AI-based Game Playing with Reinforcement Learning

DR. RAJ DASGUPTA
NAVAL RESEARCH LABORATORY, WASHINGTON D. C.
EMAIL: RAJ.DASGUPTA@NRL.NAVY.MIL
Review of Last Two Lecture Topics

- Game theory
  - Gives techniques like Nash equilibrium to determine an action for each player

- Reinforcement learning
  - Gives techniques for calculating a policy – best action that an RL agent should take at each step or step to solve a problem
Recall: Two classes of graph search algorithms - uninformed, informed

Both behave deterministically

- Evaluate the next (uninformed) or ‘best’ (informed) node in fringe
  - ‘best’ defined using evaluation or objective

Generates the worst case tree for the algorithm used, $O(b^d)$ for IDS, etc.
Probabilistic Tree Search

- Some state spaces are very large
  - Worst-case trees are also very large
- Two-player games - how many states in
  - tic-tac-toe?
  - chess?

# states in tic-tac-toe: https://math.stackexchange.com/questions/485752/tictactoe-state-space-choose-calculation
# states in chess: https://math.stackexchange.com/questions/1406919/how-many-legal-states-of-chess-exists
Probabilistic Tree Search

- Some state spaces are very large
  - Worst-case trees are also very large
- Two-player games - how many states in
  - tic-tac-toe? ~6000
  - chess? ~$10^{45}$
- Idea: Can we still find the goal node without generating the whole tree (or worst case tree) ...so that we can save time and space
- Instead of deterministic search, do stochastic search

# states in tic-tac-toe: [https://math.stackexchange.com/questions/485752/tictactoe-state-space-choose-calculation](https://math.stackexchange.com/questions/485752/tictactoe-state-space-choose-calculation)
Monte Carlo Tree Search (MCTS)

- One of the most popular stochastic tree search algorithms
  - Used by Google DeepMind to defeat human champion in AlphaGo game in 2017
  - Used combination of search and deep learning

- Several other applications
  - Games: Total War: Rome II, Poker; see https://en.wikipedia.org/wiki/Monte_Carlo_tree_search#History
  - Drug discovery, DNA sequencing
  - Autonomous vehicles
MCTS Example: Tic-Tac-Toe

- Assign utilities to players at terminal level
- Win = +1, Draw = 0, Loss = -1
- Zero sum game: gain of player X = loss of player O, vice-versa
- Show utilities of player X only, player O’s utility = - player X’s utility
MCTS Example: Tic-Tac-Toe
MCTS Example: Tic-Tac-Toe

- This is an instance of a min-max game tree
- Also called zero-sum game
- Solved using technique called \( \alpha-\beta \) pruning
- \( \alpha-\beta \) pruning too computationally intensive for very large game trees
Monte Carlo Tree Search Algorithm

Four steps:

- **Selection** - tree traversal – going down the tree (generated thus far)
- **Expansion** – add nodes to tree
- **Simulation** – randomly generate one path down tree to terminal level
  - Also called **rollout**
  - Find values (utilities) at leaf level reached by rollout
- **Backpropagation** – move values from rollout up tree
MCTS Algorithm

Selection → Expansion → Simulation → Backpropagation

Tree Policy

Default Policy

Distribution A - Approved For Public Release Distribution Unlimited
MCTS Tree Traversal + Node Expansion Algorithm

\[ UCB1(S_i) = \bar{V}_i + C \sqrt{\frac{\ln N}{n_i}}, \quad C = 2 \]

- \( n_i \): no. of times \( S_i \) visited
- \( N \): no. of times parent of \( S_i \) visited
- \( C \): constant
- \( \bar{V}_i \): average value of node \( S_i \)

start

current = \( s_0 \)

Is current a leaf?

Yes

Do rollout

is \( n_i \) value for current = 0

Yes

No

current = child node of current that maximizes \( UCB1(S_i) \)

For each available action from current, add a new node to tree

current = first new child node

Do rollout

No

Do rollout
MCTS Resources

- Short tutorial MCTS: https://www.youtube.com/watch?v=UXW2yZndl7U
- Wikipedia: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
- Youtube Udacity: https://youtu.be/onBYsen2_eA
- mcts4j library: https://github.com/avianey/mcts4j
Deep Learning based Game Playing AI

Three successively improving algorithms:
- Alpha Go, 2016
- Alpha Zero, 2017
- Alpha Star, 2019
Alpha Go: Key Ideas

- Search space (no. of states and actions) is enormous! (more than no. of atoms in universe)
- Need to reduce search space: breadth + depth
Alpha Go: Key Ideas

- Search space (no. of states and actions) is enormous! (more than no. of atoms in universe)
- Need to reduce search space: breadth + depth

**Breadth**: limit number of possible actions from a state $s$

- Using how many times an action was taken from state $s$
- Learn a function $g(s) = p(a | s)$
- $g(s)$ learned using a combination of supervised learning and RL
- Using 12 layer CNN

**Supervised part**: Trained with moves from 160K games, 30M states; 4 weeks training on 50 GPUs

**RL part**: plays with older models learned by itself during supervised learning (self-play); 1 week RL on 50 GPUs
Alpha Go: Key Ideas

- Search space (no. of states and actions) is enormous! (more than no. of atoms in universe)

- Need to reduce search space: breadth + depth

- **Breadth**: limit number of possible actions from a state $s$
  - Using how many times an action was taken from state $s$
  - Learn a function $g(s) = p(a|s)$
  - $g(s)$ learned using a combination of supervised learning and RL
  - Using 12 layer CNN
  - Supervised part: Trained with moves from 160K games, 30 M states; 4 weeks training on 50 GPUs
  - RL part: plays with older models learned by itself during supervised learning (self-play); 1 week RL on 50 GPUs
Alpha Go: Key Ideas

- Search space (no. of states and actions) is enormous! (more than no. of atoms in universe)
- Need to reduce search space: breadth + depth

**Depth**: by estimating value function $V(s)$ at state $s$

- $V(s)$ called board evaluation function
- $0 \text{ (loss)} < V(s) < 1 \text{ (win)}$
- $V(S)$ is learned using regression learning w/ stochastic gradient descent
- Using a 12 layer CNN
- 1 week on 50 GPUs
Alpha Go: Key Ideas

- Search space (no. of states and actions) is enormous! (more than no. of atoms in universe)
- Need to reduce search space: breadth + depth
Alpha Go: Key Ideas

- Search space (no. of states and actions) is enormous! (more than no. of atoms in universe)
- Need to reduce search space: breadth + depth

- Need to reduce search space: breadth + depth + lookahead
- **Lookahead:**
  - Done using MCTS on reduced search space
  - Starting from current state virtually play out the game till end
Improvement on AlphaGo

Use self-play only (learns from scratch by playing against itself) instead of using play information form human played games

Uses single neural network instead of value and policy networks

Simpler tree search than MCTS rollouts

No hyper parameter tuning (in Alpha Zero, tuning done in AlphaGo and AlphaGo Zero)
AlphaGo vs. Alpha(Go) Zero
ELO comparison

![Graph showing ELO comparison between AlphaGo and Alpha(Go) Zero along with other Go programs.](image)
Game playing AI for StarCraft-II

Multi-player game – played by multiple agents using MARL (multi-agent RL)

Partial Observability: each agent can see the game state (other players) only within its camera’s FOV

Imperfect information: only see opponent pieces within range of own units

Large action space, simultaneous moves by multiple players at same time

Counter strategies developed by human players
Main algorithm: AlphaZero + league play

League play: self-play between multiple players (two sides)

- Teammates (agents) versus opponents (competitors)

- $M$ agents, $N$ competitors

- Agents learn a policy (policy update via A2C RL)

- Competitors have fixed policy

- Best agents morph into competitors periodically

  - Ensures that agents remember to playing against some of the best opponents (past selves)
AlphaStar

- **Prioritized Fictitious Self-Play**
  - **Matchmaking** between agents and competitors done to match expert level of both sides
  - **Exploiters** (type of competitors) identify weaknesses in agents so that agents can discover and overcome those weaknesses (via self-play)
  - Uses supervised learning from human games to speed-up learning (like AlphaGo)

https://youtu.be/xP7LwZxq0ss
AlphaStar Performance

[Graph showing MMR (Mastery Rating Index) vs. player skill level categories: Silver league, Gold league, Platinum league, Diamond league, Master league, FSP, PFSP, + Main exploiters, AlphaStar supervised, Grandmaster league. AlphaStar is above and to the right of all other categories.]
Resources

Papers:


Slides:

- Alpha Go: https://www.slideshare.net/ShaneSeungwhanMoon/how-alphago-works

- Alpha (Go) Zero: https://www.slideshare.net/KarelHa1/alphazero
Sequential Games
Example Sequential Game

- Pure strategies of player 1:
  - (A, G) (A, H) (B, G) (B, H)
- Pure strategies for player 2:
  - (C, E) (C, F) (D, E) (D, F)
- Note: when player 1 chooses A, its own choice of G vs. H has no consequence
- With above pure strategies, we can write induced normal form of the game

\[
\begin{array}{c|c|c|c|c}
 & (A,G) & (A,H) & (B,G) & (B,H) \\
\hline
(A,G) & 3, 8 & 3, 8 & 8, 3 & 8, 3 \\
(A,H) & 3, 8 & 3, 8 & 8, 3 & 8, 3 \\
(B,G) & 5, 5 & 2, 10 & 5, 5 & 2, 10 \\
(B,H) & 5, 5 & 1, 0 & 5, 5 & 1, 0 \\
\end{array}
\]
Example Sequential Game

- Temporal structure of extensive form cannot be captured in induced normal form
- Redundancy in normal form
  - 16 outcomes (with duplicates) in normal form vs. 5 outcomes in extensive form
- Converting from extensive to induced normal form can lead to exponential blowup in number of outcomes

<table>
<thead>
<tr>
<th></th>
<th>(A,G)</th>
<th>(A,H)</th>
<th>(B,G)</th>
<th>(B,H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C,E)</td>
<td>3, 8</td>
<td>3, 8</td>
<td>8, 3</td>
<td>8, 3</td>
</tr>
<tr>
<td>(C,F)</td>
<td>3, 8</td>
<td>3, 8</td>
<td>8, 3</td>
<td>8, 3</td>
</tr>
<tr>
<td>(D,E)</td>
<td>5, 5</td>
<td>2, 10</td>
<td>5, 5</td>
<td>2, 10</td>
</tr>
<tr>
<td>(D,F)</td>
<td>5, 5</td>
<td>1, 0</td>
<td>5, 5</td>
<td>1, 0</td>
</tr>
</tbody>
</table>
Backward Induction

- Offers an easy, systematic way to check if a NE is subgame perfect

- Idea:
  - Given a NE strategy start at bottommost subtree which has terminal nodes (with joint utilities)
  - Go upwards towards root - at each decision node (branch) check if the NE strategy is the best response for the player moving at that node
    - Yes: move (copy) the utility to the player one level up the tree
    - No: stop; NE is not subgame perfect

- Check (A, G) (C, F) and (B, H) ()

Backward induction can be implemented as depth first traversal with recursion
Connection between Sequential Games and Game-playing AI

- AlphaStar selects actions using Nash equilibrium in sequential games
- Nash equilibrium analysis of sequential games is useful in imperfect information games:
  - StarCraft-II
  - Texas Hold 'em Poker (DeepStack)
  - Hanabi
- Uses a technique called counter factual regret (CFR) minimization instead of Nash equilibrium
Summary and Conclusions

- Game theory gives formal, mathematical methods for taking actions by players in a game with guaranteed solutions (like Nash equilibrium outcome)
  - But for small settings – mainly 2 players, with few actions
  - Developed in mathematics and economics, does not use AI techniques inherently for solutions
- Deep RL and AI-based search techniques give methods to solve larger games with many actions
  - But mainly heuristics-based solutions
  - Learns via trial and error against humans or self-play
- Combination of game theory and RL can be used to solve large games with imperfect information with guaranteed solutions
Questions

- Dr. Raj Dasgupta
- Naval Research Laboratory, Washington D. C.
- Email: raj.Dasgupta@nrl.navy.mil
8 weeks of research internships high school students to participate at Department of Navy Laboratories including NRL

Major criteria:
- Completed Grade 9
- Graduating seniors can apply
- Must be 16 years or older at time of application
- U.S. Citizenship (for NRL)

NRL research areas: AI/ML, computer science, engineering, space sciences, radar, remote sensing, plasma physics, chemistry, bio-sciences, material sciences, acoustics

Application deadline: November 30, 2020

Website: https://seap.asee.org/
Backup Slides

MORE INFORMATION
Sequential Game Overview

- Recall: Normal form game – players move (select actions) simultaneously
- Sequential games - remove the requirement that players move (select actions) simultaneously
- Players take turns to make moves (e.g., chess, tic-tac-toe, checkers)
- Also called extensive form game
- Normal form game – matrix representation
- Extensive form game – tree representation
- Each extensive form game can be converted to a corresponding normal form game – called game’s induced normal form
Sequential Game Overview

- All results related to Nash equilibrium and its calculations in normal form games also hold for extensive form games
- Extensive form game allows calculation of subgame perfect equilibrium
  - Allows to speed up Nash equilibrium calculation
Perfect information: Assume that every player can see the actions and payoffs (at joint outcomes) of every other player

Assume games are finite (finish in finite number of moves)
Definition 5.1.1 (Perfect-information game) A (finite) perfect-information game (in extensive form) is a tuple $G = (N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- $N$ is a set of $n$ players;
- $A$ is a (single) set of actions;
- $H$ is a set of nonterminal choice nodes;
- $Z$ is a set of terminal nodes, disjoint from $H$;
- $\chi : H \mapsto 2^A$ is the action function, which assigns to each choice node a set of possible actions;
- $\rho : H \mapsto N$ is the player function, which assigns to each nonterminal node a player $i \in N$ who chooses an action at that node;
- $\sigma : H \times A \mapsto H \cup Z$ is the successor function, which maps a choice node and an action to a new choice node or terminal node such that for all $h_1, h_2 \in H$ and $a_1, a_2 \in A$, if $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ then $h_1 = h_2$ and $a_1 = a_2$; and
- $u = (u_1, \ldots, u_n)$, where $u_i : Z \mapsto \mathbb{R}$ is a real-valued utility function for player $i$ on the terminal nodes $Z$. 
Sequential Game Example: Sharing Game

- Brother and sister sharing two presents
- Presents cannot be divided
- Both siblings value presents equally, and additively
- Brother moves first, will pick one of three possible splits
  - Brother keeps both
  - Sister keeps both
  - Brother keeps one, sister keeps one
Sequential Game Example: Sharing Game

- Brother and sister sharing two presents
- Presents cannot be divided
- Both siblings value presents equally, and additively
- Brother moves first, will pick one of three possible splits (moves)
  - Brother keeps both
  - Sister keeps both
  - Brother keeps one, sister keeps one
- Start node labeled with player id 1
- Each move is labeled with the corresponding split
Sequential Game Example: Sharing Game

- Sister moves next, picks one of two moves
  - Accept (yes) brother’s split (move)
  - Reject (no) brother’s split (move)
Sequential Game Example: Sharing Game

- Sister moves next, picks one of two moves
  - Accept (yes) brother’s split (move)
  - Reject (no) brother’s split (move)
- Game ends after sister’s move
- Leaf nodes are labeled with outcome of game – joint utility of players from each of the outcomes
Sequential Game Example: Sharing Game

- Pure strategies of player 1: {2-0, 1-1, 0-2}
- Pure strategies of player 2 – all possible combinations of moves, for each of player 1’s moves
  - {(2-0-no), (2-0, yes), (1-1-no), (1-1, yes), (0-2, no), (0-2, yes)}
**Theorem 5.1.3** Every (finite) perfect-information game in extensive form has a pure-strategy Nash equilibrium.

Why do perfect information extensive form games always have a pure strategy NE?
Extensive Form Game with Perfect Information: Pure Strategy NE

Theorem 5.1.3  Every (finite) perfect-information game in extensive form has a pure-strategy Nash equilibrium.

- Why do perfect information extensive form games always have a pure strategy NE?
- Why did we have mixed strategies in normal form game?
- Because each player did not know which action other player would select, and tried to guess by randomizing over other players possible actions
- In extensive form each player can observe other player’s moves as players move sequentially… no need to guess or randomize
Sub-game Perfect Equilibria
Example Sequential Game: Induced Normal Form

<table>
<thead>
<tr>
<th></th>
<th>(C,E)</th>
<th>(C,F)</th>
<th>(D,E)</th>
<th>(D,F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,G)</td>
<td>3, 8</td>
<td>3, 8</td>
<td>8, 3</td>
<td>8, 3</td>
</tr>
<tr>
<td>(A,H)</td>
<td>3, 8</td>
<td>3, 8</td>
<td>8, 3</td>
<td>8, 3</td>
</tr>
<tr>
<td>(B,G)</td>
<td>5, 5</td>
<td>2, 10</td>
<td>5, 5</td>
<td>2, 10</td>
</tr>
<tr>
<td>(B,H)</td>
<td>5, 5</td>
<td>1, 0</td>
<td>5, 5</td>
<td>1, 0</td>
</tr>
</tbody>
</table>

What are the pure strategy NE in the above game?
Example Sequential Game: Induced Normal Form

<table>
<thead>
<tr>
<th></th>
<th>(C,E)</th>
<th>(C,F)</th>
<th>(D,E)</th>
<th>(D,F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,G)</td>
<td>3,8</td>
<td>3,8</td>
<td>8,3</td>
<td>8,3</td>
</tr>
<tr>
<td>(A,H)</td>
<td>3,8</td>
<td>3,8</td>
<td>8,3</td>
<td>8,3</td>
</tr>
<tr>
<td>(B,G)</td>
<td>5,5</td>
<td>2,10</td>
<td>5,5</td>
<td>2,10</td>
</tr>
<tr>
<td>(B,H)</td>
<td>5,5</td>
<td>1,0</td>
<td>5,5</td>
<td>1,0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(C, E)</th>
<th>(C, F)</th>
<th>(D, E)</th>
<th>(D, F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A, G)</td>
<td>3,8</td>
<td>3,8</td>
<td>8,3</td>
<td>8,3</td>
</tr>
<tr>
<td>(A, H)</td>
<td>3,8</td>
<td>3,8</td>
<td>8,3</td>
<td>8,3</td>
</tr>
<tr>
<td>(B, G)</td>
<td>5,5</td>
<td>2,10</td>
<td>5,5</td>
<td>2,10</td>
</tr>
<tr>
<td>(B, H)</td>
<td>5,5</td>
<td>1,0</td>
<td>5,5</td>
<td>1,0</td>
</tr>
</tbody>
</table>

What are the pure strategy NE in the above game?
Subgame Perfect Equilibria

- Are all pure strategy NE possible in extensive form
- Consider (A, G) (C, F)
- When P1 plays H instead of G can he improve utility?
- When P2 plays E instead of F can he improve utility?
- When P1 plays B instead of A can he improve utility?
  - If P1 plays B, due to P2's move F (remember we assume other player does not deviate) P1 will end up getting either utility of 2 or 1
  - If P1 plays A, due to P2's move C (P2 does not deviate), P1 will get 3
  - So, best for P1 not to deviate to B from A
- When P2 plays D instead of C can he improve utility?
Subgame Perfect Equilibria

- Are all pure strategy NE possible in extensive form?
- Consider (A, G) (C, F)
- When P1 plays H instead of G can he improve utility?
- When P2 plays E instead of F can he improve utility?
- When P1 plays B instead of A can he improve utility?
  - If P1 plays B, due to P2’s move F (remember we assume other player does not deviate), P1 will end up getting either utility of 2 or 1
  - If P1 plays A, due to P2’s move C (P2 does not deviate), P1 will get 3
  - So, best for P1 not to deviate to B from A

No incentive to deviate: (A, G) (C, F) is NE

When P2 plays D instead of C can he improve utility?
Subgame Perfect Equilibria

- Are all pure strategy NE possible in extensive form?
- Consider (B, H) (C, E)
- When P1 plays G instead of H can he improve utility?
- When P2 plays F instead of E can he improve utility?
  - P2 could get utility 10 if P1 played G.
  - But P1 gives “threat” to P2: if you play F, I will punish you by playing H (and giving you 0 utility).
- But, is this threat credible?
Subgame Perfect Equilibria

- Are all pure strategy NE possible in extensive form?
- Consider (B, H) (C, E)
- When P1 plays G instead of H can he improve utility?
- When P2 plays F instead of E can he improve utility?
  - P2 could get utility 10 if P1 played G
  - But P1 gives “threat” to P2: if you play F, I will punish you by playing H (and giving you 0 utility)
  - But, is this threat credible?
  - P2 sees that if P1 plays H, P1 will get utility of 1 vs. getting utility of 2 by playing G
  - So, P2 reasons that the threat is not credible
Subgame Perfect Equilibria

- Are all pure strategy NE are possible in extensive form
- Consider (B, H) (C, E)
- When P1 plays G instead of H can he improve utility?
- When P2 plays F instead of E can he improve utility?
  - P2 could get utility 10 if P1 played G
  - But P1 gives “threat” to P2: if you play F, I will punish you by playing H (and giving you 0 utility)
  - But, is this threat credible?
  - P2 sees that if P1 plays H, P1 will get utility of 1 vs. getting utility of 2 by playing G
  - So, P2 reasons that the threat is not credible...which means P1 will play G, and P2 will get utility of 10...incentive to deviate for P2 from E (util. 5) to F (util. 10)
Backward Induction

ALGORITHM DETAILS
Subgame Perfect Equilibria

- Are all pure strategy NE are possible in extensive form?
- Consider (B, H) (C, E)
- When P1 plays G instead of H can he improve utility?
- When P2 plays F instead of E can he improve utility?
  - P2 could get utility 10 if P1 played G
  - But P1 gives “threat” to P2: if you play F, I will punish you by playing H (and giving you 0 utility)
  - But, is this threat credible?
  - P2 sees that if P1 plays H, P1 will get utility of 1 vs. getting utility of 2 by playing G
  - So, P2 reasons that the threat is not credible...which means P1 will play G, and P2 will get utility of 10...incentive to deviate for P2 from E (util. 5) to F (util. 10)

In such scenarios, we say the NE (of induced normal form) is not subgame perfect
Backward Induction

- Start with equilibria at bottommost level of subtrees (in subgames)
- Assume that those equilibria will be played as we move upwards to larger sub-trees
- Can be implemented as depth first traversal of game tree

Recall: Converting from extensive to induced normal form can lead to exponential blowup in number of outcomes

```
function BACKWARDINDUCTION (node h) returns u(h)
if h ∈ Z then
  return u(h)  // h is a terminal node
best_util ← −∞
forall a ∈ χ(h) do
  util_at_child ← BACKWARDINDUCTION(σ(h, a))
  if util_at_childρ(h) > best_utilρ(h) then
    best_util ← util_at_child
return best_util
```
Backward Induction

function BACKWARDINDUCTION (node h) returns u(h)
if h ∈ Z then
    return u(h)

best_util ← −∞
for all a ∈ χ(h) do
    util_at_child ← BACKWARDINDUCTION(σ(h, a))
    if util_at_child_ρ(h) > best_util_ρ(h) then
        best_util ← util_at_child

return best_util

util_at_child_ρ(h) is the utility received by player ρ(h) - element of joint utility vector corresponding to player ρ(h)’s utility

Symbols (from Def 5.1.1)
- Z: terminal nodes
- χ(h): action set at node h
- σ(h, a): next node reached by playing action a at node h
- ρ(h): player at node h

If (utility returned from subtree for player ρ(h) > best utility for player ρ(h) player till now) then
update best utility for ρ(h)
Backward Induction

Symbols (from Def 5.1.1)
- $Z$: terminal nodes
- $\chi(h)$: action nodes at node $h$
- $\sigma(h,a)$: next node reached by playing action $a$ at node $h$
- $\rho(h)$: player at node $h$

Returns $u(h)$ – utility ‘label’ for each node $h$ (not the equilibrium strategy)

Player $\rho(h)$ can then use this label to find its best (equilibrium) action at node $h$ given by $\text{arg} \max_{a \in \chi(h)} u_i(\sigma(h, a))$

```
function BACKWARDINDUCTION (node h) returns u(h)
if $h \in Z$ then
    return $u(h)$  // $h$ is a terminal node
best_util $\leftarrow -\infty$
forall $a \in \chi(h)$ do
    util_at_child $\leftarrow$ BACKWARDINDUCTION($\sigma(h, a)$)
    if util_at_child $\rho(h)$ $>$ best_util $\rho(h)$ then
        best_util $\leftarrow$ util_at_child
return best_util
```
Backward Induction

returns $u(h)$

Player $r(h)$ can then use this label to find its best (equilibrium) action at node $h$ given by

$$\text{arg max}_{a_i \in \chi(h)} u_i(\sigma(h, a_i))$$

Returns $u(h)$ – utility ‘label’ for each node $h$ (not the equilibrium strategy)

Player $p(h)$ can then use this label to find its best (equilibrium) action at node $h$ given by $\text{arg max}_{a_i \in \chi(h)} u_i(\sigma(h, a_i))$
Backward Induction

Returns $u(h)$ – utility ‘label’ for each node $h$ (not the equilibrium strategy)

Player $p(h)$ can then use this label to find its best (equilibrium) action at node $h$ given by $\arg\max_{a_i \in \chi(h)} u_i(\sigma(h, a_i))$

Select action that will maximize utility in subgame (recursively)

Symbols (from Def 5.1.1)
- $Z$: terminal nodes
- $\chi(h)$: action set at node $h$
- $\sigma(h, a)$: next node reached by playing action $a$ at node $h$
- $p(h)$: player at node $h$
Backward Induction

- Backward induction is an instance of depth first traversal
- Recall, depth first traversal has time complexity $O(b^m)$
  - $b$: branch factor
  - $m$: maximum depth
- For chess, $b=10$, $m =150$...backward induction still takes order of $10^{150}$ steps...still a problem