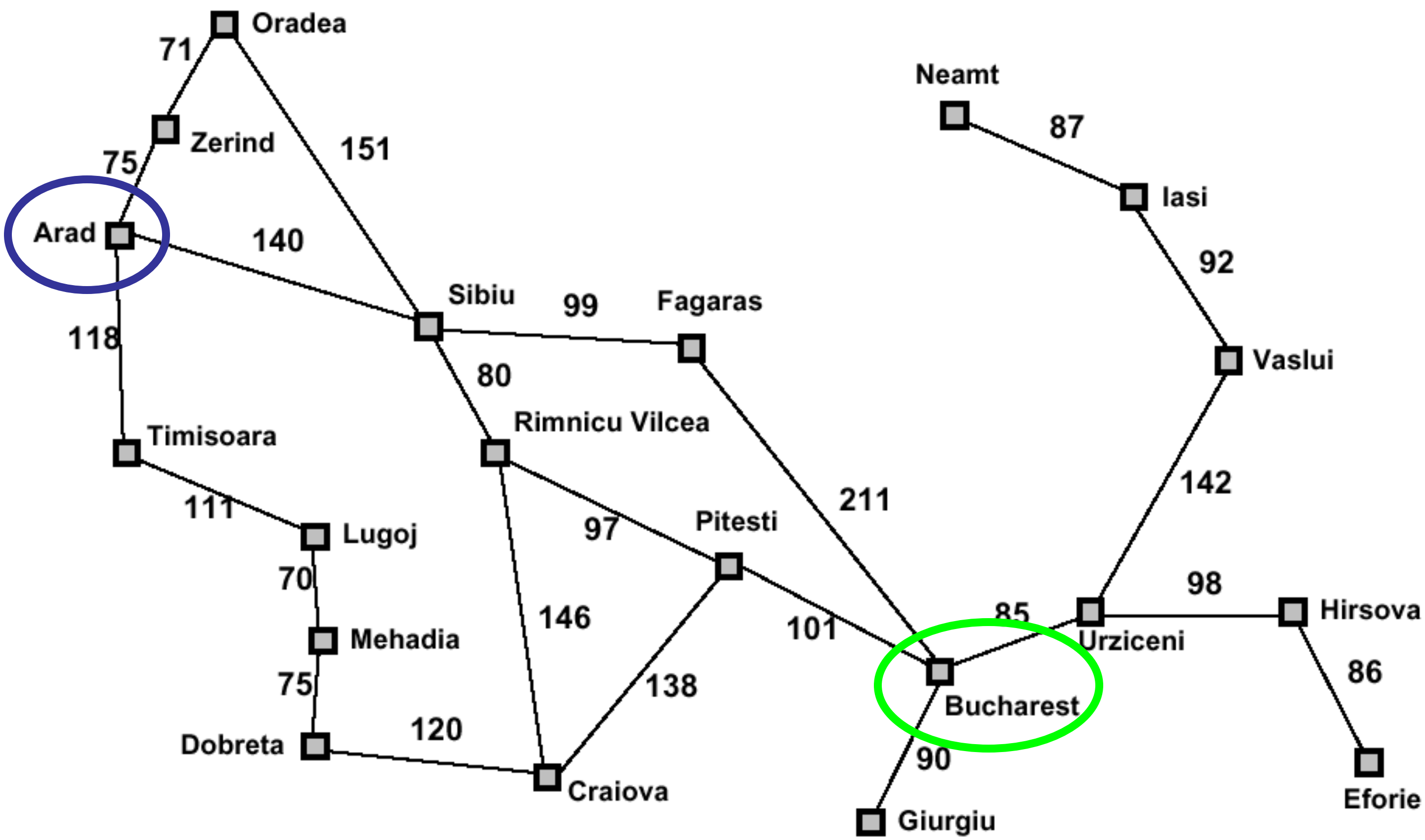
- Greedy search often performs well, but:
  - It doesn't always find the best solution / or any solution
  - It may get stuck
  - It performance completely depends on the particular ***h*** function

A* Search (Pronounced "A-Star")

- Uniform-cost search minimizes ***g(n)*** ("past" cost)
- Greedy search minimizes ***h(n)*** ("expected" or "future" cost)
- **"A*** Search" combines the two:
  - Minimize $f(n) = g(n) + h(n)$
  - Accounts for the "past" and the "future"
  - Estimates the cheapest solution (complete path) through node ***n***

function A*-SEARCH(*problem*, ***h***) **returns** a solution or failure
  return BEST-FIRST-SEARCH(*problem*, ***f***)

A* Example

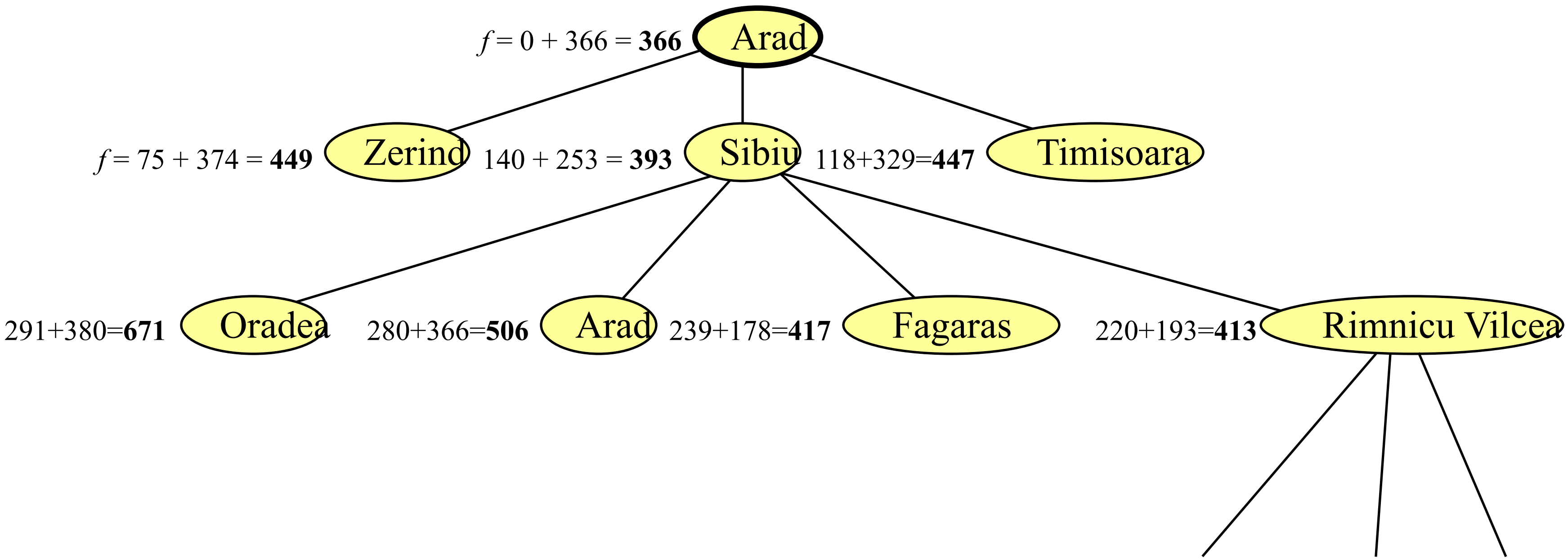


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$$f(n) = g(n) + h(n)$$

A* Example



f = 0 + 366 = **366**
Arad
f = 75 + 374 = **449**
Zerind
140 + 253 = **393**
Sibiu
118+329=**447**
Timisoara
291+380=**671**
Oradea
280+366=**506**
Arad
239+178=**417**
Fagaras
220+193=**413**
Rimnicu Vilcea

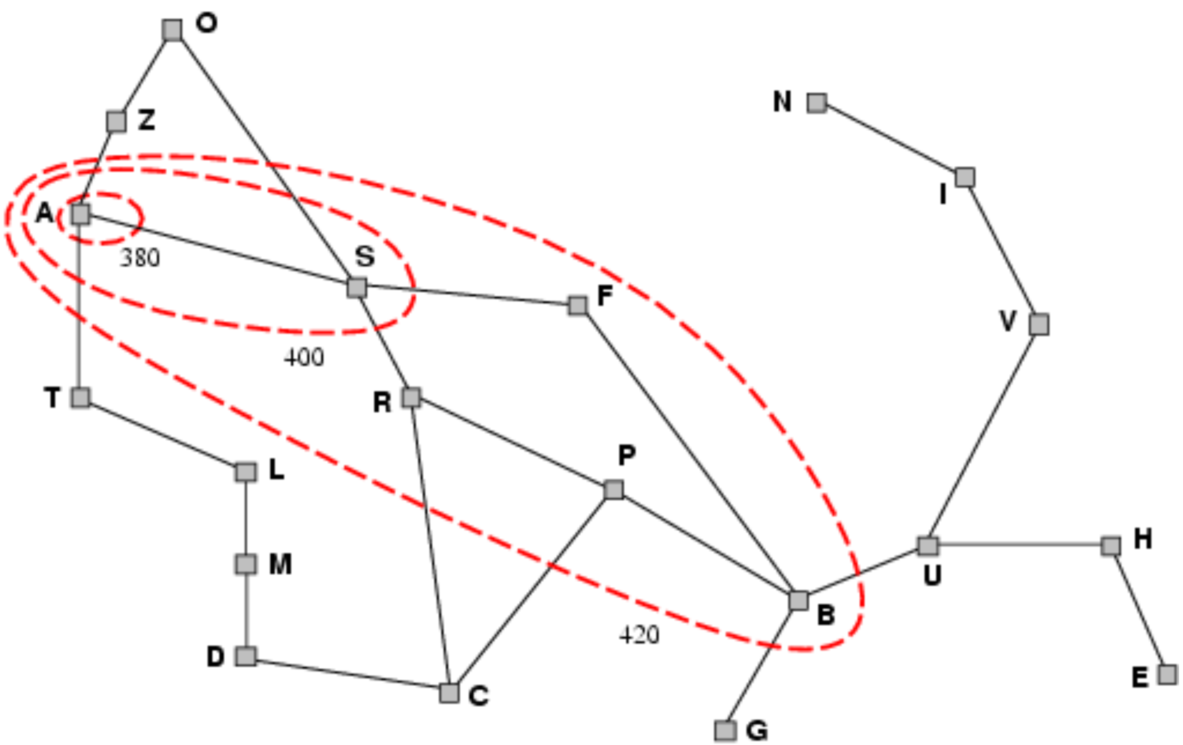
When does A* search "work"?

- Focus on optimality (finding the optimal solution)

- "**A***** Search" is optimal if *h* is **admissible**
  - ***h*** is optimistic – it never overestimates the cost to the goal
    - $h(n) \leq$ true cost to reach the goal
  - So $\boldsymbol{f(n)}$ never overestimates the actual cost of <u>the best solution passing through node *n*</u>

Visualizing A* search

- A* expands nodes in order of increasing f value
- Gradually adds "f-contours" of nodes
- Contour i has all nodes with $f=f_i$, where $f_i < f_{i+1}$
-



Optimality of A^* with an Admissible h

- Let OPT be the optimal path cost.
  - All non-goal nodes on this path have f ≤ OPT.
    - Positive costs on edges
  - The goal node on this path has f = OPT.

- A* search does not stop until an f-value of OPT is reached.
  - All other goal nodes have an f cost higher than OPT.

- All non-goal nodes on the optimal path are eventually expanded.
  - The optimal goal node is eventually placed on the priority queue, and reaches the front of the queue.

Optimal Efficiency of A*

A* is **optimally efficient** for any particular *h(n)*
That is, no other optimal algorithm is guaranteed to expand fewer nodes with the same *h(n)*.

- Need to find a good and efficiently evaluable h(n).

A* Search with an Admissible h

- Optimal? Yes
- Complete? Yes
- Time complexity? Exponential; better under some conditions
- Space complexity? Exponential; keeps all nodes in memory

Recall: Graph Search vs Tree Search

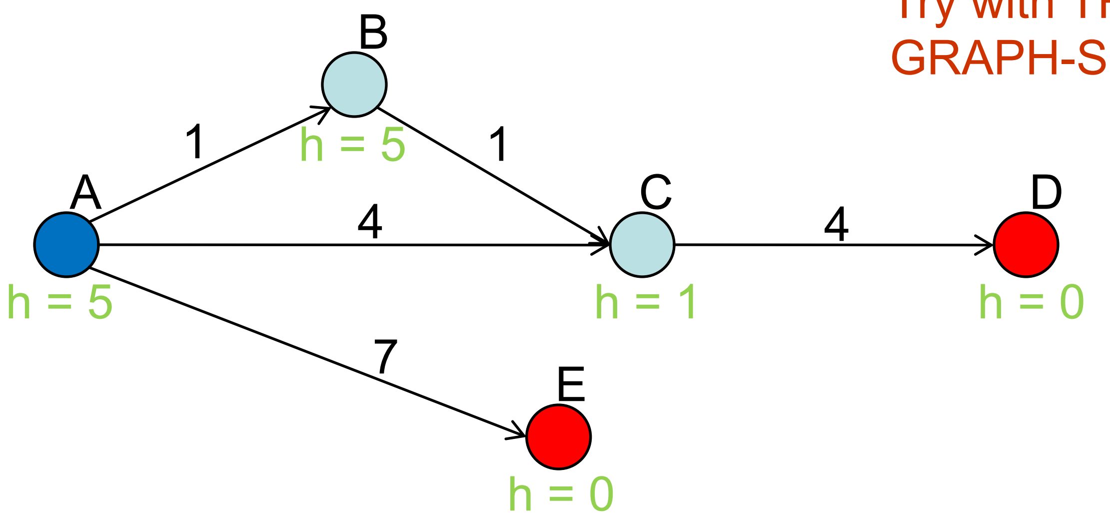
- Tree Search
  – We might repeat some states
  – But we do not need to remember states

- Graph Search
  - We remember all the states that have been explored
  - But we do not repeat some states

Avoiding Repeated States using A* Search

- Is GRAPH-SEARCH optimal with A*?

Try with TREE-SEARCH and GRAPH-SEARCH

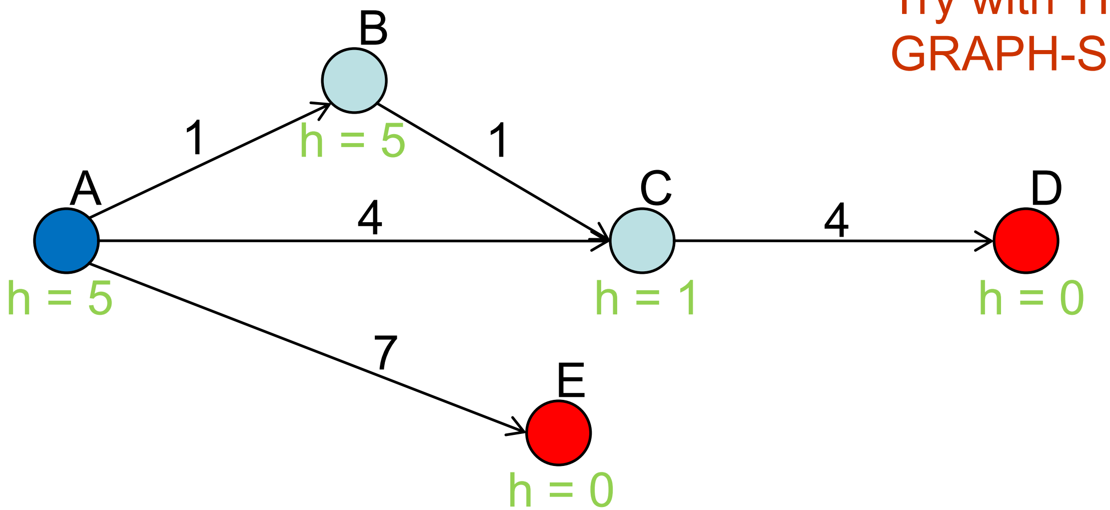


Graph Search
Step 1: Among B, C, E, Choose C
Step 2: Among B, E, D, Choose B
Step 3: Among D, E, Choose E. (you are not going to select C again)

Avoiding Repeated States using A* Search

- Is GRAPH-SEARCH optimal with A*?

Try with TREE-SEARCH and GRAPH-SEARCH

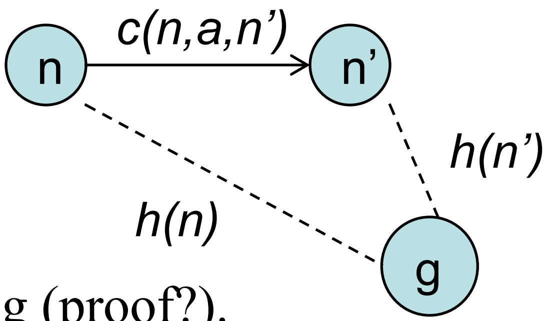


Solution 1: Remember all paths: Need extra bookkeeping

Solution 2: Ensure that the first path to a node is the best!

Consistency (Monotonicity) of heuristic h

- A heuristic is consistent (or monotonic) provided
  - for any node n, for any successor n' generated by action a with cost c(n,a,n')
    - $h(n) \leq c(n,a,n') + h(n')$
  - akin to triangle inequality.
  - guarantees admissibility (proof?).
  - values of f(n) along any path are non-decreasing (proof?).
    - Contours of constant f in the state space
- GRAPH-SEARCH using consistent f(n) is optimal.
- Note that h(n) = 0 is consistent and admissible.



Remainder of the lecture

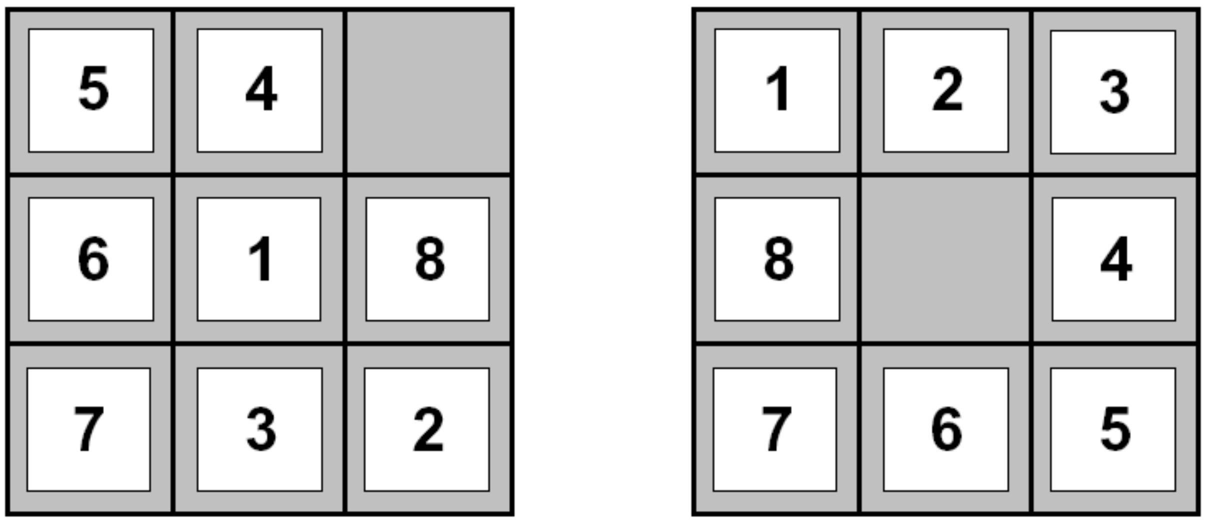
- Examples
- Choosing heuristics
- Games and Minimax Search

Heuristics

- What's a heuristic for
  - Driving distance (or time) from city A to city B ?
  - 8-puzzle problem ?
  - M&C ?
  - PACMAN?

- **<u>Admissible</u>** heuristic
  - Does not overestimate the cost to reach the goal
  - "Optimistic"
- Are the above heuristics admissible? Consistent?

Example: 8-Puzzle



Start State Goal State

Comparing and combining heuristics

- Heuristics generated by considering relaxed versions of a problem.
- Heuristic h_1 for 8-puzzle
  - Number of out-of-order tiles
- Heuristic h_2 for 8-puzzle
  - Sum of Manhattan distances of each tile
- h_2 dominates h_1 provided $h_2(n) \geq h_1(n)$.
  - h_2 will likely prune more than h_1.
- $\max(h_1, h_2, .., h_n)$ is
  - admissible if each h_i is
  - consistent if each h_i is
- Cost of sub-problems and pattern databases
  - Cost for 4 specific tiles
  - Can these be added for disjoint sets of tiles?

Effective Branching Factor

- Though informed search methods may have poor *worst-case* performance, they often do quite well if the heuristic is good
  - Even if there is a huge branching factor

- One way to quantify the effectiveness of the heuristic: the **effective branching factor, *b****
  - N: total number of nodes expanded
  - d: solution depth
  - $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

- For a good heuristic, b^* is close to 1

Example: 8-puzzle problem

Averaged over 100 trials each at different solution lengths

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Ave. # of nodes expanded

Solution length

Summary of informed search

- How to use a heuristic function to improve search
  - Greedy Best-first search + Uniform-cost search = A* Search
- When is A* search optimal?
  - h is Admissible (optimistic) for Tree Search
  - h is Consistent for Graph Search
- Choosing heuristic functions
  - A good heuristic function can reduce time/space cost of search by orders of magnitude.
  - Good heuristic function may take longer to evaluate.

Memory Bounded Search

- Memory, not computation, is <u>usually</u> the limiting factor in search problems
  - Certainly true for A* search
- Why? What takes up memory in A* search?

- Solution: Memory-bounded A* search
  - Iterative Deepening A* (IDA*)
  - Simplified Memory-bounded A* (SMA*)
  - (Read the textbook for more details.)

  - Very popular choice: Beam Search (Application in Decoding for Large Language Model!)

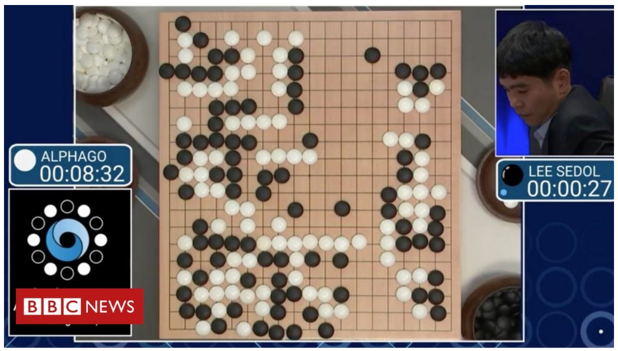
Summary of informed search

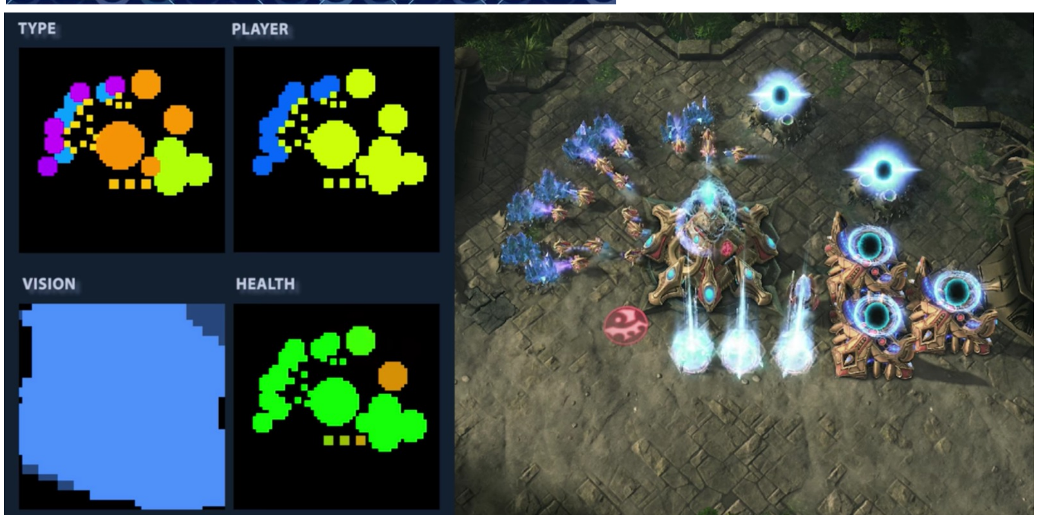
- How to use a heuristic function to improve search
  - Greedy Best-first search + Uniform-cost search = A* Search

- When is A* search optimal?
  - h is Admissible (optimistic) for Tree Search
  - h is Consistent for Graph Search

- Choosing heuristic functions
  - A good heuristic function can reduce time/space cost of search by orders of magnitude.
  - Good heuristic function may take longer to evaluate.

Games and Adversarial Search





- Games: problem setup
- Minimax search
- Alpha-beta pruning

Illustrative example of a simple game (1 min discussion)

Example: game 1

You choose one of the three bins.
I choose a number from that bin.
Your goal is to maximize the chosen number.

A	B	C
-50 50	1 3	-5 15

(Example taken from Liang and Sadigh)

Game as a search problem

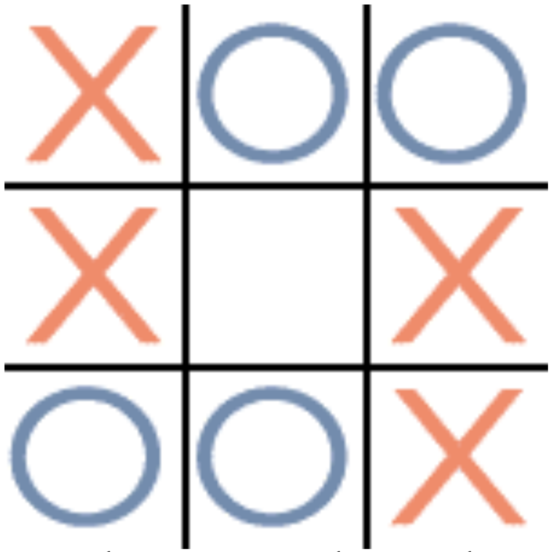
- S_0 The initial state
- PLAYER(s): Returns which player has the move
- ACTIONS(s): Returns the legal moves.
- RESULT(s, a): Output the state we transition to.
- TERMINAL-TEST(s): Returns True if the game is over.
- UTILITY(s,p): The payoff of player p at terminal state s.

Two-player, Turn-based, Perfect information, Deterministic, Zero-Sum Game

- Two-player: Tic-Tac-Toe, Chess, Go!
- Turn-based: The players take turns in round-robin fashion.
- Perfect information: The State is known to everyone
- Deterministic: Nothing is random
- Zero-sum: The total payoff for all players is a constant.
  - *The 8-puzzle is a one-player, perfect info, deterministic, zero-sum game.*
  - *How about Rock-Paper-Scissors?*
  - *How about Monopoly?*
  - *How about Starcraft?*

Tic-Tac-Toe

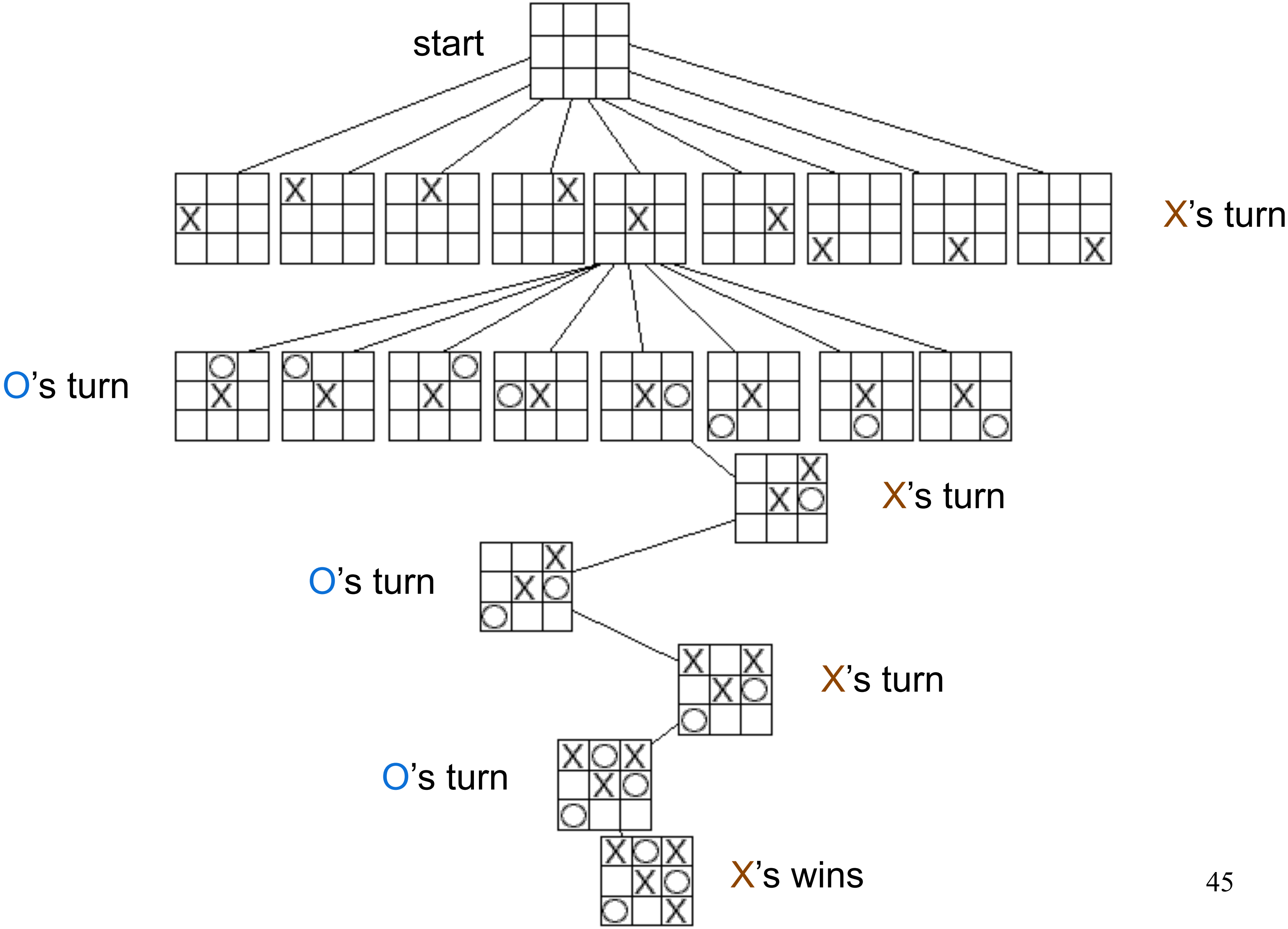
- The first player is **X** and the second is **O**
- Object of game: get three of your symbol in a horizontal, vertical or diagonal row on a 3x3 game board
- **X** always goes first



- Players alternate placing **X**s and **O**s on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled (a draw)

What's the state, action, transition, payoff for Tic-Tac-Toe?

Partial game tree for Tic-Tac-Toe



Game trees

- A game tree is like a search tree in many ways …
  – nodes are search states, with full details about a position
    - characterize the arrangement of game pieces on the game board
  – edges between nodes correspond to moves
  – leaf nodes correspond to a set of goals
    - { win, lose, draw }
    - usually determined by a score for or against player
  – at each node it is one or other player’s turn to move
- A game tree is not like a search tree because you have an opponent!

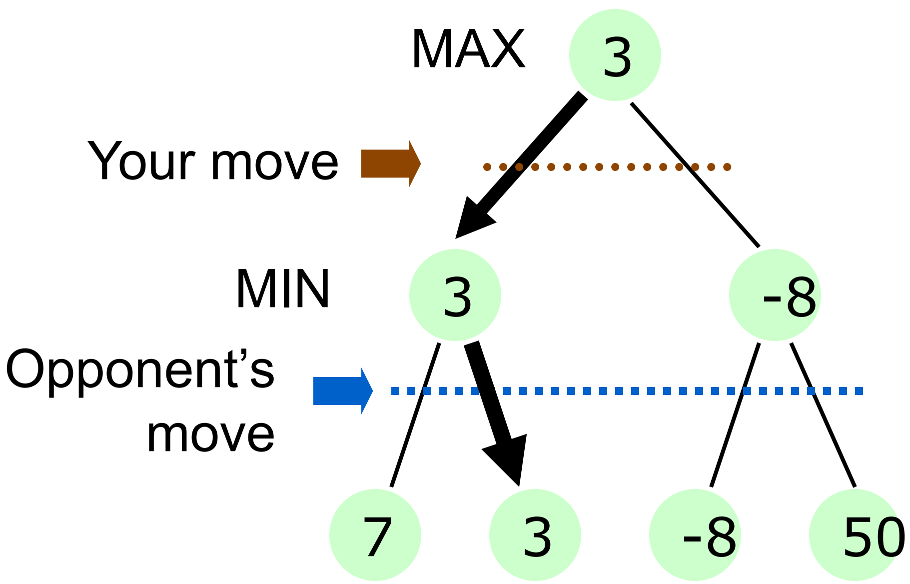
Two players: MIN and MAX

- In a zero-sum game:
  - payoff to Player 1 = - payoff to Player 2
- The goal of Player 1 is to maximizing her payoff.
- The goal of Player 2 is to maximizing her payoff as well
  - Equivalent to minimizing Player 1's payoff.

Minimax search

- Assume that both players play perfectly
  – do not assume player will miss good moves or make mistakes
- Score(s): The score that MAX will get towards the end if both player play perfectly from s onwards.
- Consider MIN's strategy
  – MIN's best strategy:
    - choose the move that minimizes the score that will result when MAX chooses the maximizing move
  – MAX does the opposite

Minimaxing

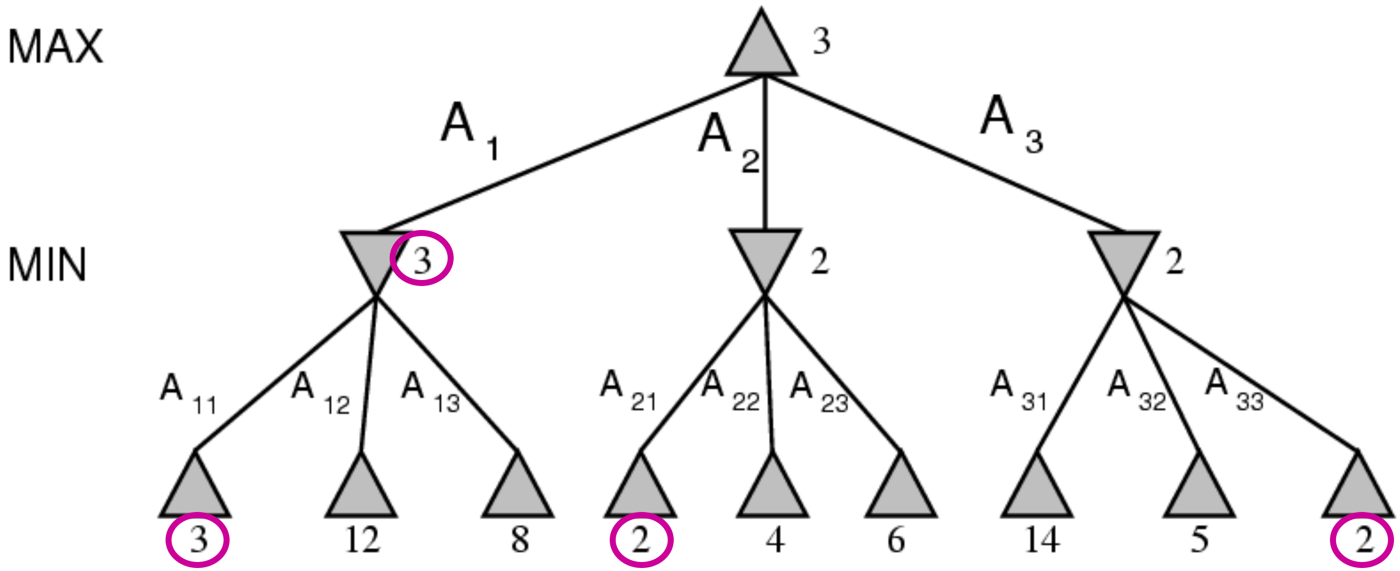


- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

Each move is called a "ply". One round is K-plies for a K-player game.

Minimax example

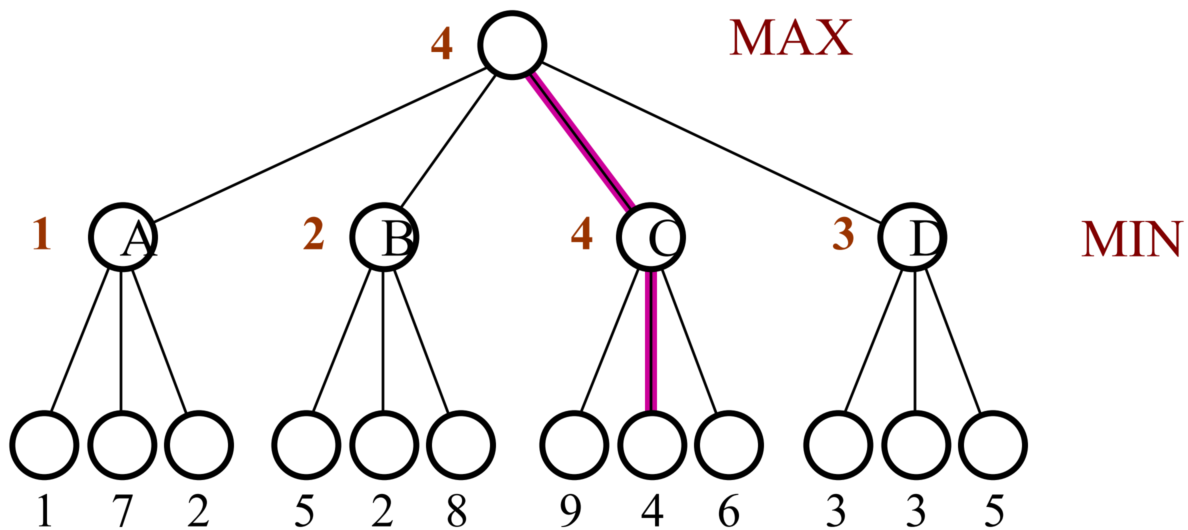
Which move to choose?



The **minimax decision** is move $\mathbf{A_1}$

Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)

- Generate the tree of minimax values
  - Then choose best (maximum) move
  - Don’t need to keep all values around
    - Good memory property

- Depth-first search is used to implement minimax
  - Expand all the way down to leaf nodes
  - Recursive implementation

Minimax properties

- Optimal?

  Yes, against an optimal opponent, **if** the tree is finite

- Complete?

  Yes, **if** the tree is finite

- Time complexity?

  Exponential: **O(** $\boldsymbol{b^m}$ **)**

- Space complexity?

  Polynomial: **O(** $\boldsymbol{bm}$ **)**

But this could take forever…

- Exact search is intractable
  - Tic-Tac-Toe is $9! = 362{,}880$
  - For chess, $b \approx 35$ and $m \approx 100$ for “reasonable” games
  - Go is $361! \approx 10^{750}$

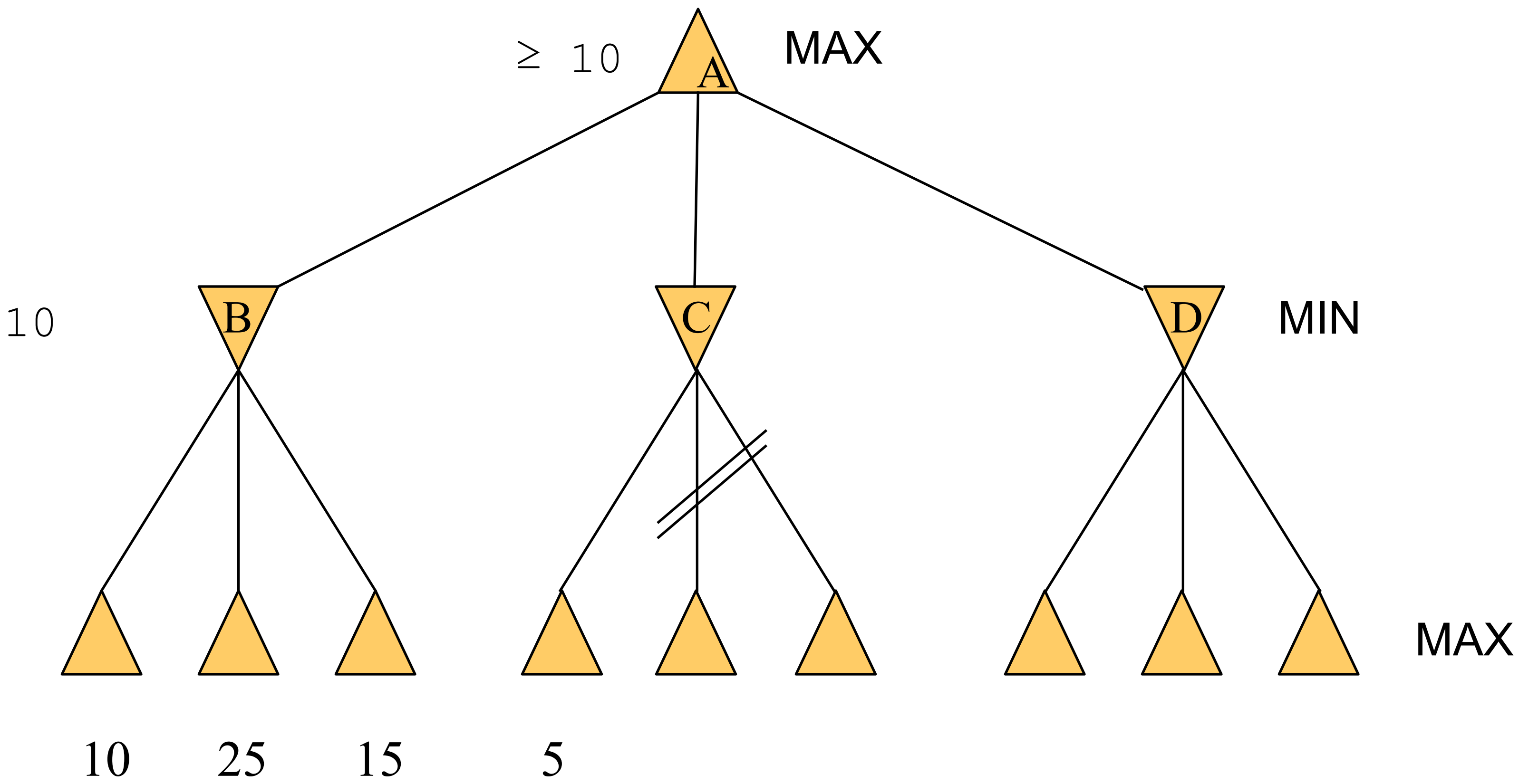
- Idea 1: Pruning

- Idea 2: Cut off early and use a heuristic function

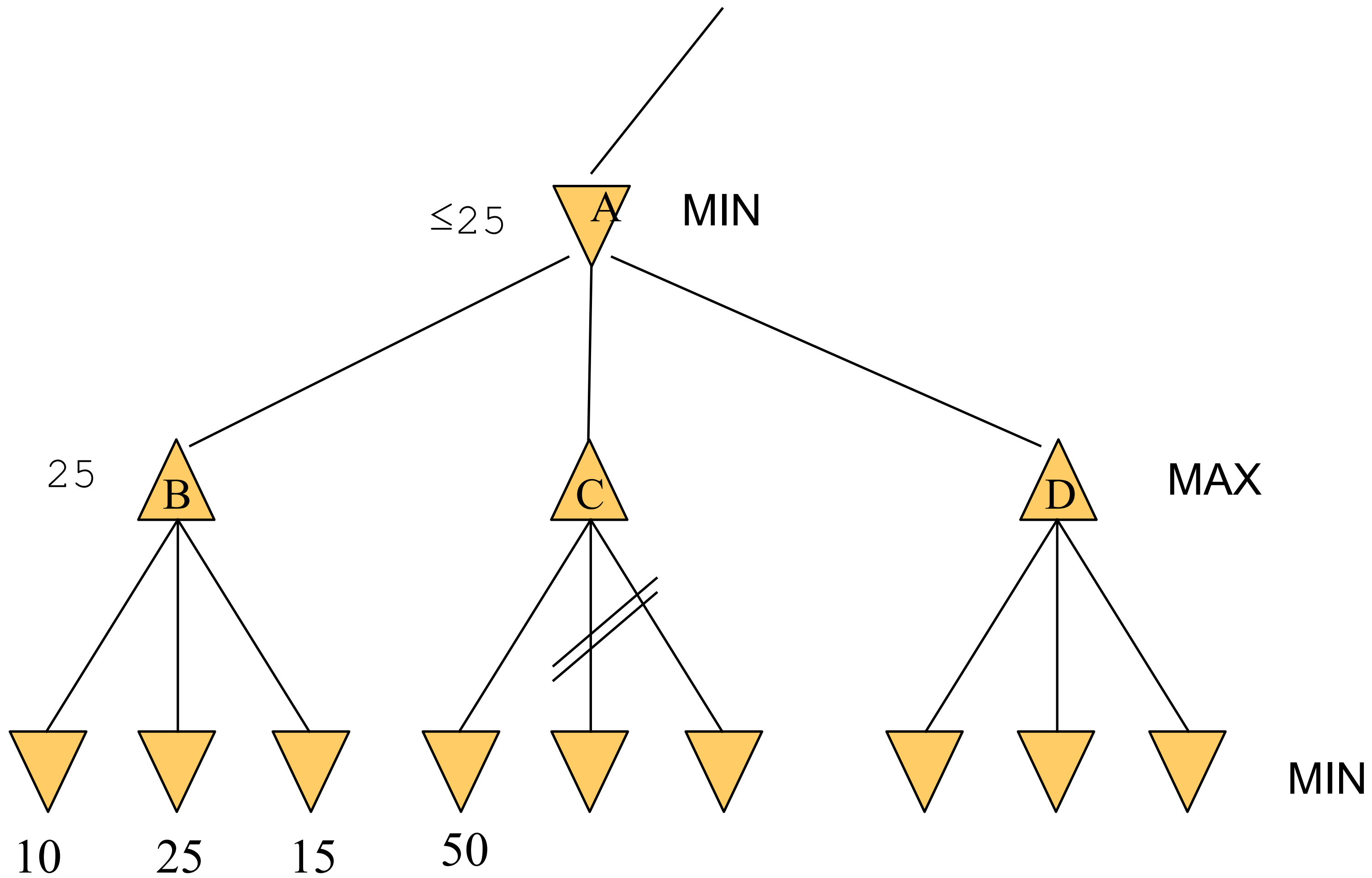
Pruning

- What’s really needed is “smarter,” more efficient search
  - Don’t expand “dead-end” nodes!
- **Pruning** – eliminating a branch of the search tree from consideration
- **Alpha-beta pruning**, applied to a minimax tree, returns the same “best” move, while pruning away unnecessary branches
  - Many fewer nodes might be expanded
  - Hence, smaller effective branching factor
  - …and deeper search
  - …and better performance
    - Remember, minimax is *depth-first* search

Alpha pruning



Beta pruning



Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering $O(b^{m/2})$
- Random ordering $O(b^{3m/4})$
- For specific games like Chess, you can get to almost perfect ordering.

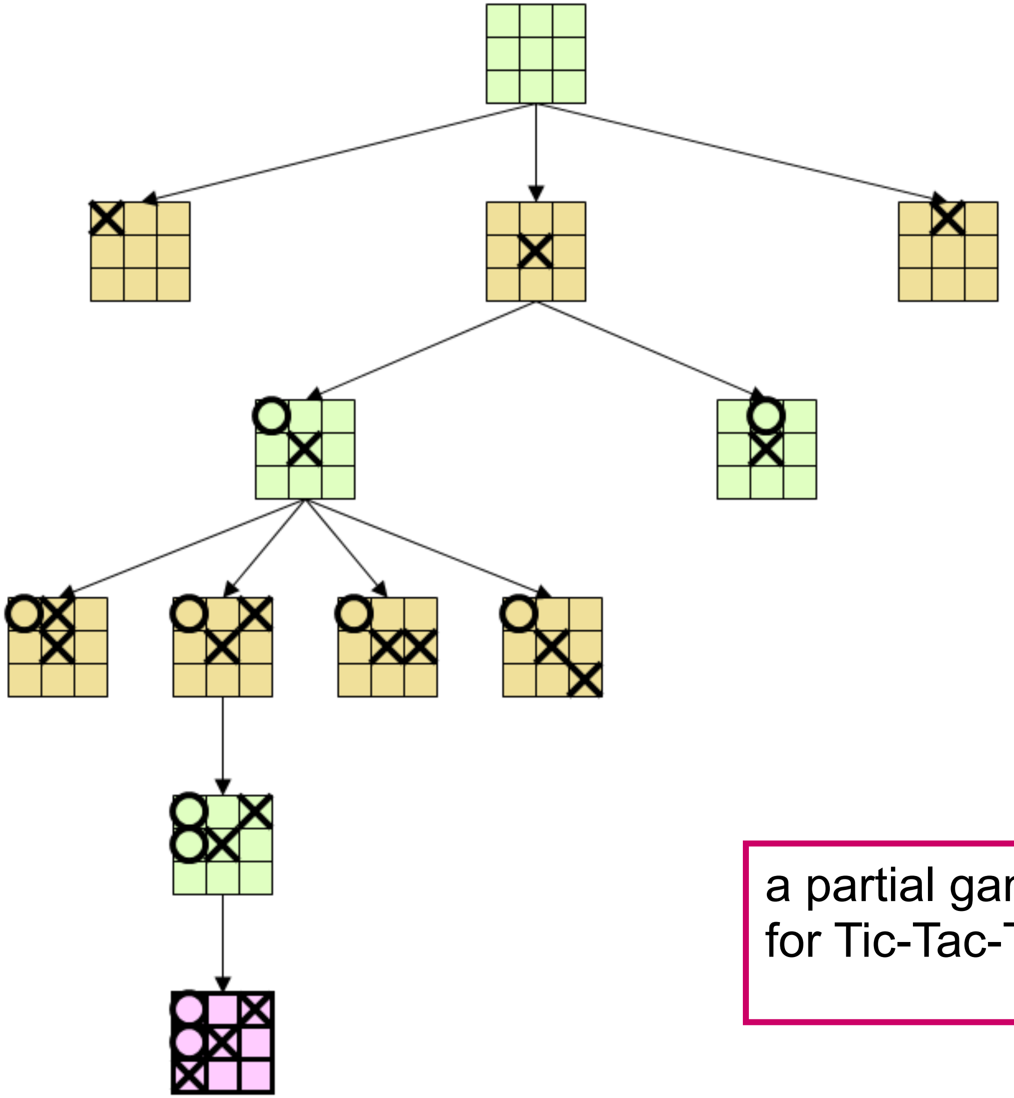
Heuristic (Evaluation function)

- It is usually impossible to solve games completely
- Rather, <u>cut the search off early</u> and apply a heuristic evaluation function to the leaves
  - ***h*(*s*)** estimates the expected utility of the game from a given position (node/state) ***s***
  - like depth bounded depth first, lose completeness
  - Explore game tree using combination of evaluation function and search
- The performance of a game-playing program depends on the quality (and speed!) of its evaluation function

Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_d f_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{$ # pawns advantage, # bishops advantage, # knights advantage, # rooks advantage, # queens advantage $\}$
  - Is this what Deep Blue used?
  - What are some problems with this?
- More complex evaluation functions may involve <u>learning</u>
  - Adjusting weights based on outcomes
  - Perhaps non-linear functions
  - How to choose the *features*?

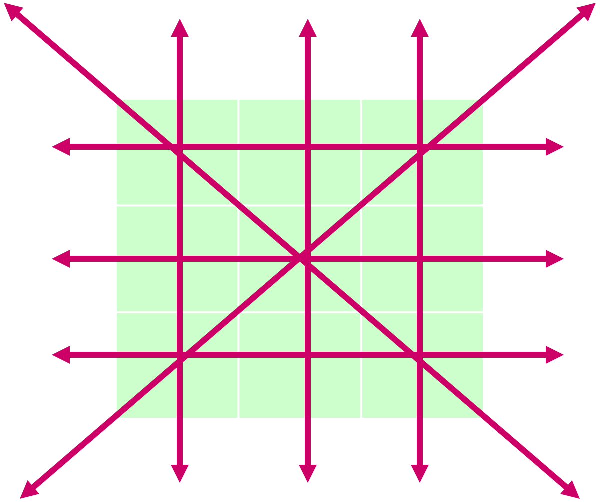
Tic-Tac-Toe revisited



a partial game tree
for Tic-Tac-Toe

Evaluation function for Tic-Tac-Toe

- A simple evaluation function for Tic-Tac-Toe
  - count the number of rows where **X** can win
  - subtract the number of rows where **O** can win
- Value of evaluation function at start of game is zero
  - on an empty game board there are 8 possible winning rows for both **X** and **O**



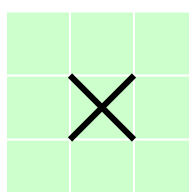
8-8 = 0

Evaluating Tic-Tac-Toe

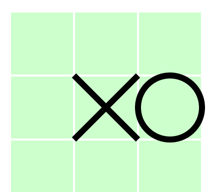
```
evalX = (number of rows where X can win)  –
              (number of rows where O can win)
```

- After **X** moves in center, score for **X** is +4
- After **O** moves, score for **X** is +2
- After **X**'s next move, score for **X** is +4

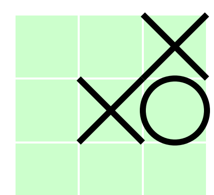
8-8 = 0



8-4 = 4



6-4 = 2

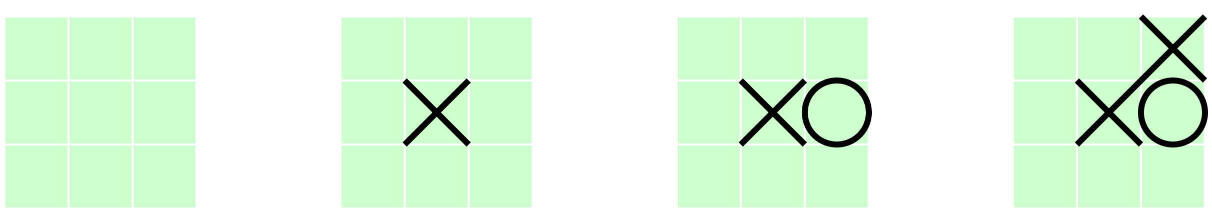


6-2 = 4

Evaluating Tic-Tac-Toe

```
evalO = (number of rows where O can win)  –
            (number of rows where X can win)
```

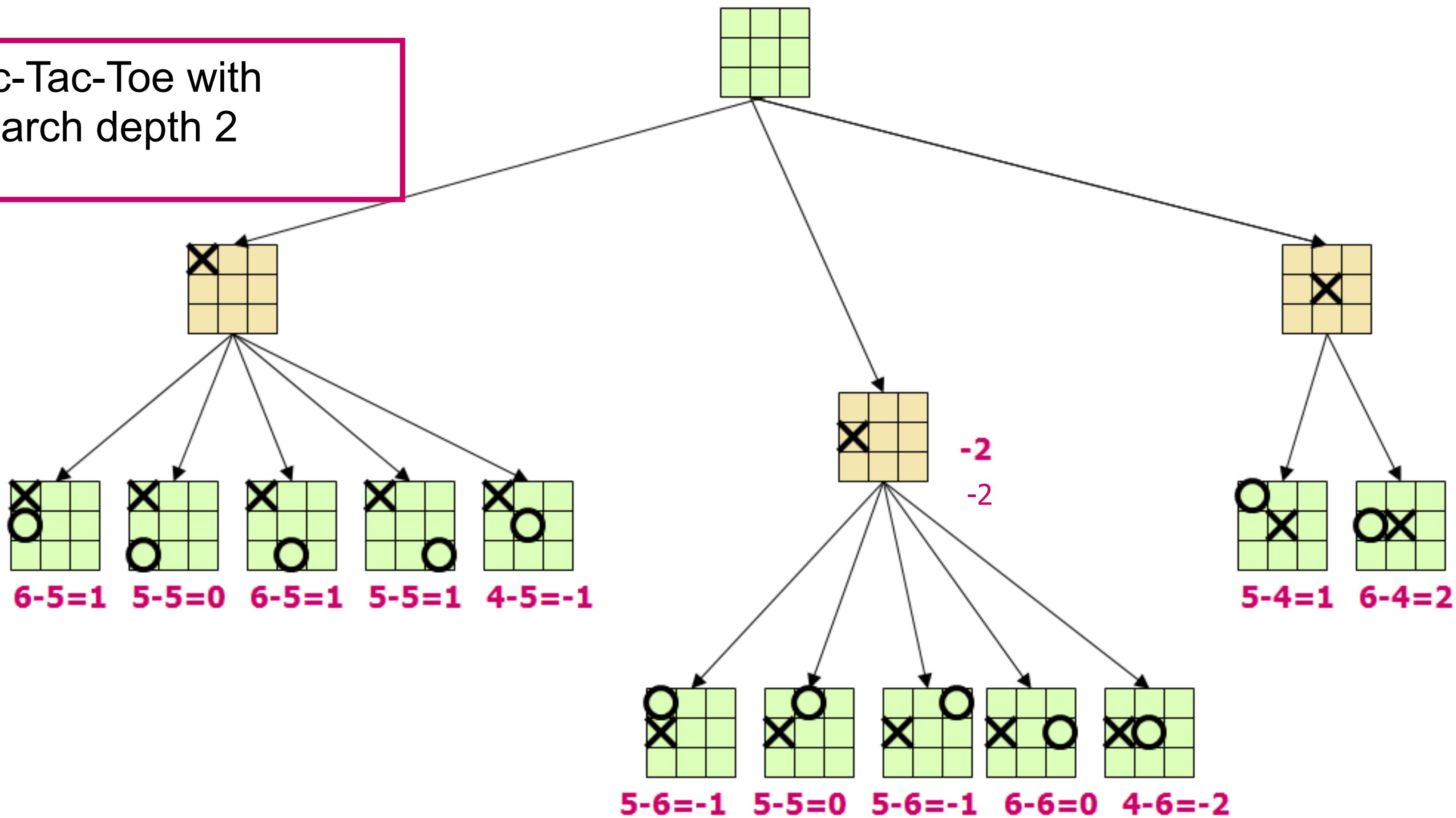
- After **X** moves in center, score for **O** is -4
- After **O** moves, score for **O** is +2
- After **X**'s next move, score for **O** is -4



8-8 = 0   4-8 = -4   4-6 = -2   2-6 = -4

Search depth cutoff

Tic-Tac-Toe with search depth 2



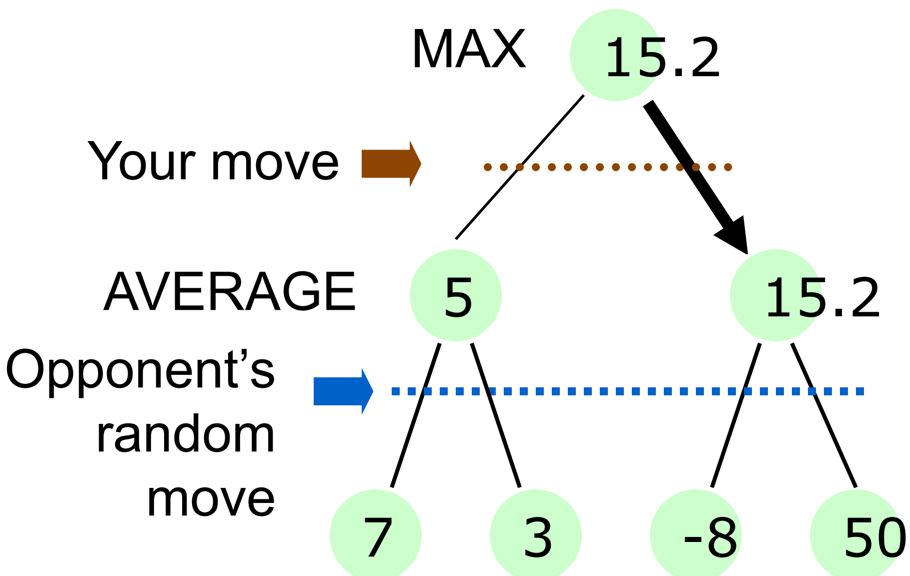
Evaluations shown for X

Expectimax: Playing against a benign opponent

- Sometimes your opponents are not clever.
  - They behave randomly.
  - You can take advantage of that by modeling your opponent.

- Example of game of chance:
  - Slot machines
  - Tetris

Expectimax example



- Your opponent behave randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability [0.5,0.5]
- If you move right, your opponent will select actions with [0.6,0.4]

Note: pruning becomes tricky in expectimax… think about why.

Summary of game playing

- Minimax search
- Game tree
- Alpha-beta pruning
- Early stop with an evaluation function
- Expectimax

More reading / resources about game playing

- Required reading: AIMA 5.1-5.3
- Stochastic game / Expectiminimax: AIMA 5.5
  - Backgammon. TD-Gammon
  - Blackjack, Poker
- Famous game AI: Read the “Historical notes” of the AIMA Chapter 5
  - Deep blue
  - TD Gammon
- AlphaGo: https://www.nature.com/articles/nature16961