6.034: Introduction to Artificial Intelligence

Adversarial search & games

Lecture 5 Handout

Robert C. Berwick

September 14, 2020

Menu for today

Games: Adversarial search

- ❑ Intelligence & Chess
- ❑ Ways to Play Chess
- ❑ How to search: Minimax algorithm
- ❑ Improving search: Alpha-beta pruning
- ❑ Progressive Deepening
- ❑ Reflections on contemporary chess computers & Gold Star ideas

Games: Adversarial Search

How could a computer play chess?

1. Human-like
2. If then: make most plausible move

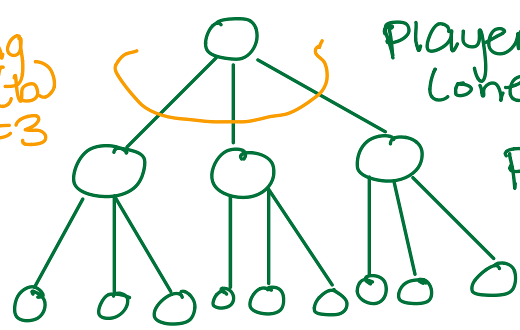
*3. Look ahead & evaluate
   - evaluate moves by points, etc.

score → $S = g(f_1, f_2, \ldots f_n)$

4. British museum algorithm
   - though not that productive

Vocabulary for game trees

branching factor (b)
here, b=3



Player 1 moves
(one turn or 1 "ply")

player 2 moves

depth here,
d=2

leaf nodes $= b^d = 9$

Could we use cloud computing to evaluate all possible chess moves in the same way?

10^{120} moves

10^{80} atoms in universe

$\pi \times 10^{7}$ seconds/year

10^{9} nanoseconds/second

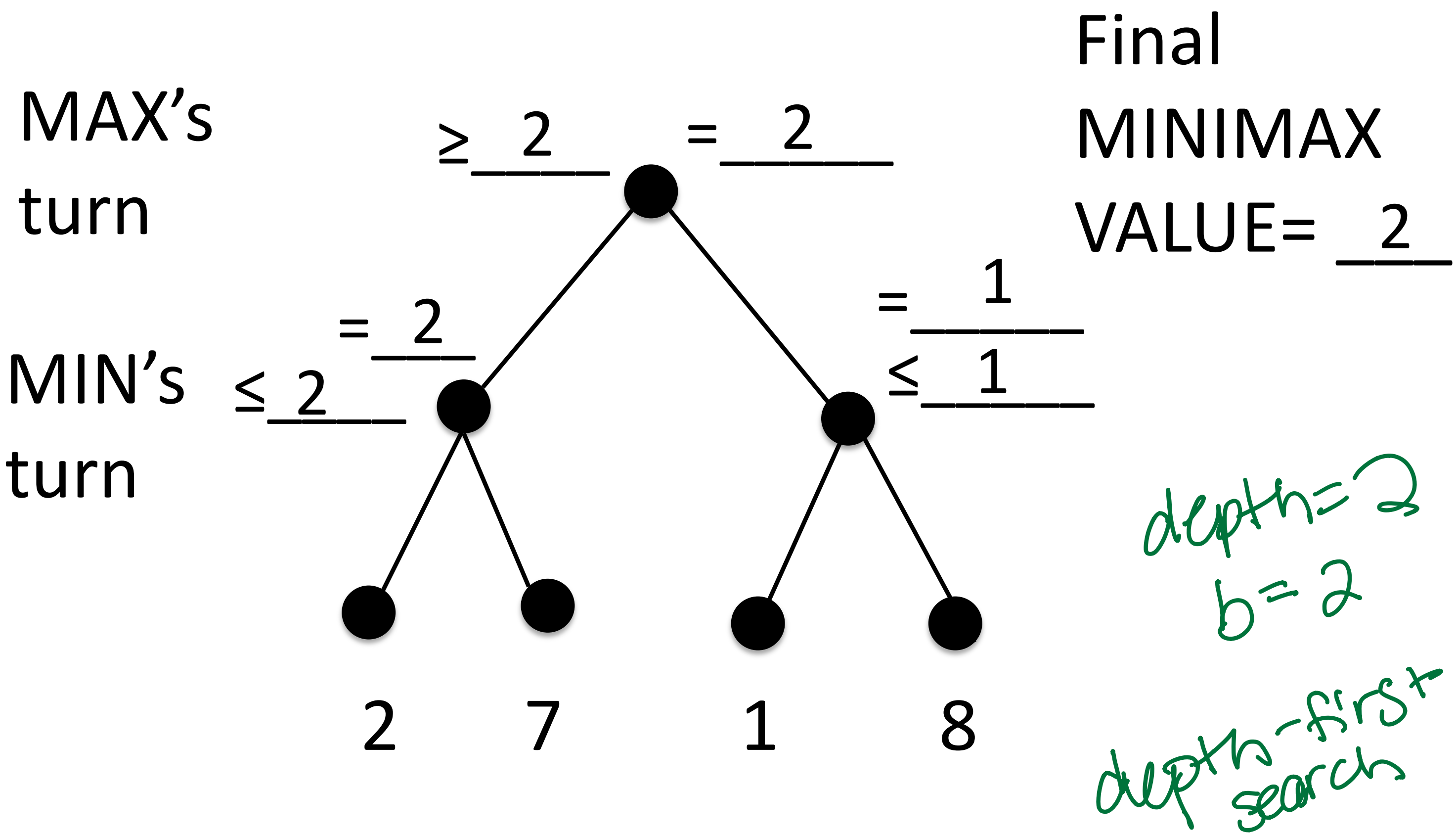
10^{10} years since Big Bang

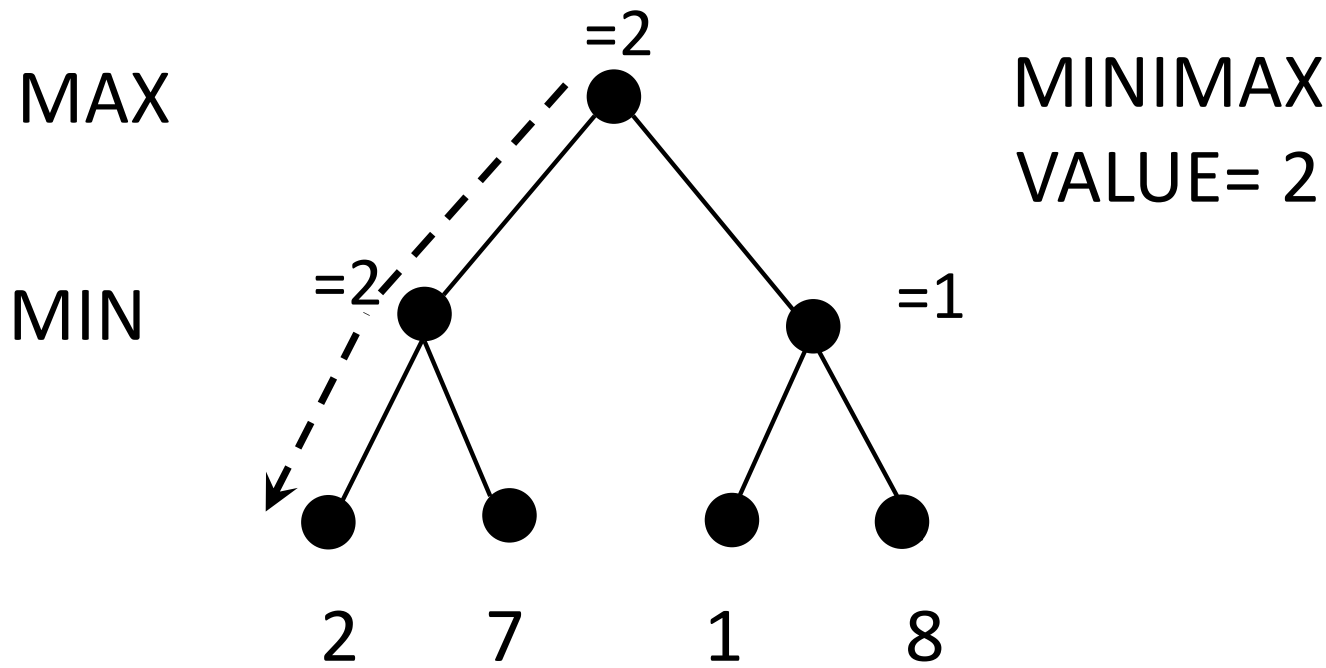
$\approx 10^{26}$ nanoseconds x 10^{80} atoms

$\approx 10^{106}$ ops, each atom operating @ nanosecond speeds

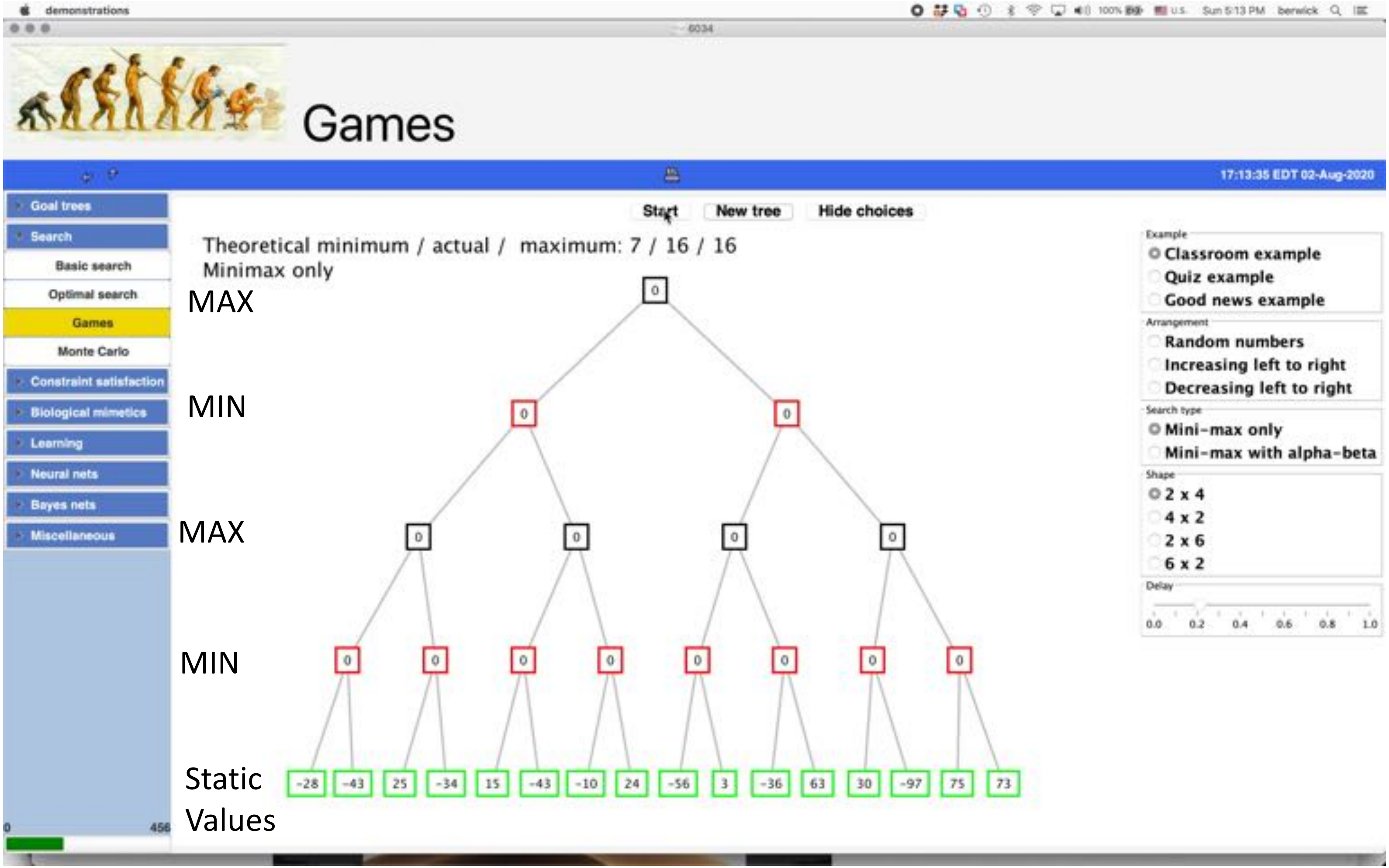
Not enough compute power for 10^{120} moves!

Minimax algorithm for searching game tree to find optimal move sequence





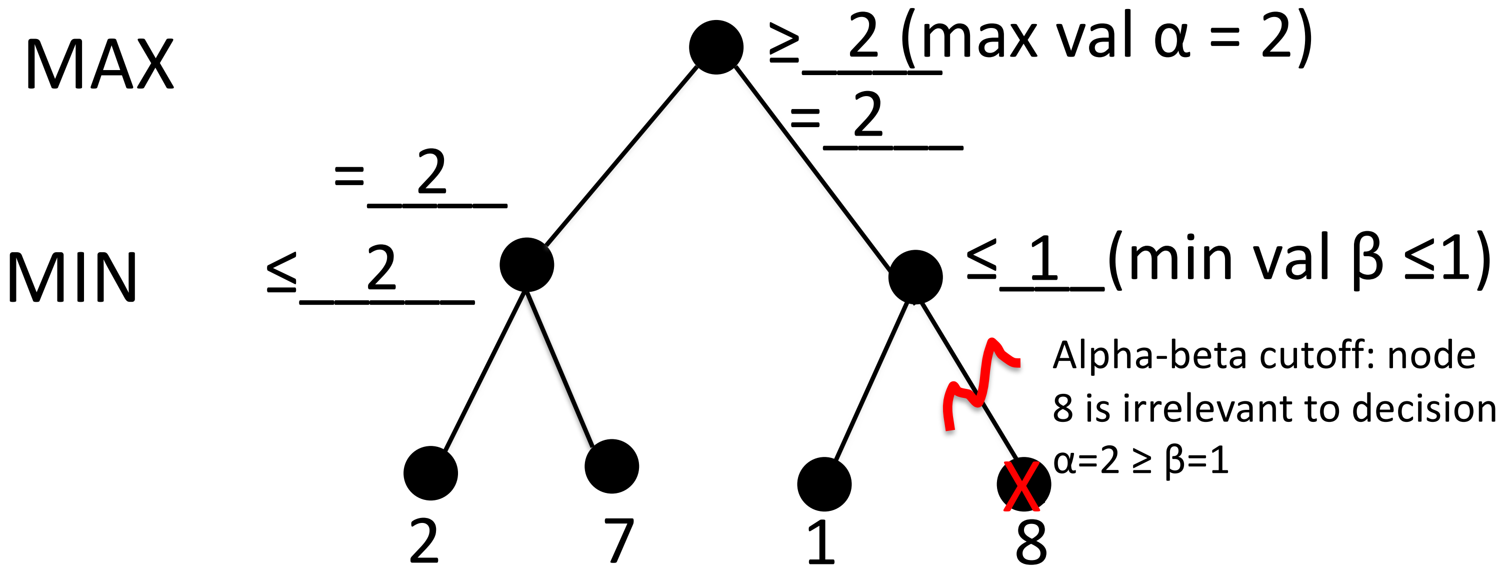
Note that the best minimax path is preserved under any transform of the leaf scores that preserves their rank ordering



Games
Goal trees
Search
Basic search
Optimal search
Games
Monte Carlo
Constraint satisfaction
Biological mimetics
Learning
Neural nets
Bayes nets
Miscellaneous
17:13:35 EDT 02-Aug-2020
Start
New tree
Hide choices
Theoretical minimum / actual / maximum: 7 / 16 / 16
Minimax only
MAX
MIN
MAX
MIN
Static Values
-28 -43 25 -34 15 -43 -10 24 -56 3 -36 63 30 -97 75 73
Example
Classroom example
Quiz example
Good news example
Arrangement
Random numbers
Increasing left to right
Decreasing left to right
Search type
Mini-max only
Mini-max with alpha-beta
Shape
2 x 4
4 x 2
2 x 6
6 x 2
Delay
0.0 0.2 0.4 0.6 0.8 1.0

Minimax with alpha-beta (α-β) pruning

don't have to evaluate all static values of leaf nodes



α value = MAX is guaranteed to gain at least this much, or ≥ α; a floor on MAX's gain.

β value = MIN is guaranteed to lose at most this much, or ≤ β; a ceiling on MIN's loss.

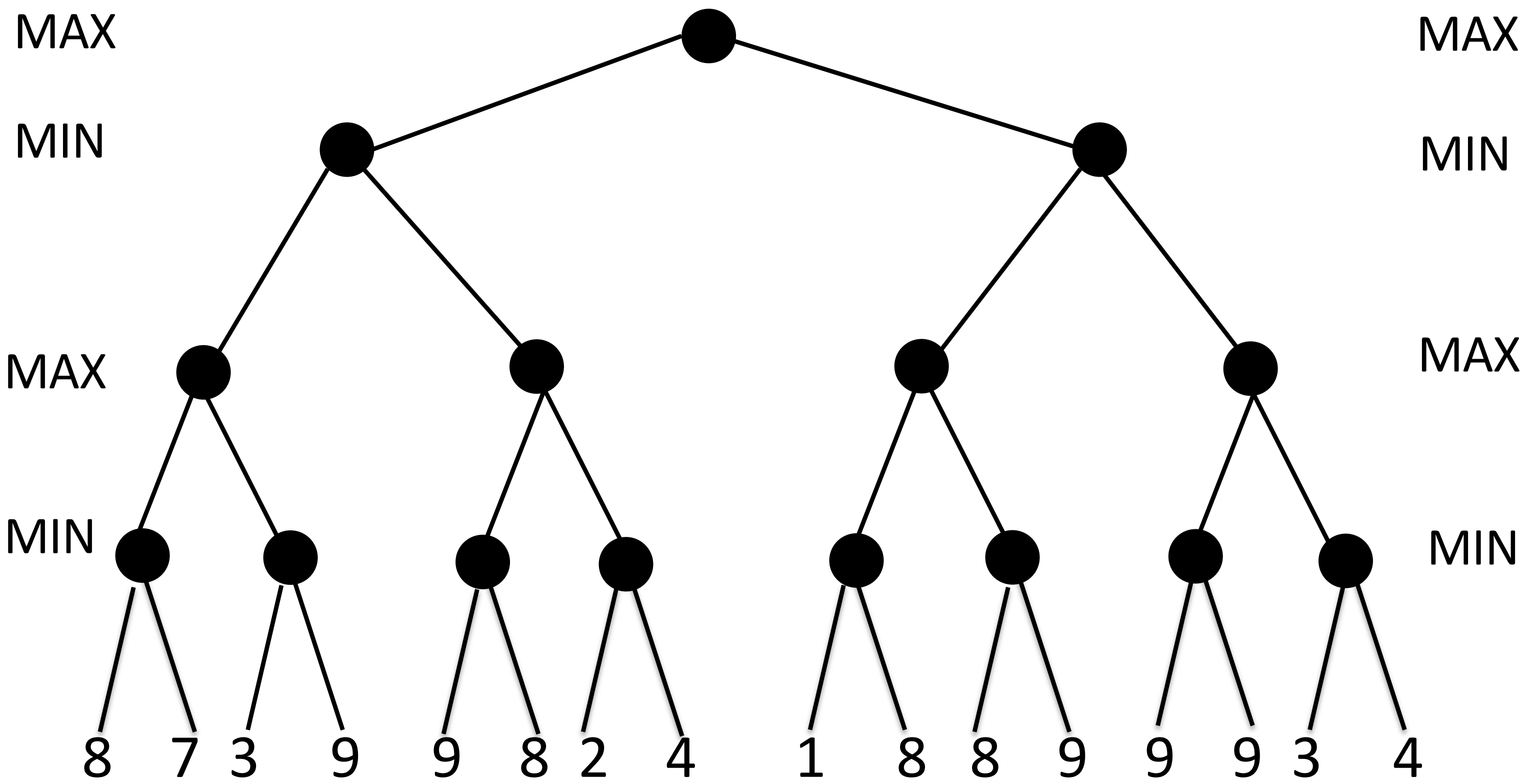
Note: [α < β], or floor < ceiling. Why?

Q: Alpha-Beta pruning value = Minimax value w/o pruning?

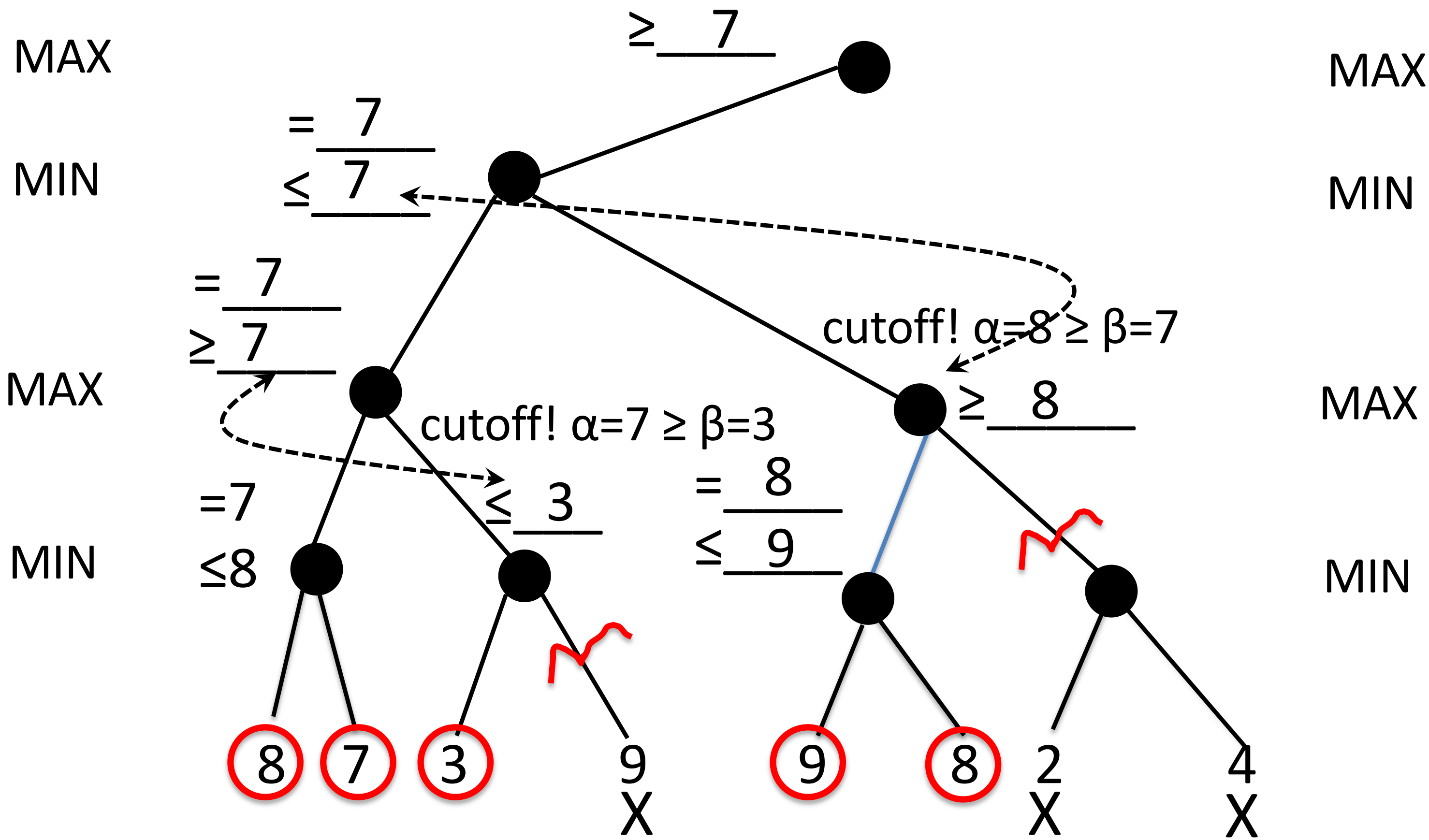
Ans: 2

prune if α≥β

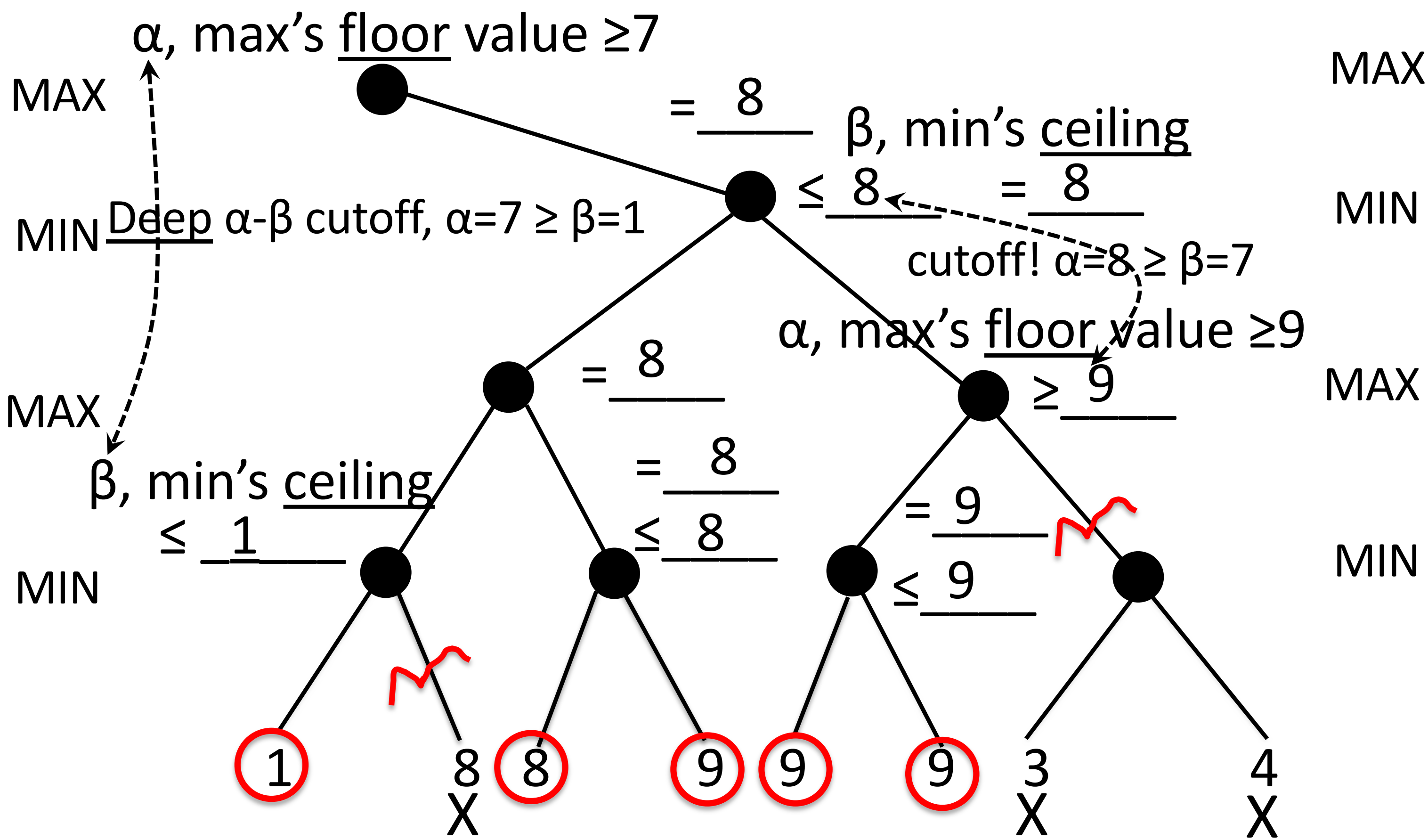
A deeper game tree illustrates how much pruning alpha-beta can do (b = 2, d= 3)



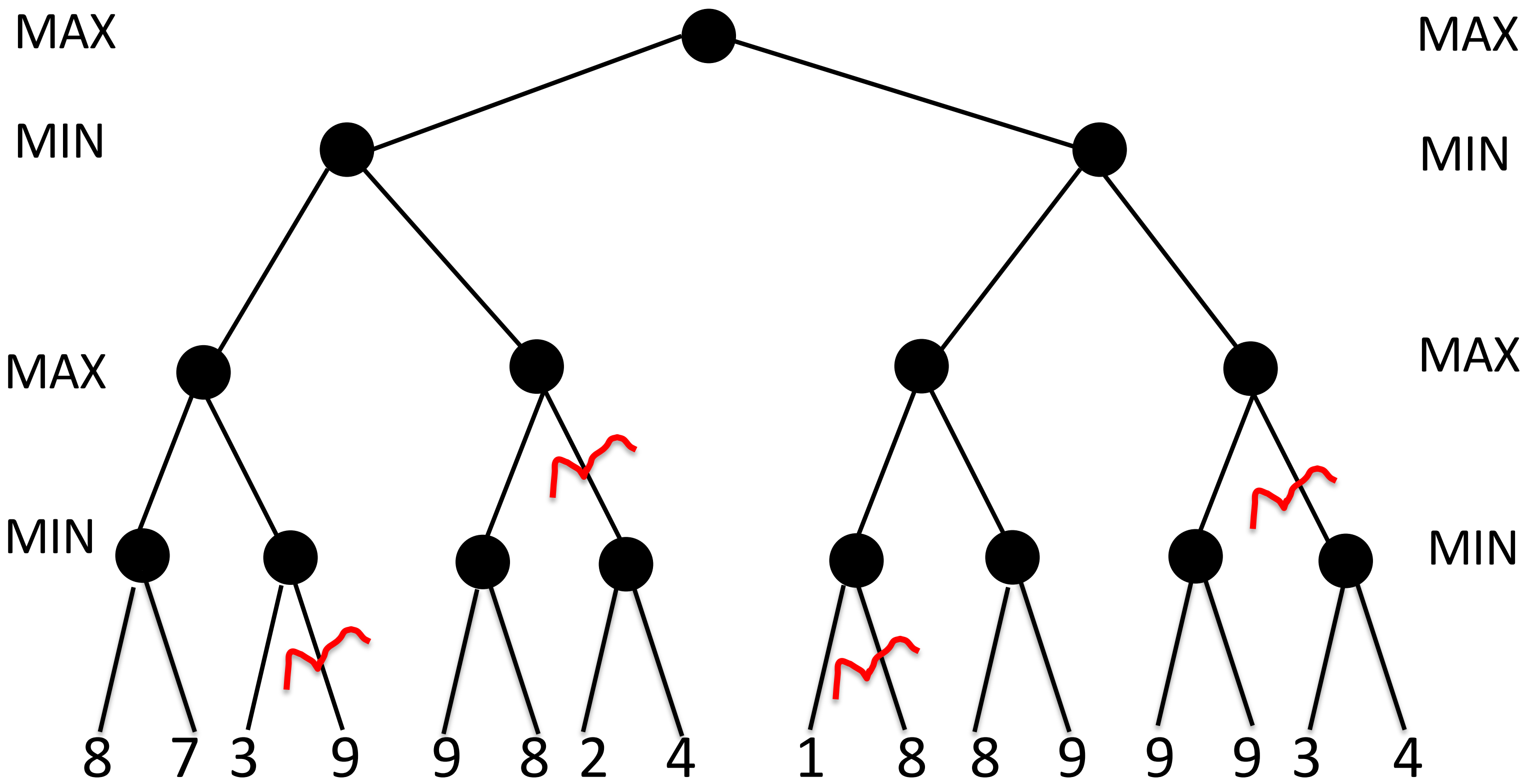
A deeper game tree illustrates how much pruning alpha-beta can do
First half of tree (max-α values; min-β values)



Second half of tree: cutoff whenever α≥ β

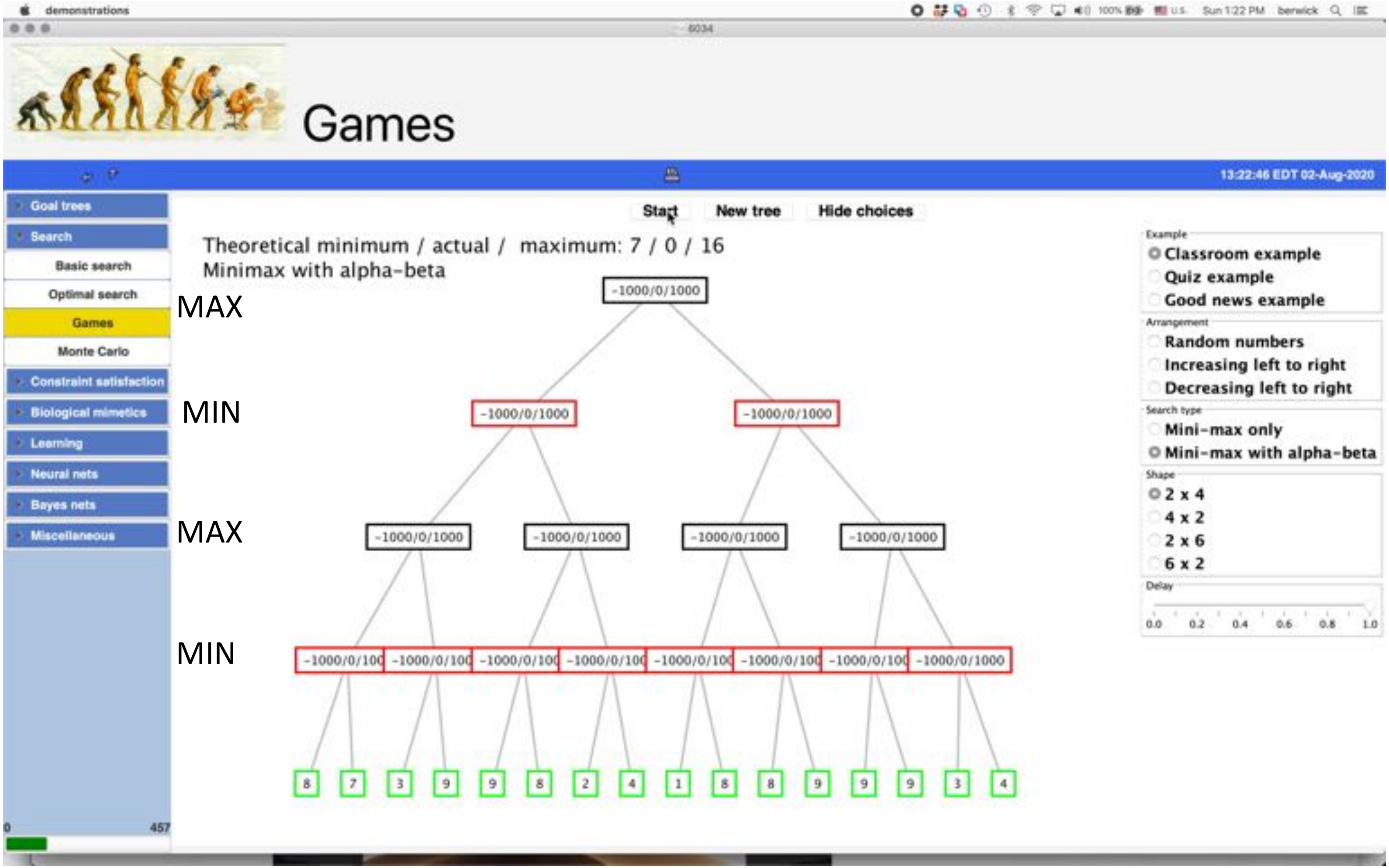


A deeper game tree



6 out of 16 static evaluations are <u>not</u> made

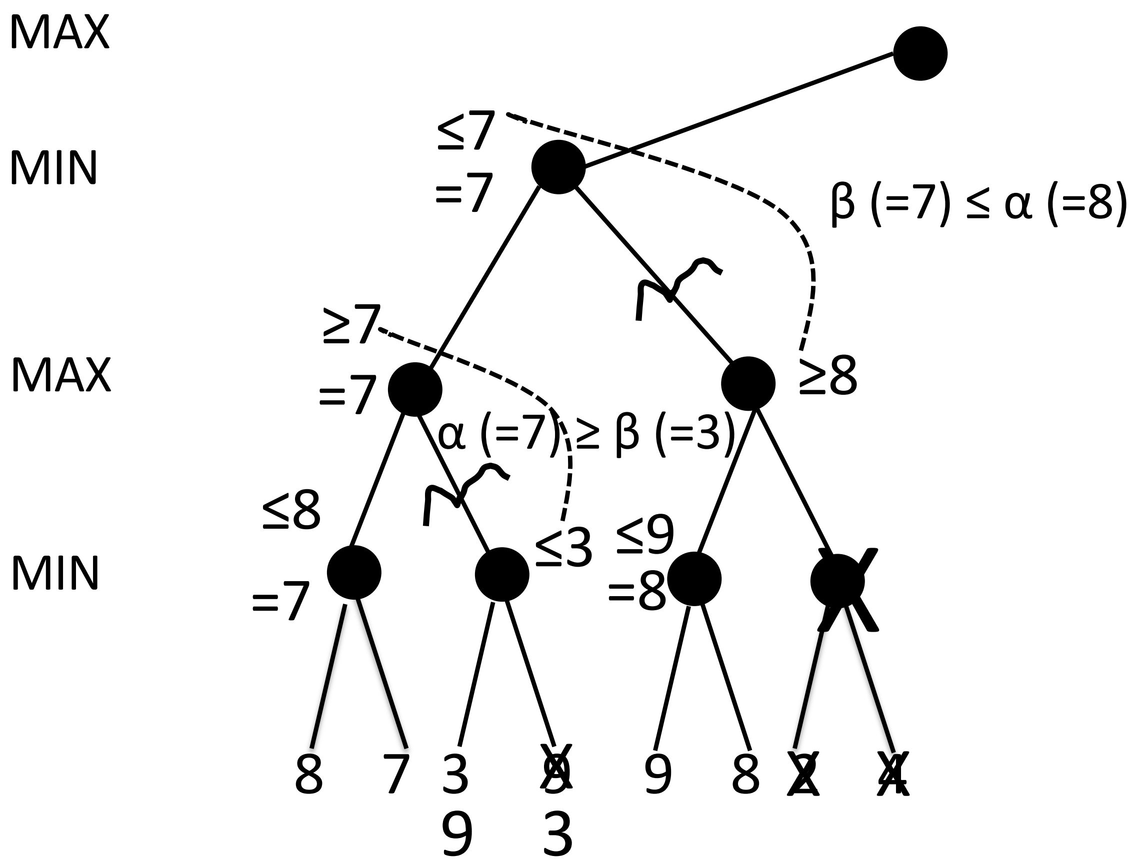
Dead horse principle (principle AWP)


Games
13:22:46 EDT 02-Aug-2020
Goal trees
Search
Basic search
Optimal search
Games
Monte Carlo
Constraint satisfaction
Biological mimetics
Learning
Neural nets
Bayes nets
Miscellaneous
Start
New tree
Hide choices
Theoretical minimum / actual / maximum: 7 / 0 / 16
Minimax with alpha-beta
MAX
MIN
MAX
MIN
-1000/0/1000
8
7
3
9
9
8
2
4
1
8
8
9
9
9
3
4
Example
Classroom example
Quiz example
Good news example
Arrangement
Random numbers
Increasing left to right
Decreasing left to right
Search type
Mini-max only
Mini-max with alpha-beta
Shape
2 x 4
4 x 2
2 x 6
6 x 2
Delay
0.0
0.2
0.4
0.6
0.8
1.0

Ordering of static values can greatly affect alpha-beta pruning

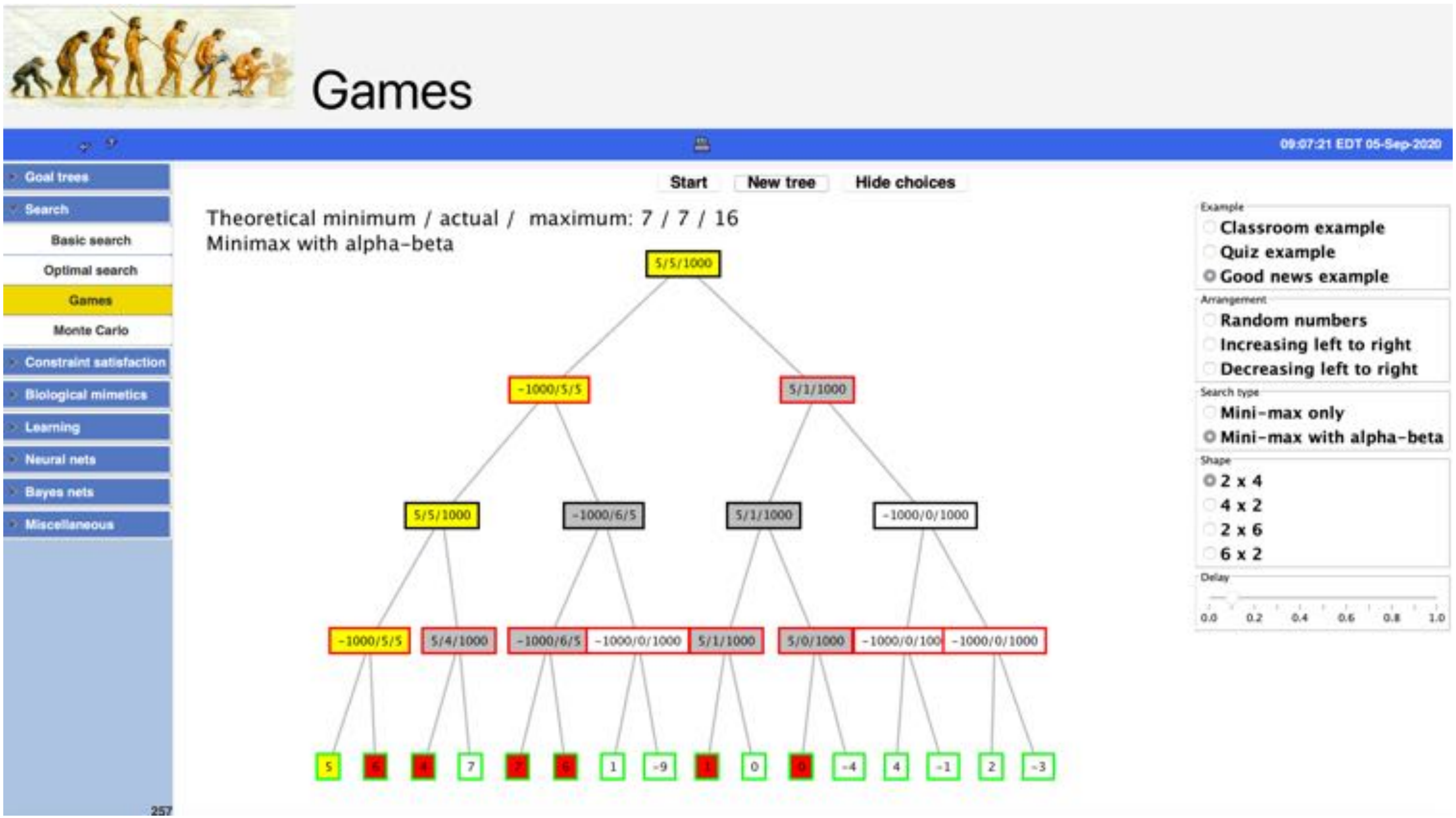
- If the most favorable successor nodes for both MAX and MIN are on the __LEFT__ so we explore them__FIRST__, then this leads to maximal pruning
- If the most favorable successor nodes for both MAX and MIN are on the __RIGHT__so they are explored __LAST__, then there is less pruning, possibly none at all
- Maximal pruning (in terms of branching factor b and tree depth d is approx: __$2\ b^{d/2}$__, so can search down 2x as far using α-β search in this "good news" optimal case, compared to std minimax

Prune whenever α ≥ β



What would happen to pruning if nodes 3, 9 reversed? 3 Not pruned!

Optimal (good news) α-β game tree pruning



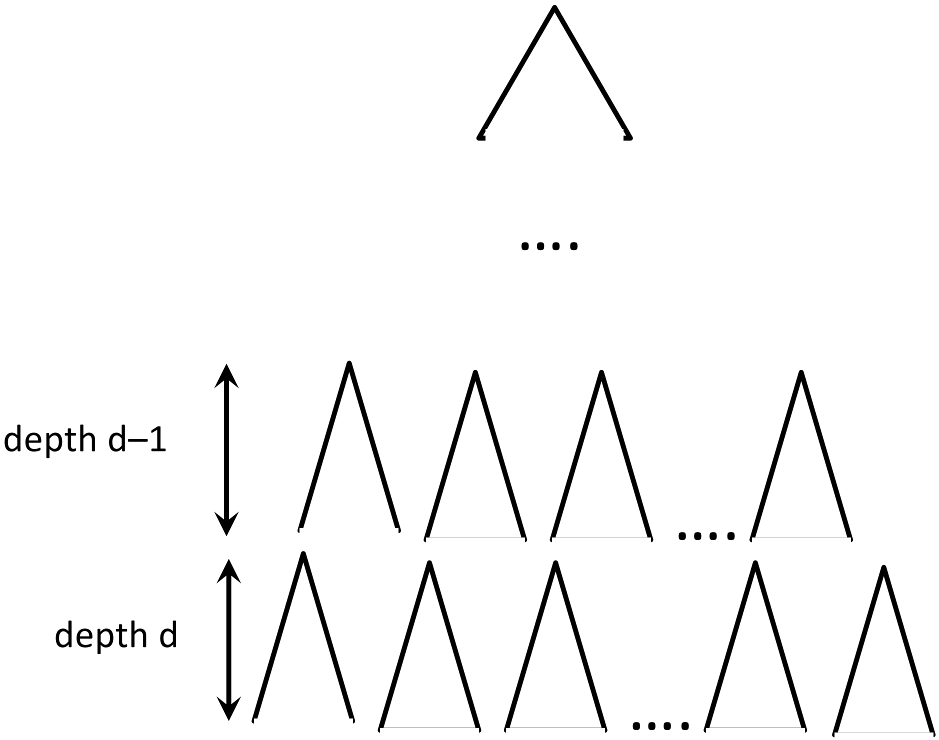
$\approx 2b^{d/2}$ $\approx$ maximum savings using α-β if optimally ordered game tree

But suppose we run out of compute time?
The branching factor depends on the game & board state – won’t know for sure how deep we can go in 1 minute….
Suppose we are playing blitz chess?
Will we always have a (good) move at hand?
Let’s combine this thought with the idea of optimally ordering the node evaluations…

Anytime algorithm

Martial arts principle

How not to run out of time



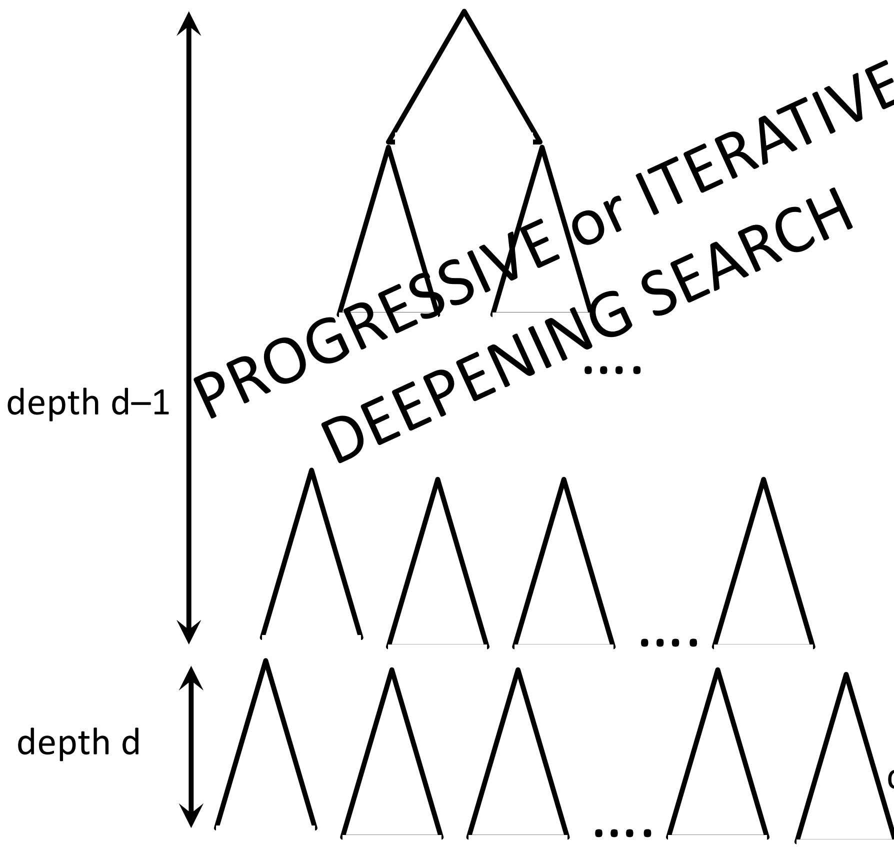
b^{d-1} nodes, or $1/b^{th}$ as much as bottom level (e.g., if branching factor is 15, only $1/15^{th}$ as much)

b^d nodes

So, we can have a move in hand at level d–1 for only $1/b^{th}$ of computation cost required to go all the way down to level d

We can do this recursively to great benefit…

How not to run out of time (ignoring α-β for now)



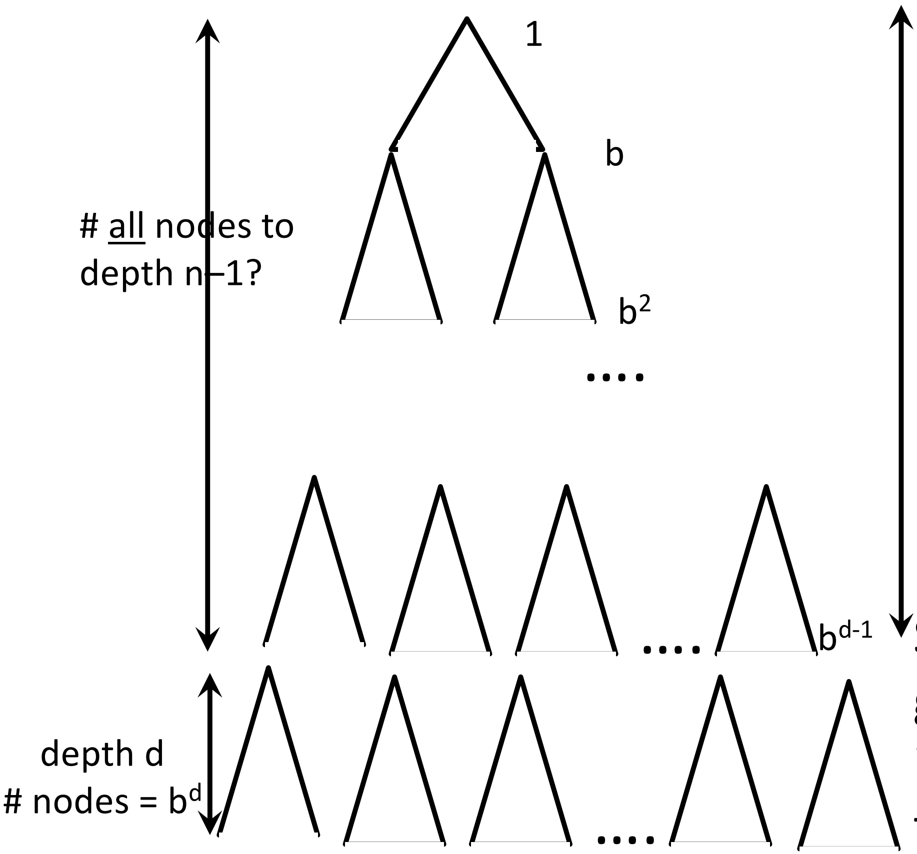
1. α-β EXPLORE tree to depth 1, or b nodes, and use the result to sort the best moves first; if there's time…
2. α-β EXPLORE tree to depth 2, or b^2 nodes, revisiting depth 1; sort the moves; if any time remains…

....

α-β EXPLORE tree to depth d–1, or b^{d-1} nodes, revisiting all depths to d–2; sort the moves; if any time remains…

α-β EXPLORE tree to depth d, or b^d nodes

Computational cost of progressive deepening?
How many total nodes S in game tree to depth d–1?



$$S=1+b+b^2+\ldots+b^{d-1}$$

$$bS= \quad b+b^2+\ldots+b^d$$

$$\Rightarrow bS-S= b^d-1$$

$$S(b-1)= b^d-1$$

$$S= \frac{b^d-\cancel{1}}{b-\cancel{1}} \approx b^{d-1}$$

So, search through <u>all</u> the game tree up thru depth d–1 is <u>only</u> time b^{d-1}, or $1/b^{th}$ of time to search to <u>full</u> depth d

Progressive (Iterative) deepening search

- Optimal DFS method (redundant use of space, but doesn't lose on arbitrary depth **d** trees)
- Earlier searches tend to improve the commonly used heuristics, so that a more accurate estimate of the score of various nodes at the final depth search can occur
- Because early iterations use small values for **d** they execute extremely quickly. This allows the algorithm to supply early indications of the result almost immediately, followed by refinements as **d** increases

Gold star ideas today

- Dead horse principle, aka “AWP”: α-β search
- Martial arts principle – use adversary’s strength against them: progressive deepening
- Anytime algorithms: progressive deepening
- Simple ≠ Trivial: sometimes, bulldozers work