Deterministic Tree Search

aka Deterministic Tree-based Planning
aka “Search”

finding a path from start state to a goal state

with thanks for slides to Russ Greiner

Search Algorithm Properties

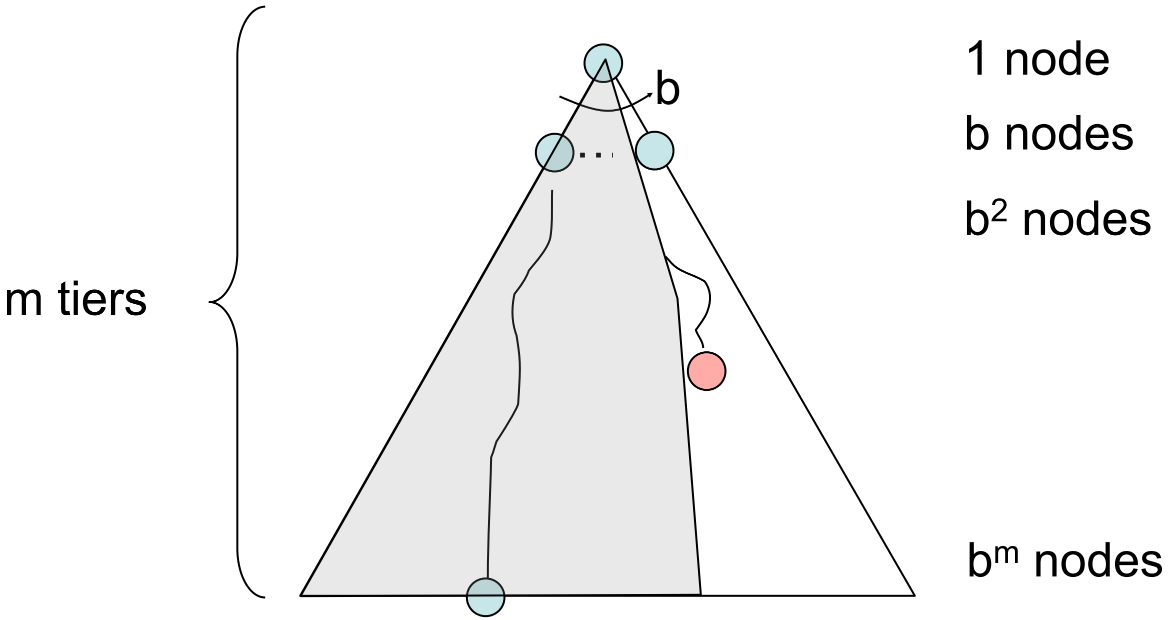
- Complete? Guaranteed to find a solution if one exists?
- Optimal? Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

Variables:

n	Number of states in the problem
b	The average branching factor *B* (the average number of successors)
s	Depth of the shallowest solution
m	Max depth of the search tree

Depth-First Search (DFS)

- With cycle checking, DFS is complete.

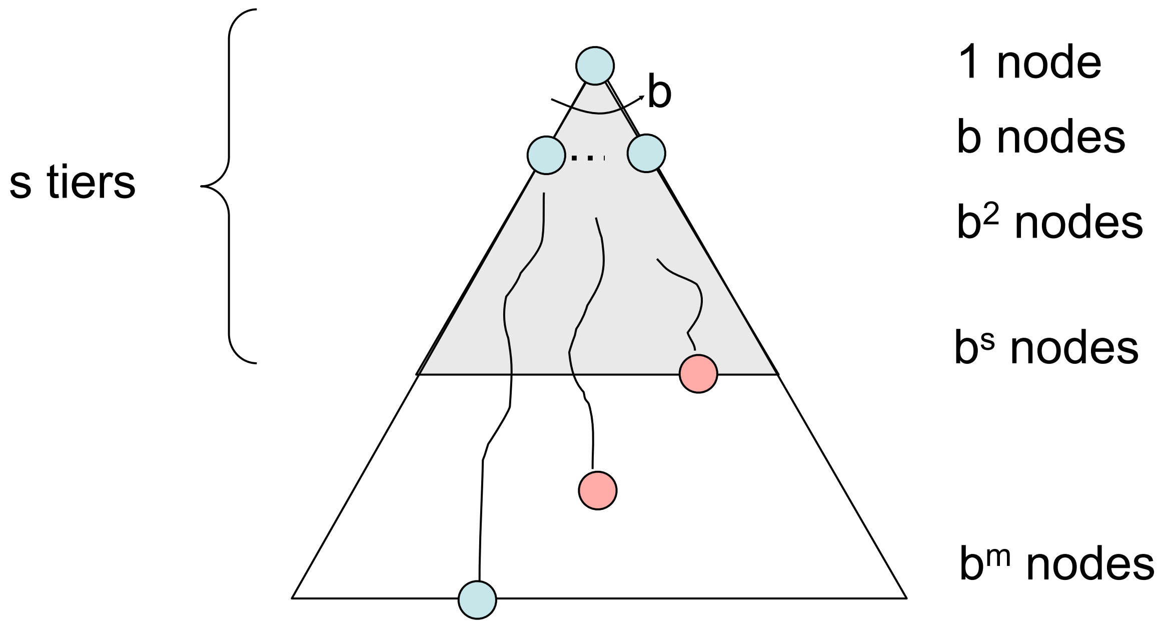


Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$

- When is DFS optimal?

Breadth-First Search (BFS)

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N*	$O(b^{s+1})$	$O(b^s)$



- When is BFS optimal?

Comparisons

- When will BFS outperform DFS?

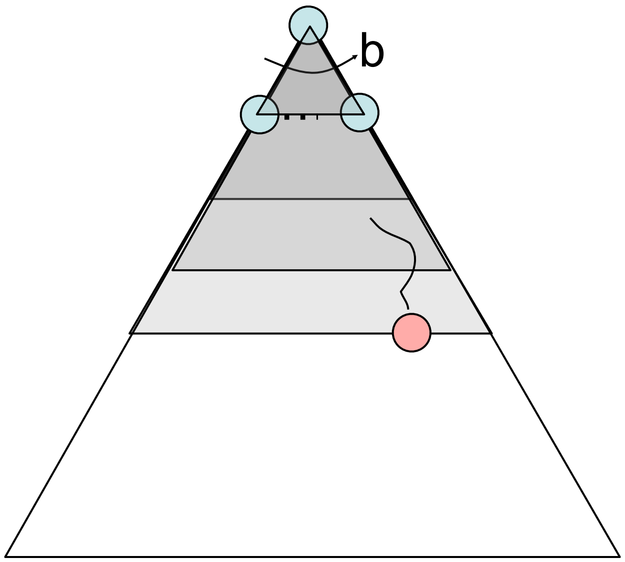
- When will DFS outperform BFS?

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

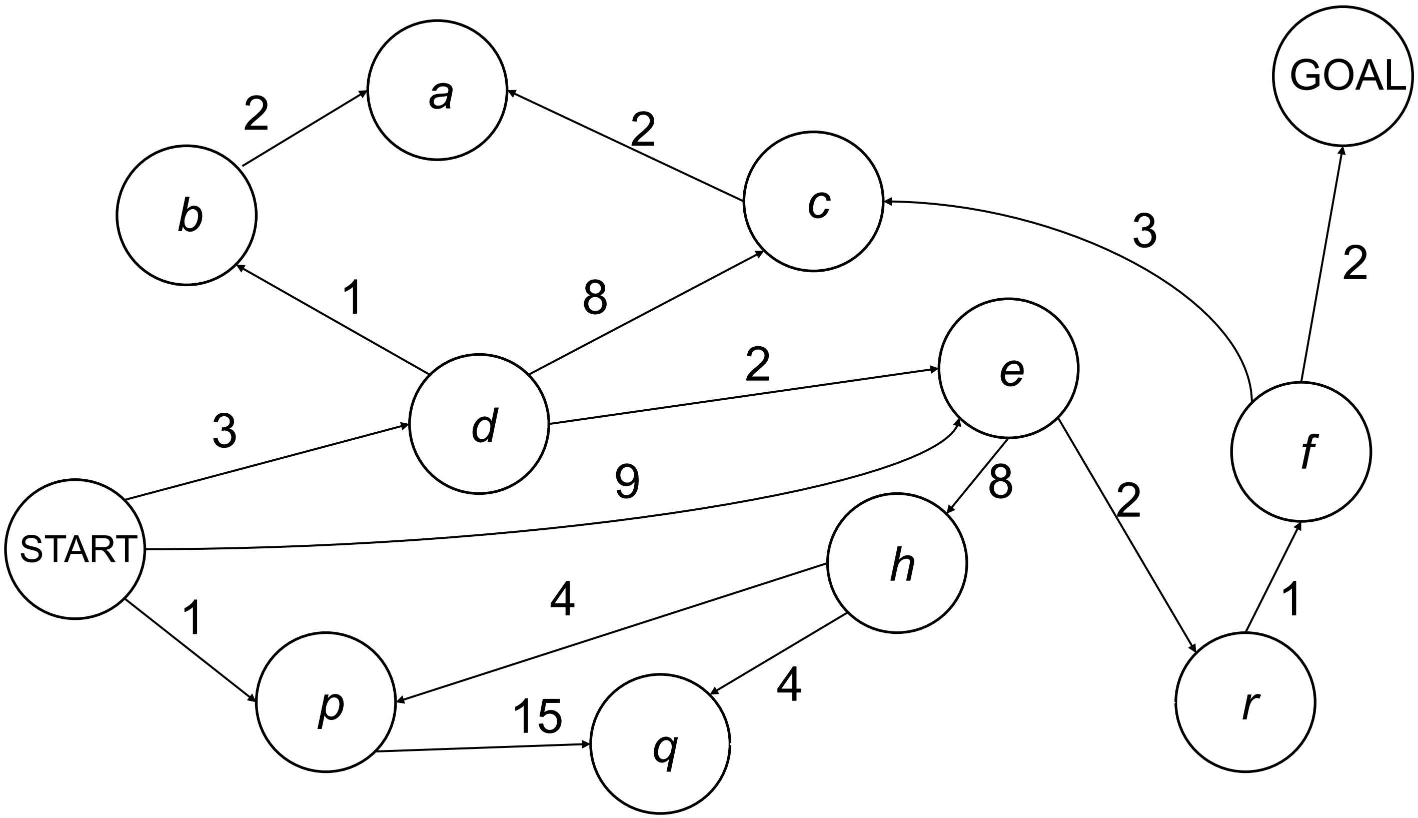
1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N*	$O(b^{s+1})$	$O(b^s)$
ID		Y	N*	$O(b^{s+1})$	$O(bs)$

Costs on Actions



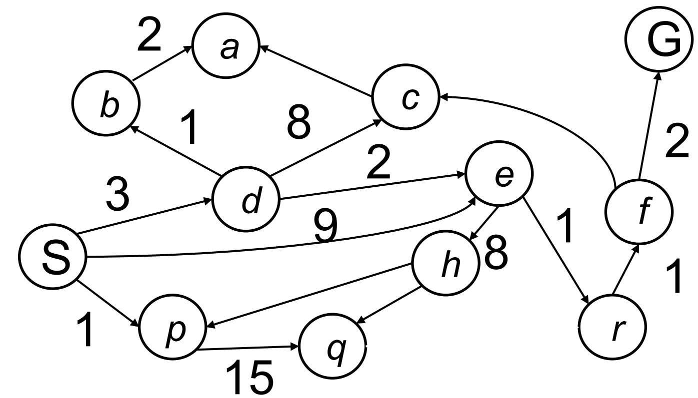
Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

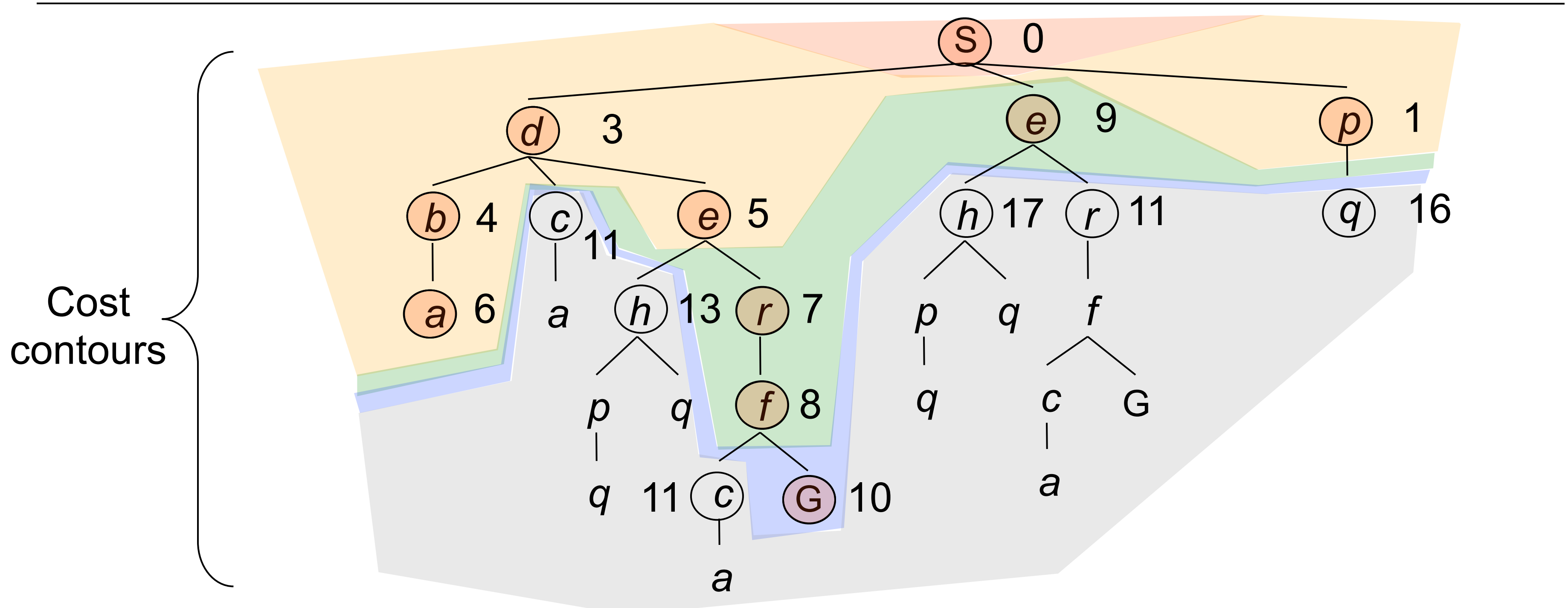
We will now cover an algorithm which does find the least-cost path.

Uniform Cost Search

Expand cheapest node first:

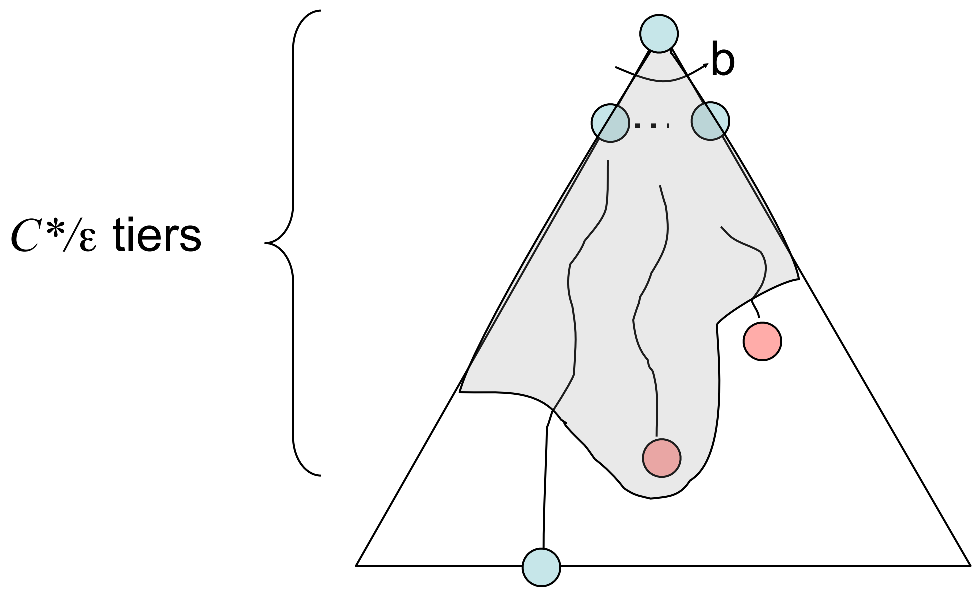
Fringe is a priority queue





Uniform Cost Search

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^{m+1})$	$O(bm)$
BFS		Y	N	$O(b^{s+1})$	$O(b^s)$
UCS		Y*	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$



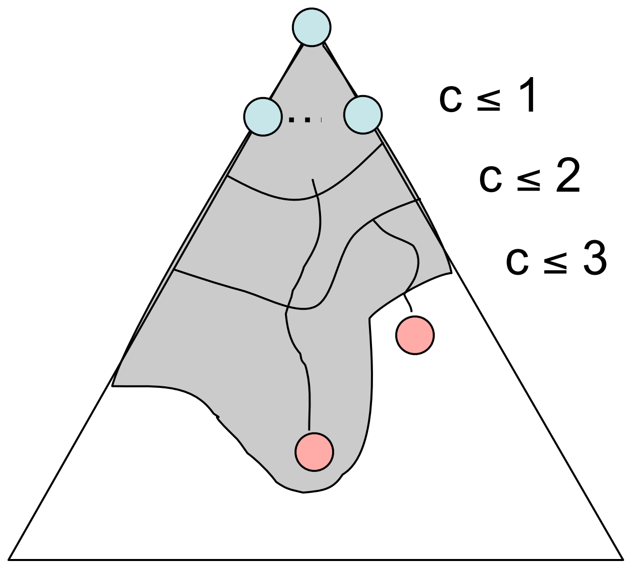
** UCS can fail if actions can get arbitrarily cheap*

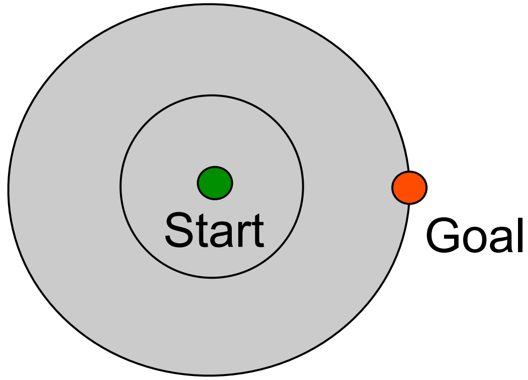
Uniform Cost Issues

- Remember: explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every "direction"
  - No information about goal location
  - "blind" search





Recap: Search

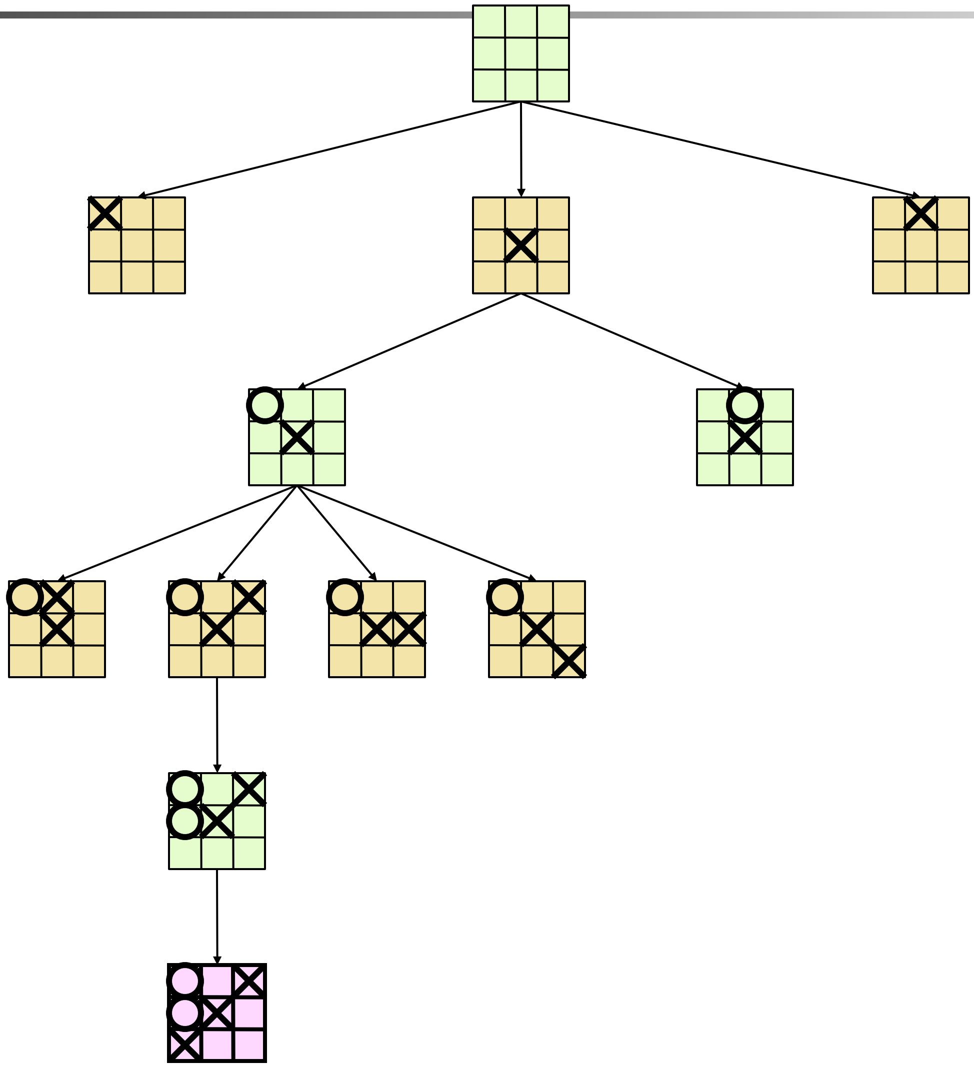
- Search problem:
  - States (configurations of the world)
  - Deterministic transitions: a function from states to lists of (next state, cost) pairs; drawn as a graph
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search Algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)

Game-tree search

- Approximate
  - Evaluation functions
  - Anytime algorithms
- Adversarial
  - Minimax search
  - Alpha-Beta pruning

based on work by R Greiner, D Lin, Jean-Claude Latombe, N Nilsson

Partial Game Search Tree for Tic-Tac-Toe



But… in general the search tree is too big to make it possible to reach the terminal states!

Examples:

- Checkers: ~10^{40} nodes
- Chess: ~10^{120} nodes

Evaluation Function of a State

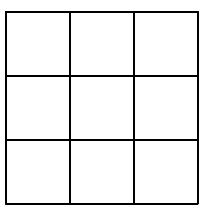
- $e(s) = +\infty$ if s is a win for MAX
- $e(s) = -\infty$ if s is a win for MIN
- $e(s)$ = a measure of how “favorable” is s for MAX

  > 0 if s is considered favorable to MAX

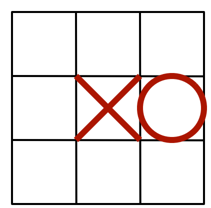
  < 0 otherwise

Example: Tic-Tac-Toe

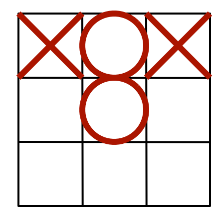
e(s) = number of rows, columns, and diagonals open for MAX
- number of rows, columns, and diagonals open for MIN



8-8 = 0



6-4 = 2



3-3 = 0

Evaluation Function for chess

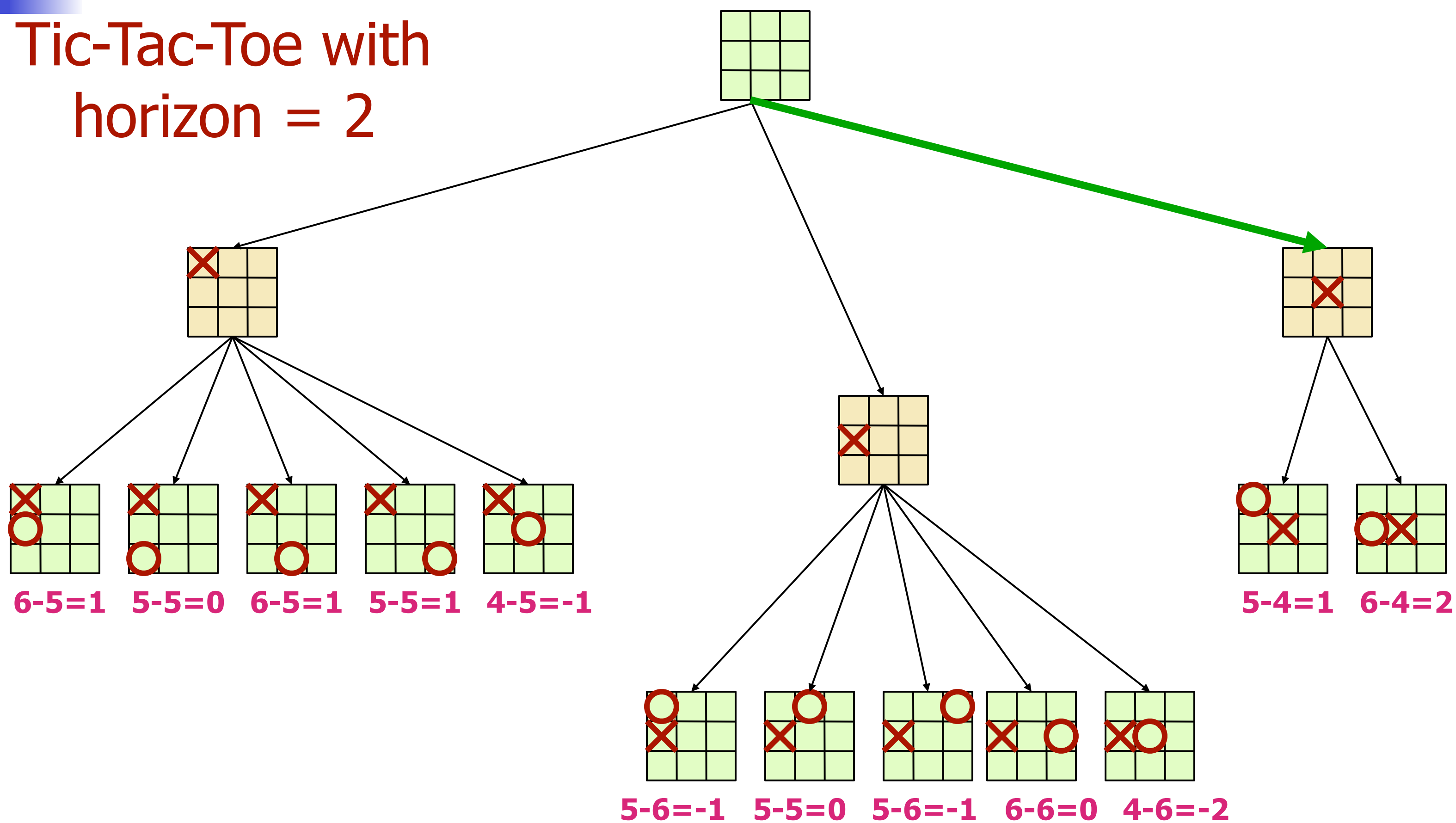
- e(s) = weighted sum of feature

$$\mathrm{e}(s) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi} \doteq \theta_1\phi_1 + \theta_2\phi_2 + \cdots + \theta_n\phi_n$$

- Features
  - # of white pawns – # of black pawns
  - # of white bishops – # of black bishops
  - # of white rooks – # of black rooks
  - …
- Weights
  - 1 for pawns, 3 for bishops, 5 for rooks

Example

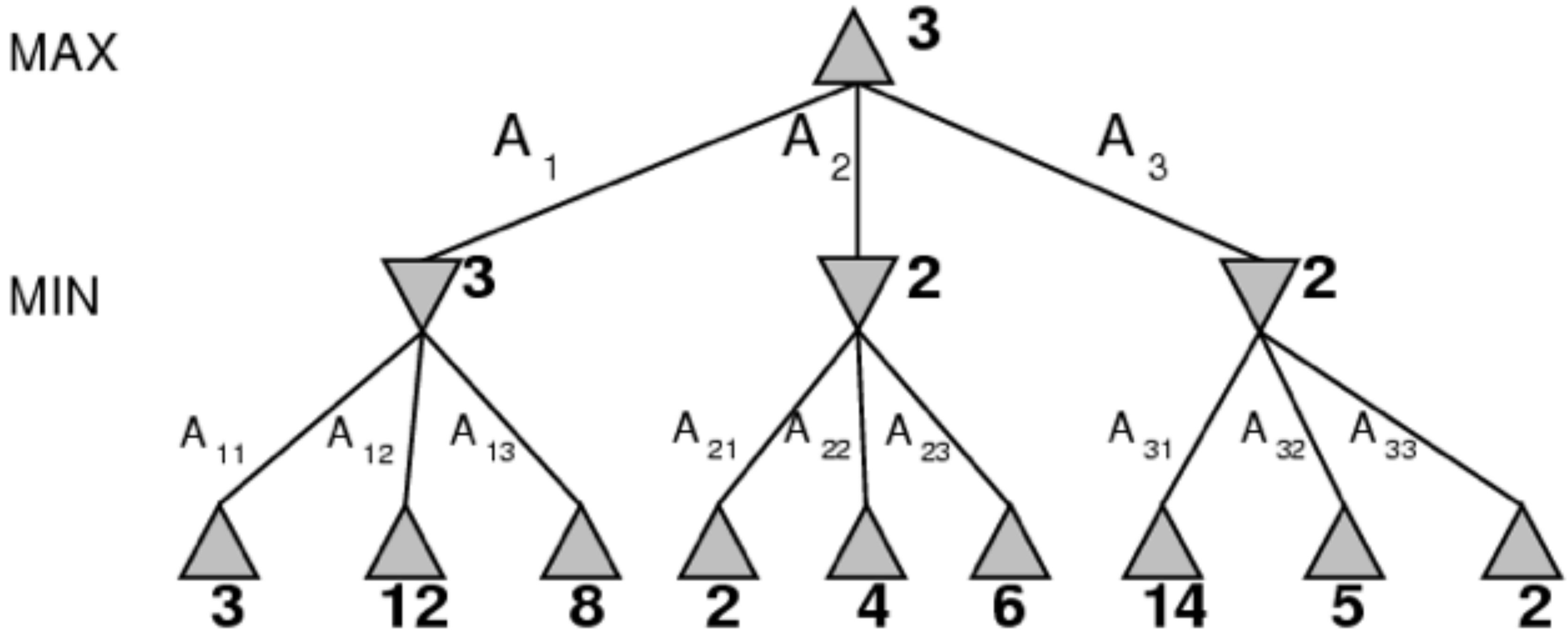
Tic-Tac-Toe with horizon = 2



Minimax

- Achieves “Perfect” play for deterministic, perfect-information games
- Idea: choose move leading to position with highest minimax value
  - best achievable payoff against best play

Example: a 2-ply game

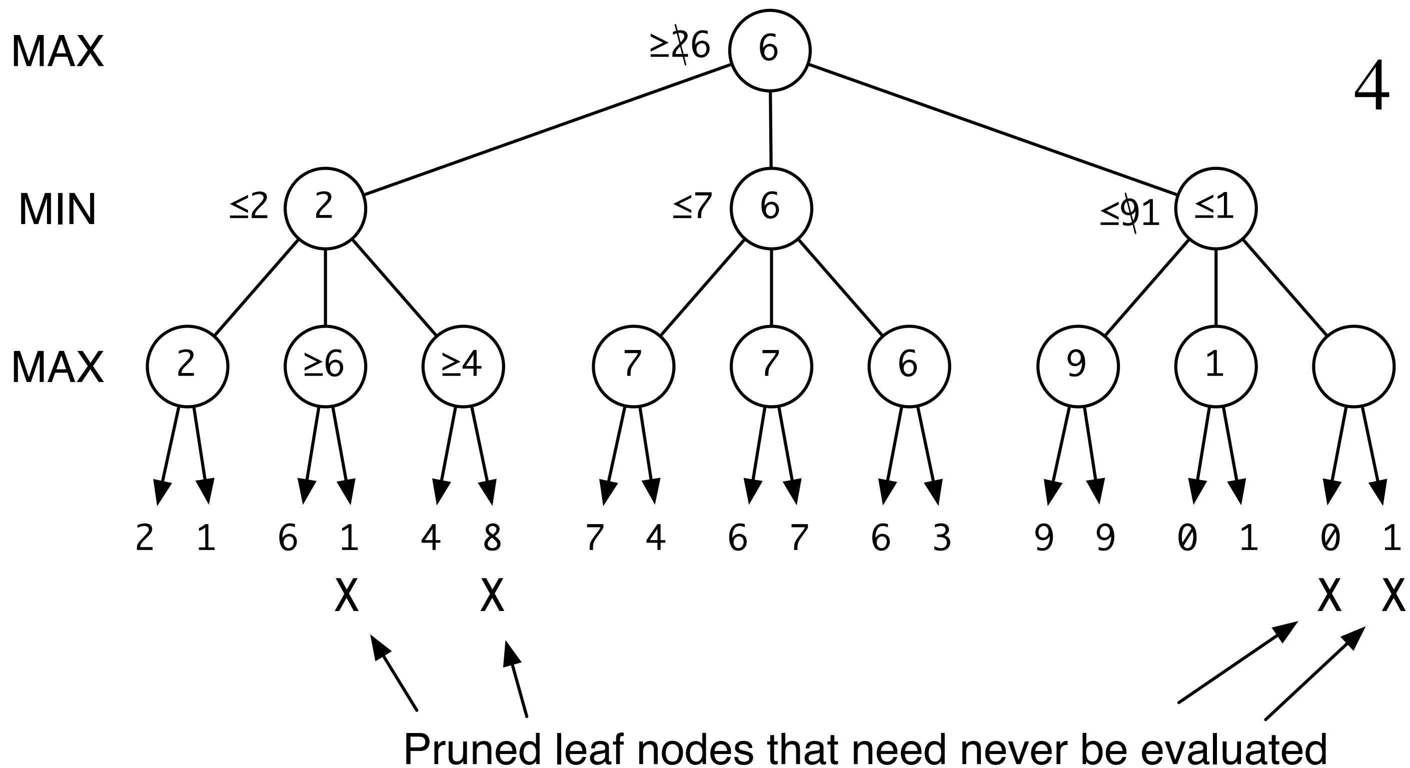


Minimax (back of the envelope)

- Does minimax work in practice?

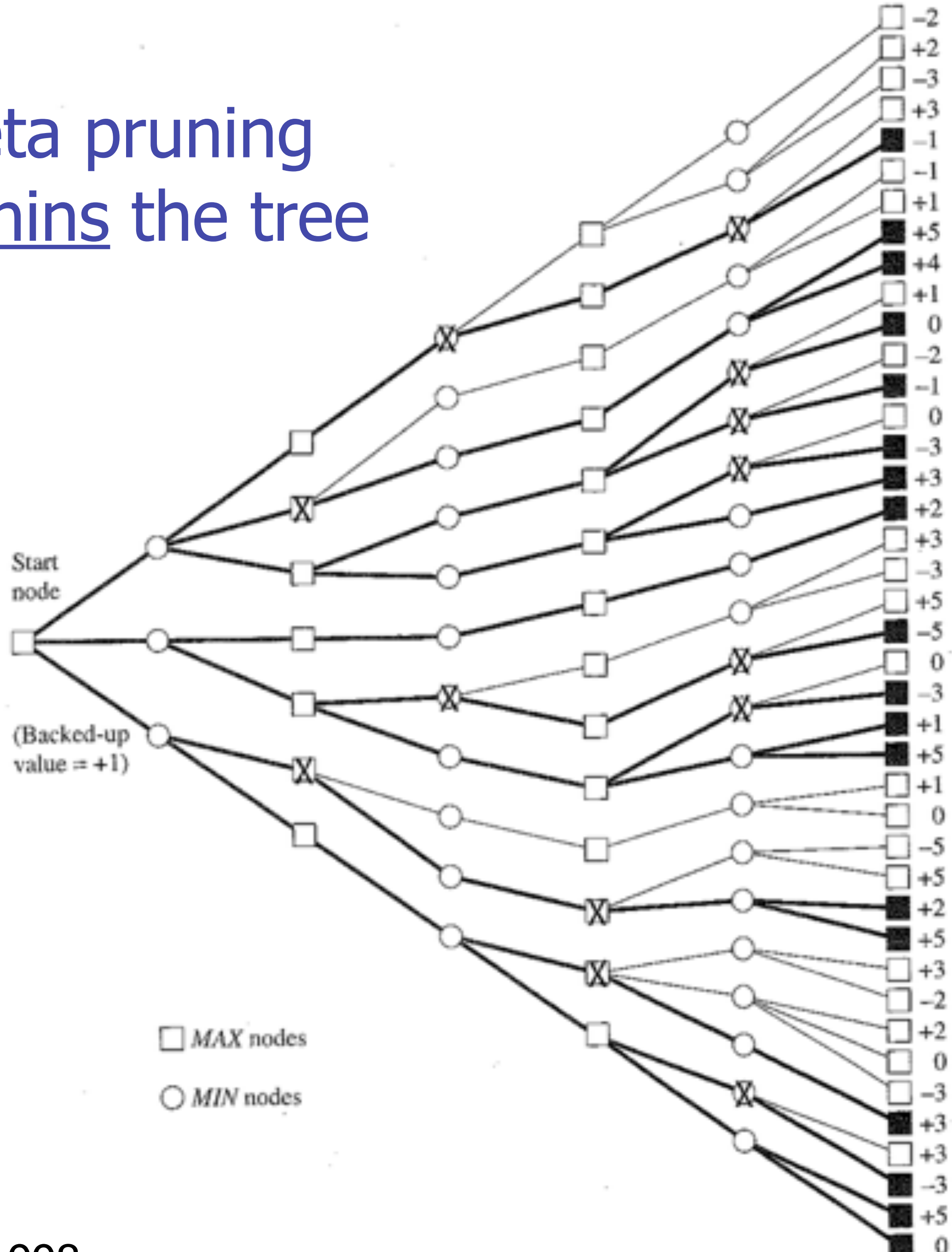
  In chess: we can do about $b^d = 10^6$
  thus, if b=35, then d=4
  but 4-ply lookahead is a hopeless chess player!
- 4-ply = human novice
  8-ply = typical PC, human master
  12-ply = Deep Blue, Kasparov

α-β Pruning



See alpha-beta exercises for more examples

Alpha-beta pruning greatly <u>thins</u> the tree



From Nilsson, 1998

Properties of α-β Search

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering”:
  - time complexity = $O(b^{d/2})$
  - doubles depth of search
  - can easily reach depth 8
    - play good chess!

Summary

- There are lots of different kinds of search
  - With different optimizations and guarantees
- All involve planning – using your knowledge of the worlds dynamics to anticipate the consequences of your action, and then picking the best
- All search involves computing how good it is to be in each state
  - And benefits from a good initial estimate