Search II

Advanced Analysis of Algorithms – COMS3005A

Steve James

Steven.James@wits.ac.za

AI: A Modern Approach (3rd ed), Sections 3.5, 3.6, 5.1 – 5.4

Previously

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Uninformed strategy for node expansion
  - Shallowest
  - Deepest
  - Smallest total cost from root

Generic, problem-independent!

Now

- What if we know something about the problem?
    - How can we incorporate knowledge?
    - Task-specific expansion strategy?
- What if we have an adversary?
    - Can't just find a goal
        - Opponent can prevent us from doing so!

From UCS to heuristics

- Recall: UCS is like BFS with priority queue
- Nodes ordered by priority from smallest to largest
  - Priority is cost estimate $f(n)$

Evaluation function

- In UCS, $f(n) = g(n)$, where g is cost of reaching n from root
- Heuristic function $h(n) =$ estimate of cost from n to goal
- g is backward cost (start to node), h is (estimated) forward cost (node to goal)

Greedy best-first search

- UCS is $f(n) = g(n)$
- GBFS is $f(n) = h(n)$
  - Order nodes by estimated cost to goal

- Where does this estimate come from?
  - Domain knowledge!

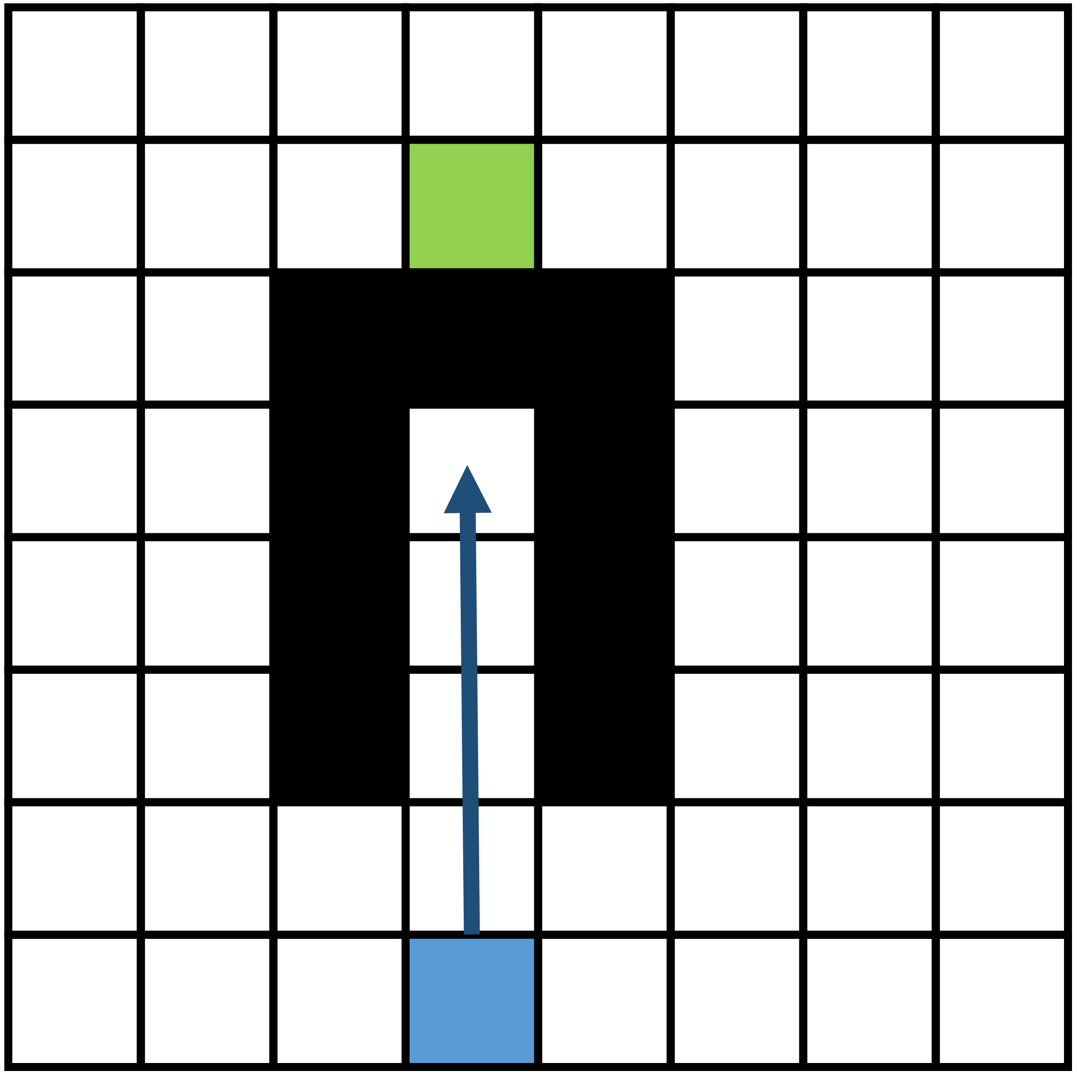
- e.g. In navigating task with location x

$$h(x) = |x - g|_2$$

Euclidean distance

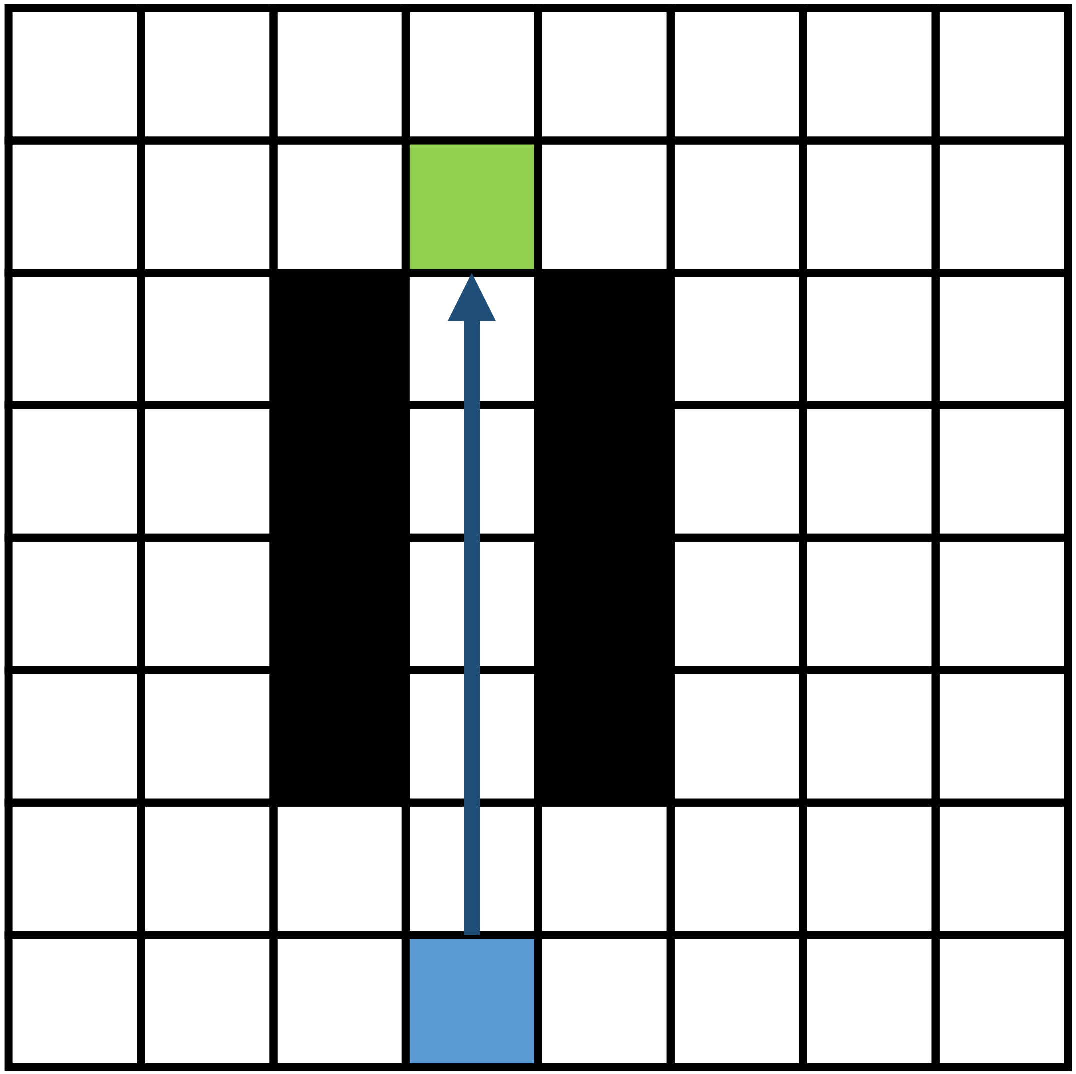
Example

If allowing repeated nodes, will be stuck! (incomplete)



Euclidean distance heuristic

Example



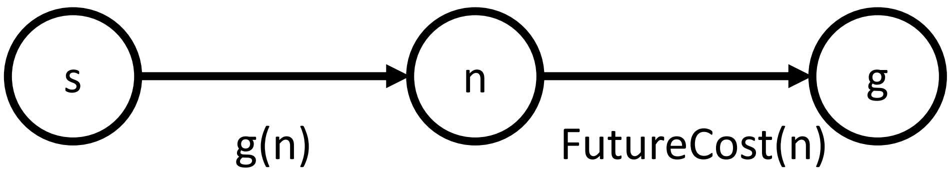
Euclidean
distance
heuristic

GBFS properties

- Incomplete for tree search (i.e. repeated nodes allowed)
  - Complete for graph search in finite space
- Space and time complexity are both $O(b^m)$
  - For max search depth
- But can be reduced substantially depending on heuristic and problem

A* search

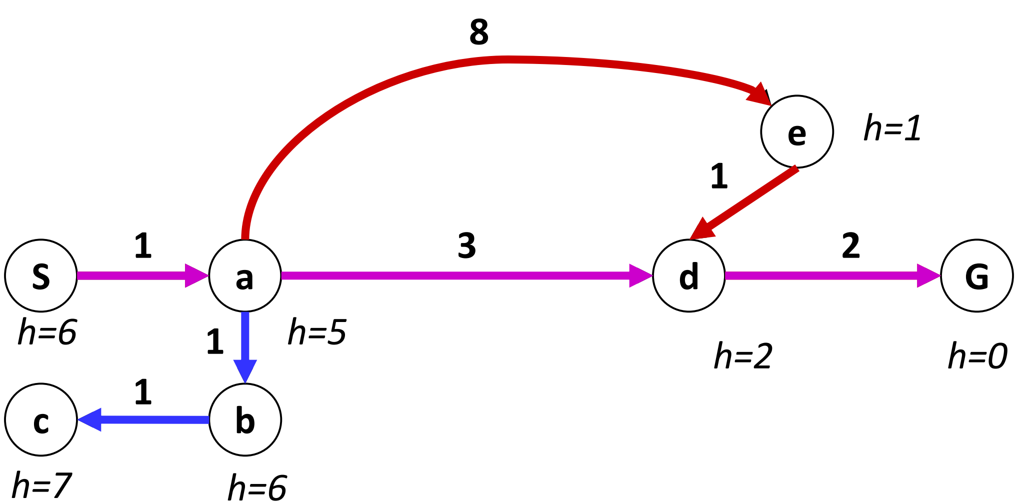
- UCS: $f(n) = g(n)$
- GBFS: $f(n) = h(n)$

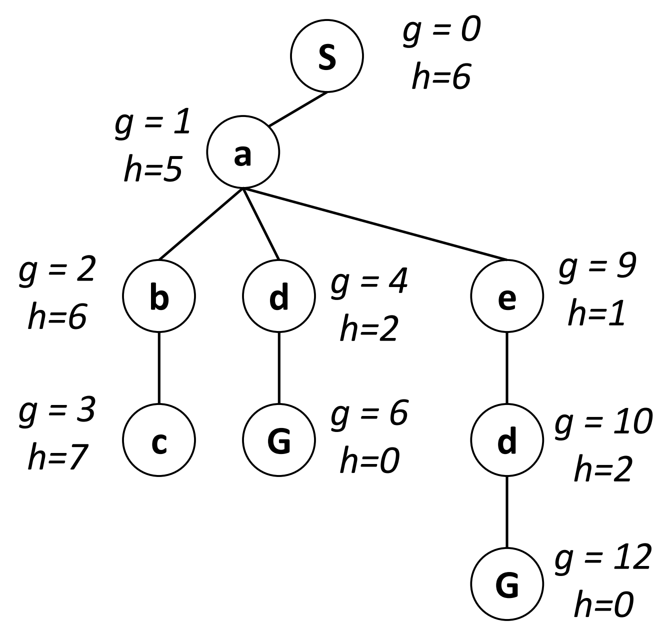


- Ideally: explore in order of $g(n) + FutureCost(n)$
- A*: explore in order $f(n) = g(n) + h(n)$
  - h is estimate of future cost
- Implement A* as UCS with different priority!

A* vs UCS vs GBFS

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



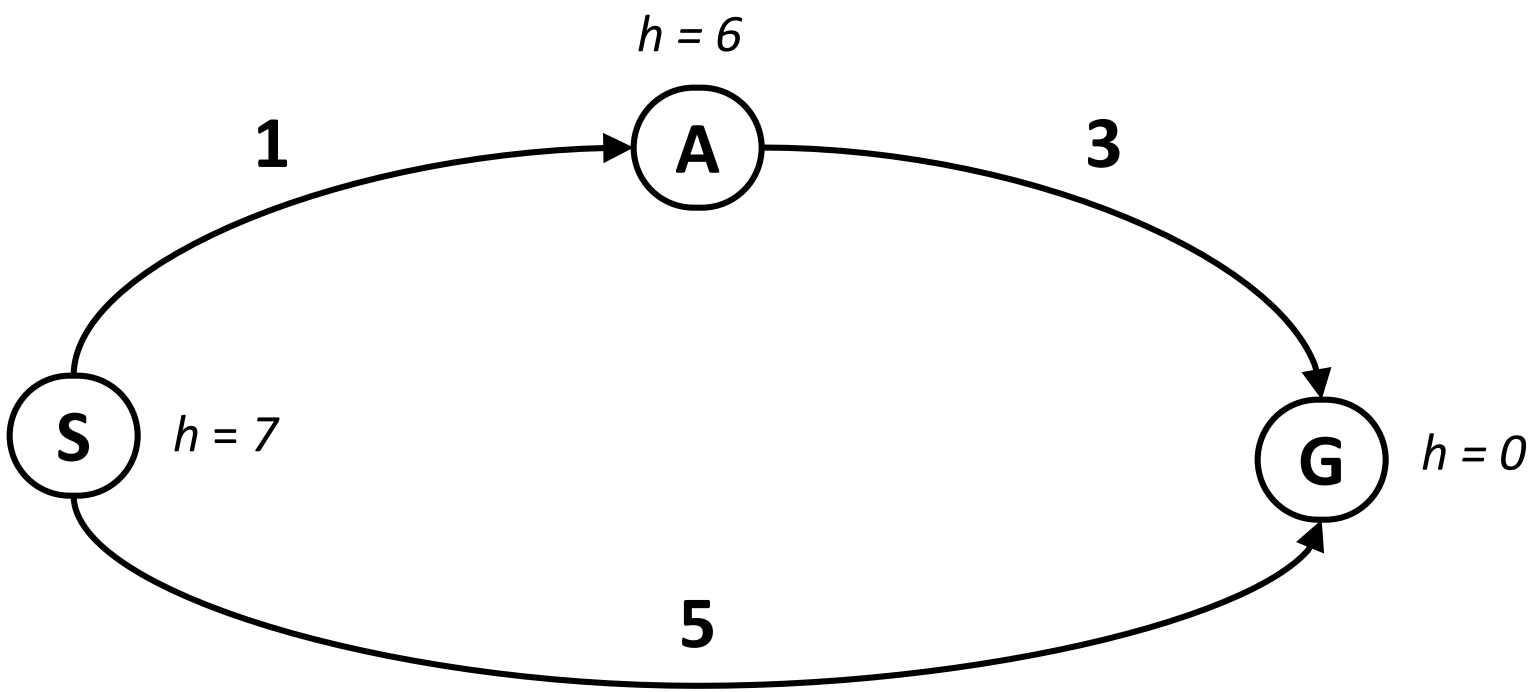


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

Optimality of A*

- Depends on heuristic



- Fails! Bad cost goal < estimated good cost

Heuristics in A*

- h admissible if $h(n) \leq TrueFutureCost(n)$
  - We never overestimate the cost!
  - $f(n) = g(n) + h(n)$ means we never overestimate cost from start to goal through n
  - Optimistic: estimates cost as less than it actually is

- h is consistent if $h(n) \leq c(n, a, n') + h(n')$
  - All consistent heuristics are admissible

Triangle inequality

- A* tree search is optimal if h is admissible
- A* graph search is optimal if h is consistent

Properties of A*

- A* is complete, optimal
- A* is optimally efficient w.r.t. heuristic
  - No other algorithm will expand fewer nodes
- Time complexity is $O\left(b^{\epsilon d}\right)$
  - ϵ is relative error of heuristic, d is solution depth
- Alternate view, $O((b^*)^d)$ where b^* is effective branching factor
  - A* doesn't need to consider certain nodes (prunes) and so branching factor is reduced!
- Since exponential, memory is biggest issue
  - Can do iterative deepening equivalent (IDA*)

Given appropriate heuristic

Like BFS

Creating heuristics

- Main challenge for A*: how to make good heuristics that are admissible/consistent?
- For navigation, straight-line path is good
    - Can never be faster than that!
- Could learn heuristics from data (e.g. machine learning/precomputed database lookups)
- Use relaxations – solve an easier, less constrained version of a problem
    - E.g. Relaxed version of a maze → remove all the walls!

8-puzzle relaxation

5	4	
6	1	8
2	3	7

→

1	2	3
4	5	6
7	8	

- Relaxation 1: allow tiles to be placed anywhere directly
  - i.e. heuristic is number of misplaced tiles
  - Clearly underestimates true cost of moving tiles
- Relaxation 2: allow tiles to be slid around, ignoring other tiles
  - i.e. heuristic is sum of distances of each tile to its final location

Effect of heuristics

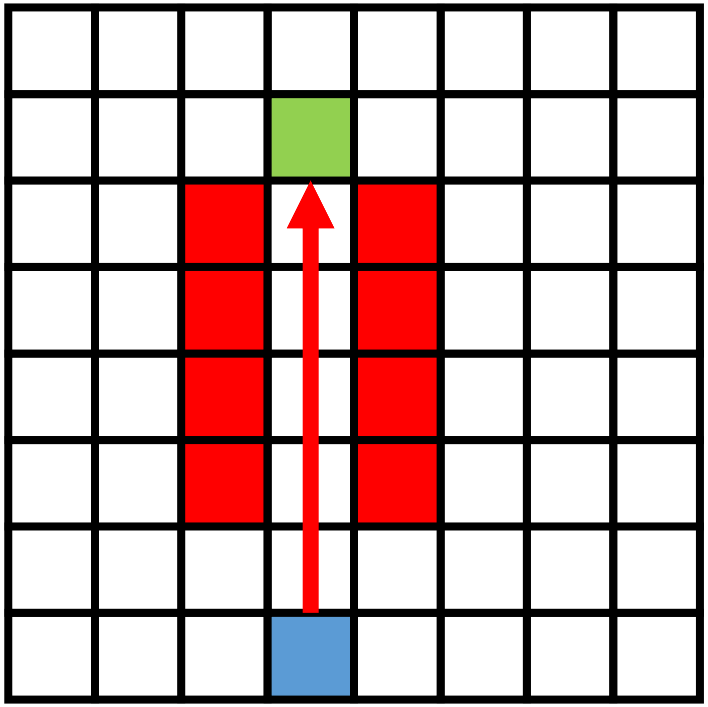
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
Relax 1	13	39	227
Relax 2	12	25	73

- Tradeoff between quality of estimate and work per node
  - Closer heuristic is to true cost, fewer nodes are expanded…
  - But more work to compute heuristic per node!

Adversarial search

- So far, aim was to reach goal with min cost
- But what if another agent is trying to stop us?

- Imagine trying to reach goal
- But every 3 steps, opponent can push us into adjacent cell
- Must take into account their actions

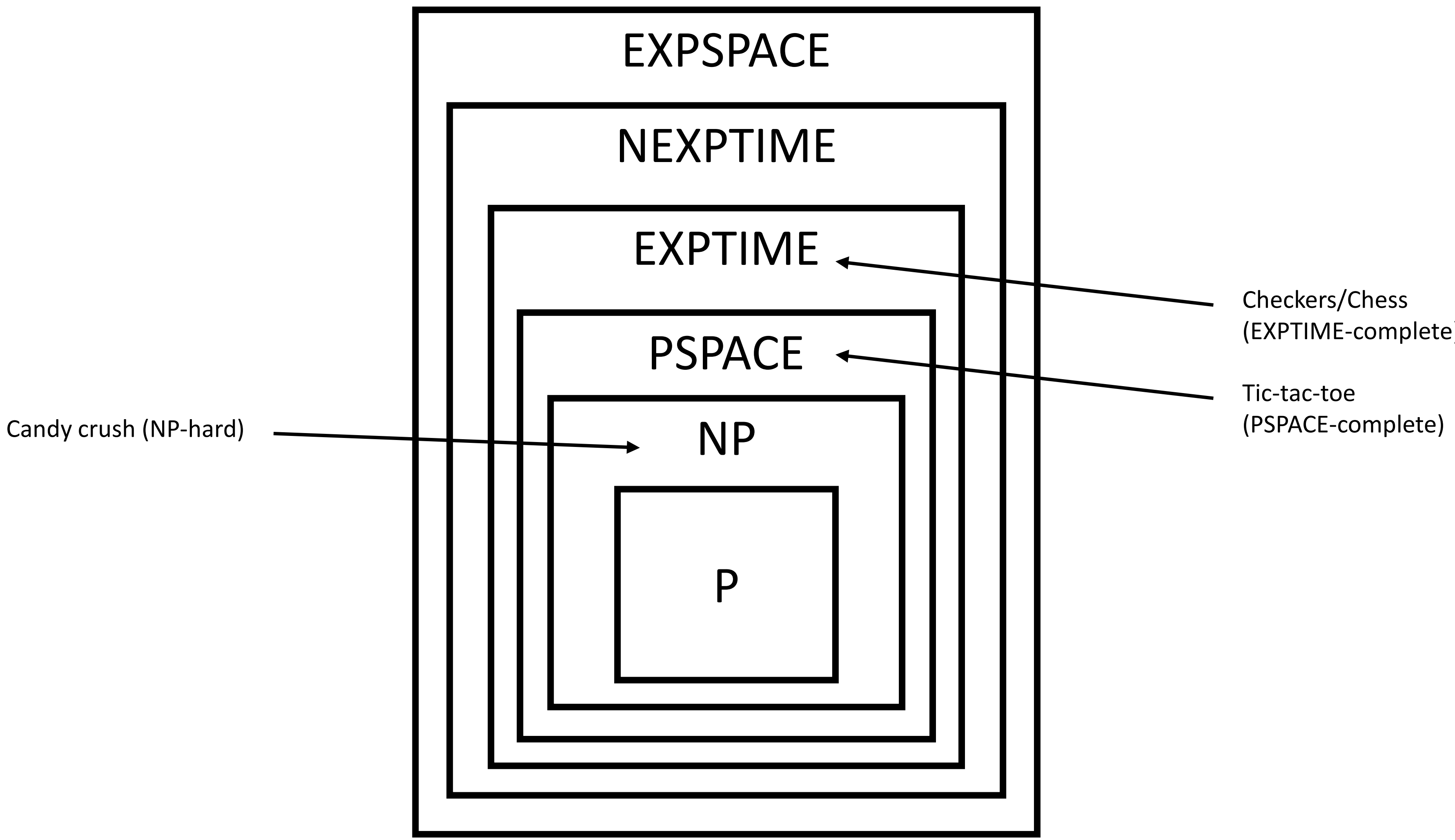


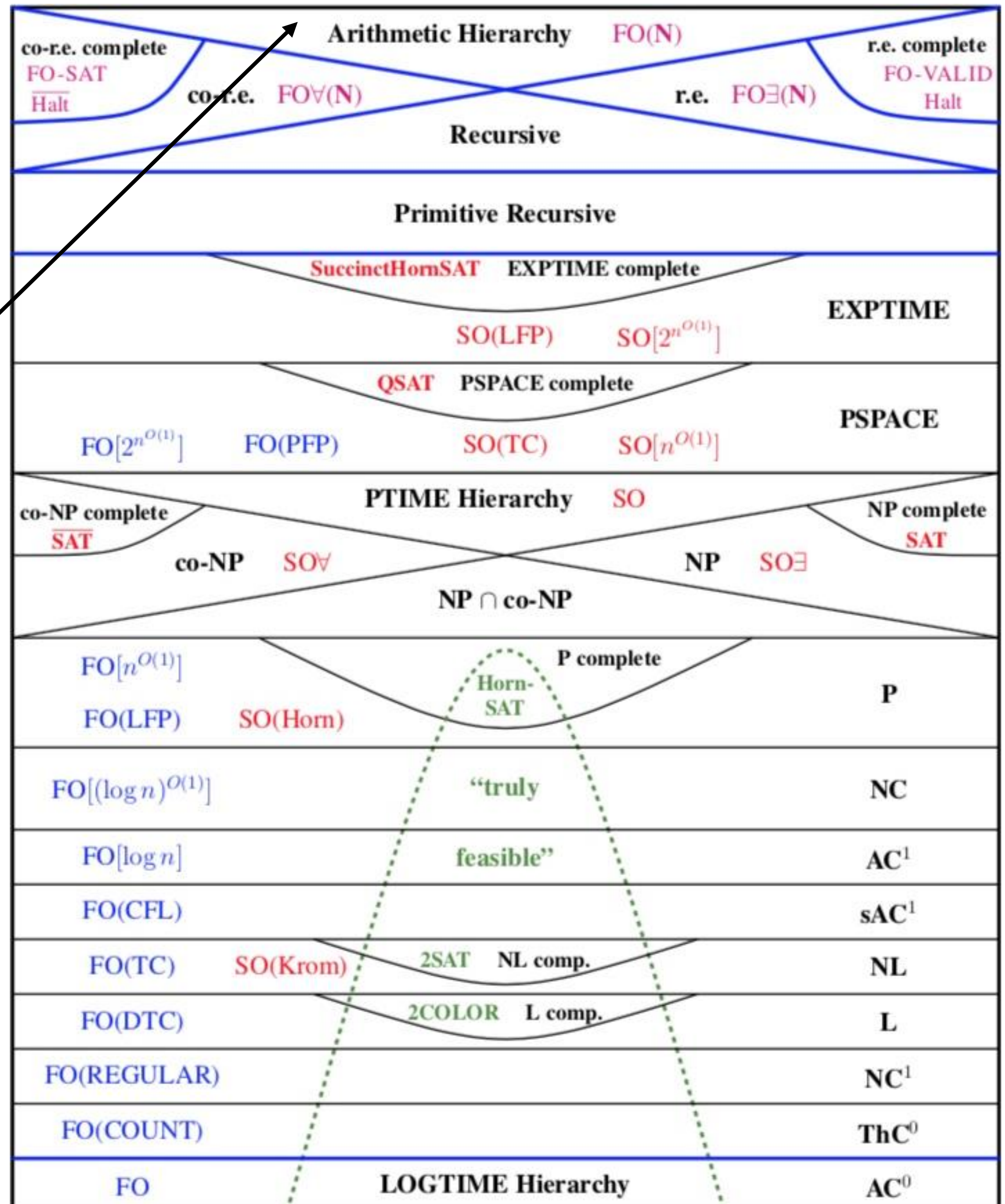
Games are big

- Tic-tac-toe $\sim 10^3$
- Connect Four $\sim 10^3$
- English draughts $\sim 10^{23}$
- Othello $\sim 10^{28}$
- Chess $\sim 10^{44}$
- Shogi $\sim 10^{71}$
- # atoms in observable universe $\sim 10^{82}$
- Twixt $\sim 10^{140}$
- Go (19x19 board) $\sim 10^{170}$

Size of state space
(reachable states)

Games are hard





Magic the Gathering
(AH-hard)

Zero-sum games

Will consider 2 players, but extends to n

- Assume two players, competitive
  - Player 1 wins, player 2 loses and vice versa
- Game is defined by:
  - Initial state
  - $Player(s)$: whose turn it is
  - $Actions(s)$: available actions
  - $Result(s, a)$: successor function or transition model
  - $Terminal(s)$: is the game over/state terminal
  - $Utility(s, p)$: the value for the game ending in s for player p
- Zero sum game: sum of utilities for all players is constant: e.g. Win = +1, Draw = 0, Loss = -1

Two player, zero-sum

- Two players, MAX and MIN
- We are MAX, try to maximise utility
- Opponent is MIN, tries to minimise utility
- Denote $V(s)$ as utility at a given state
  - Utility is known at terminal states
- Start at root node, expand tree
  - Players alternate turns
  - At level 0, us to play. Level 1, them to play, etc
- We want to compute optimal play, assuming our opponent is also optimal

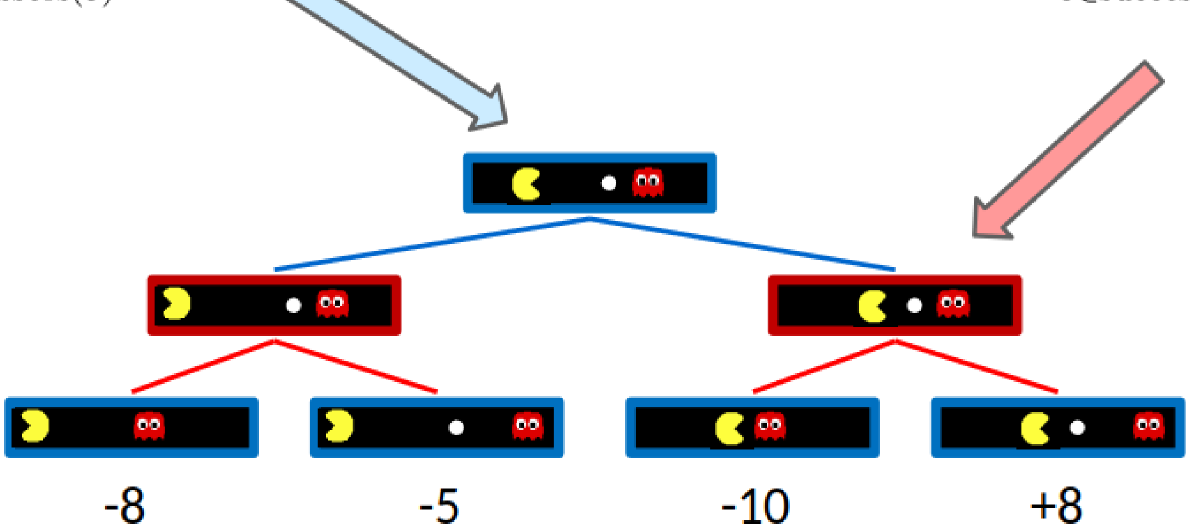
Each level is called a ply

Example

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

Calculating minimax

- Want to calculate $V(s)$ for all s
- If s is terminal, use utility function directly
- else if player to play is MAX:
  - Value is best maximising value at state
- else player to play is MIN:
  - Value is best minimising value at state

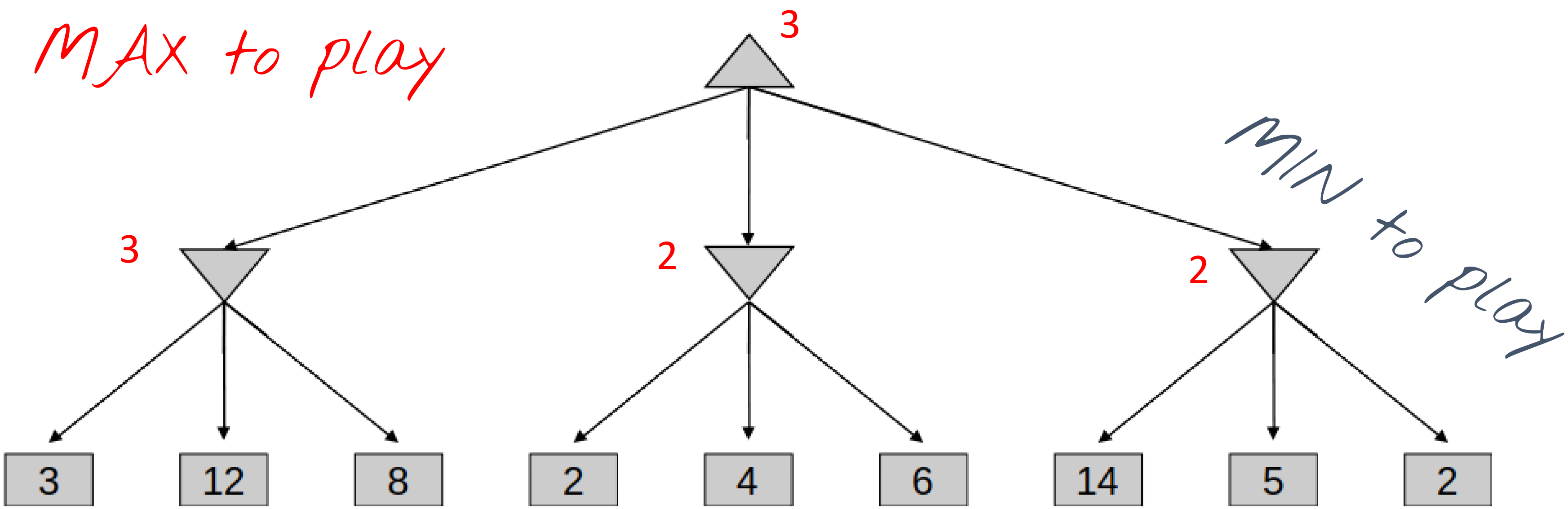
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

$$
\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}
$$

Example

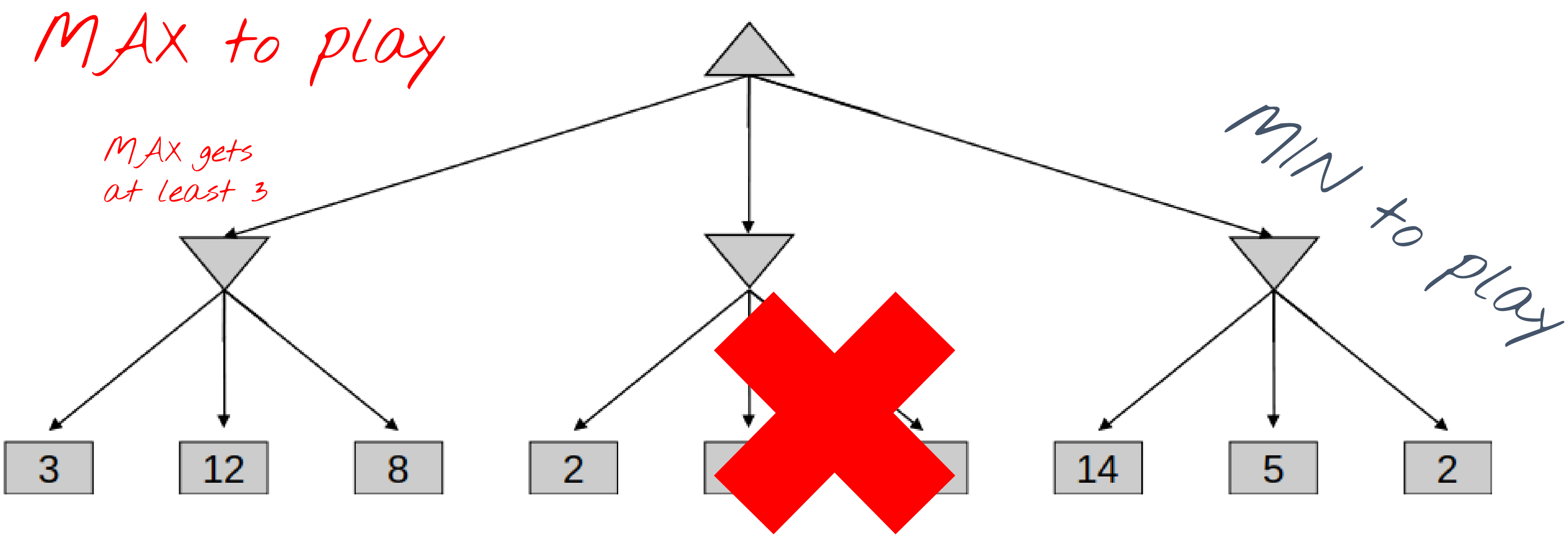


Minimax properties

- Like DFS:
  - Time: $O(b^m)$
  - Space: $O(bm)$

- But instead of searching for single goal, we need to exhaustively try everything!
  - And we only get value at leaf nodes
- Chess, for e.g., $b \sim 20, m \sim 70$
  - Exact solution infeasible
  - So what do we do?

Pruning the tree



$\alpha\beta$-pruning

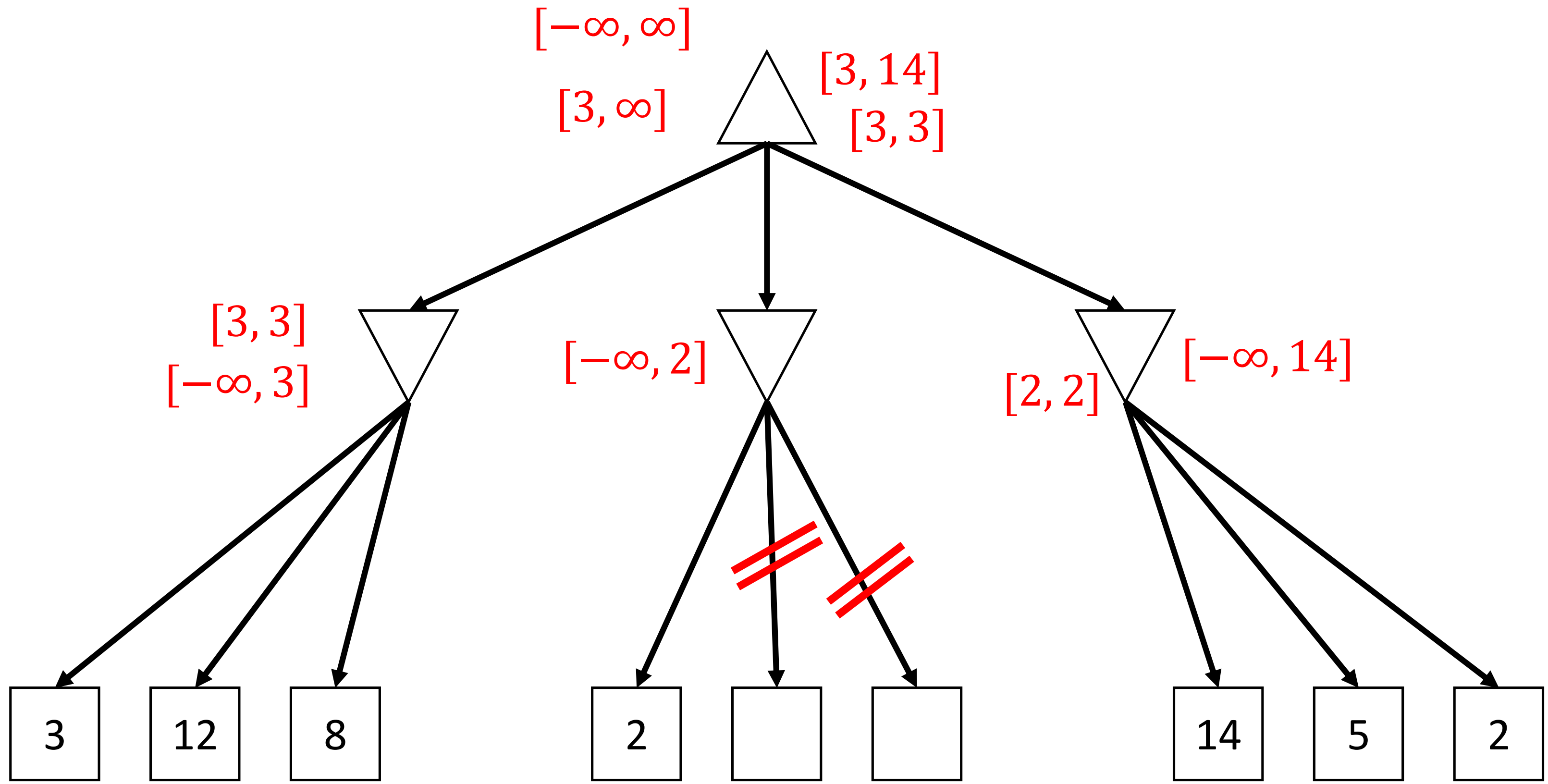
- α – minimum score MAX is guaranteed of
- β – maximum score MIN is guaranteed of

Worst case guarantees

- If at a given node, $\beta < \alpha$
  - Then MIN can guarantee a score that makes MAX sad ☹
  - So MAX will never go down this road
  - No need to expand rest of node's children!

Could be pruning entire subtrees!

- Symmetric argument for other way around
- If children are expanded in optimal order, complexity is halved: $O(b^{m/2})$



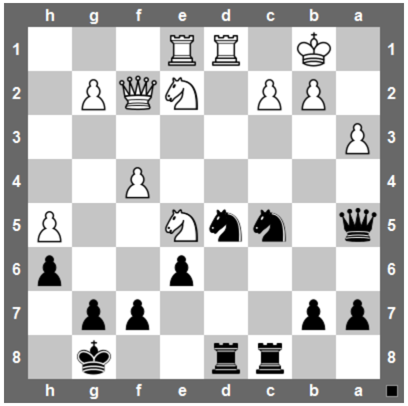
α – minimum score MAX is guaranteed of
β – maximum score MIN is guaranteed of

Depth-limited search

- Even with pruning, can’t reach leaf nodes in real games
- So must limit depth of search
  - Must replace utility function with estimate (like heuristic in A*)
  - No longer optimal
- More plies = better performance
- Given time budget, use IDS!

Evaluation functions

- Estimate of utility of non-terminal state
- Ideally: want actual minimax value of state
  - But this is unknown
- In practice: use domain knowledge
  - E.g. $\text{eval}(s) = w_1(|pawns_w - pawns_b|) + w_2(|bishops_w - bishops_b|) + \cdots$



- Tradeoff between complexity vs depth
  - More complex eval function may be more accurate, but longer to compute → less time to search deeper
    - Stockfish: fast eval function, huge depth
    - Komodo: slow, complex eval function, less depth

Other improvements

- Base algorithm of $\alpha\beta$ + IDS + eval function
- Transposition tables: stored previous states and their evals
- Aspiration windows: pretend that the $\alpha\beta$ window is smaller than it is
- Evaluation functions optimised from data (machine learning etc)
- Move ordering: try certain classes of moves first (e.g. captures, then regular moves)

Summary

- For single-agent search:
  - Uninformed node expansion (DFS/BFS/UCS)
  - Informed with heuristics (GBFS/A*)
    - A* optimal, but need admissible/consistent heuristics
    - Heuristics from problem relaxation
- Adversarial search:
  - Minimax for optimal play
  - Can use pruning to reduce nodes
  - In practice, must cut-off depth
    - Use evaluation function
- Primary focus of AI for 60 years!