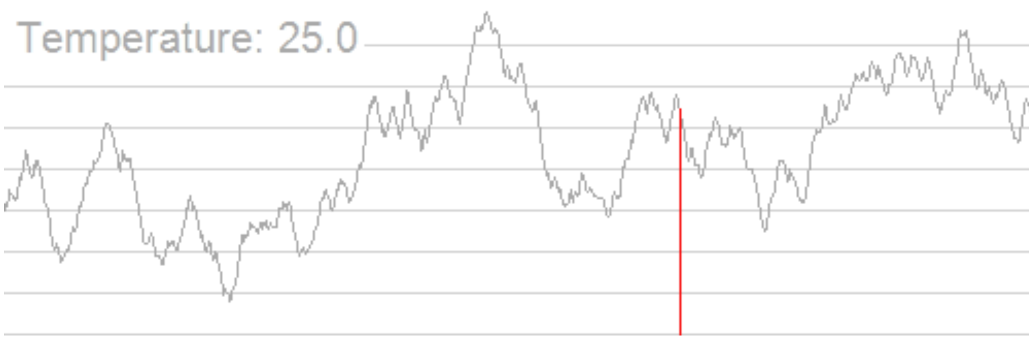
CS440/ECE448 Lecture 31: Stochastic Search & Stochastic Games

Mark Hasegawa-Johnson, 4/2022



Hill_Climbing_with_Simulated_Annealing.gif, Public domain image, Kingpin13, 2013

A contemporary backgammon set. Public domain photo by Manuel Hegner, 2013, https://commons.wikimedia.org/w/index.php?curid=25006945

Outline

- Stochastic games: the game itself is random
- Stochastic search: the game is deterministic, but we use randomness in the search, to find a fast approximate solution

Stochastic games

How can we incorporate dice throwing into the game tree?

Minimax

State evolves deterministically (when a player acts, that action uniquely determines the following state).

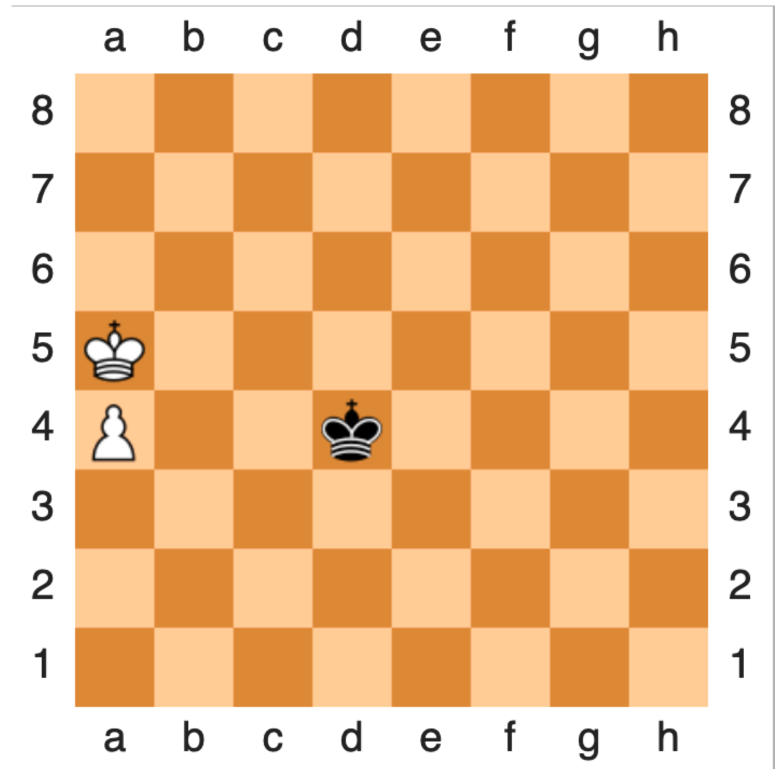
Current state is visible to both players.

Each player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the resulting utility:

$$U(s) = \max_{s' \in C(s)} U(s')$$

$$U(s') = \min_{s'' \in C(s')} U(s'')$$



Expectiminimax

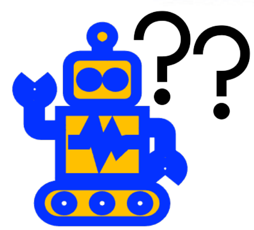
State evolves **stochastically** (when a player acts, the game changes RANDOMLY, with a probability distribution $P(s'|s,a)$ that depends on the action, a).

Current state, s, is visible to the player.

The player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Expected value** (over all possible successor states) of the resulting utility:

$$Q(s,a) = \sum_{s'} P(s'|s,a) U(s')$$



Expectiminimax

State evolves **stochastically** (when a player acts, that action influences the state transition probability).

Current state is visible to both players.

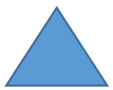
Each player tries to maximize his or her own reward:

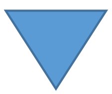
- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the
- **Expected value** (over all possible successor states) of the resulting utility:

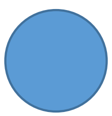
$$U(s) = \max_{a} \sum_{s'} P(s'|s,a)U(s')$$

$$U(s') = \min_{a'} \sum_{s''} P(s''|s',a')U(s'')$$

Expectiminimax: notation

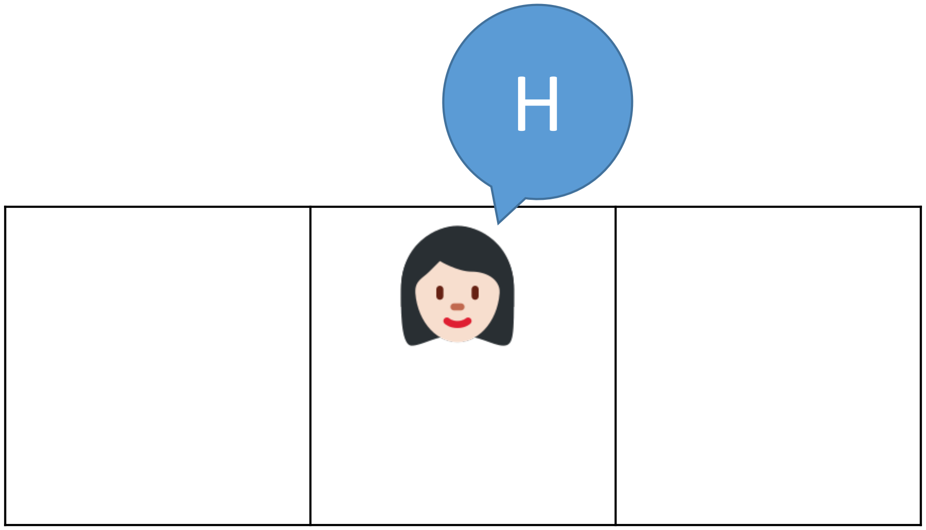
 = MAX node. $U(s) = \max_{a \in A(s)} Q(s, a)$

 = MIN node. $U(s) = \min_{a \in A(s)} Q(s, a)$

 = Chance node. $Q(s, a) = \sum_{s'} P(s'|s, a) U(s')$

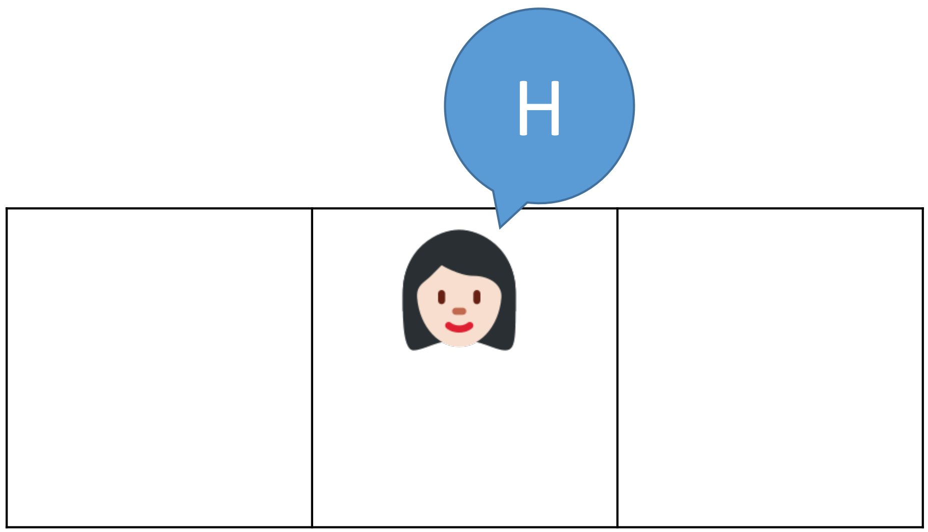
Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.



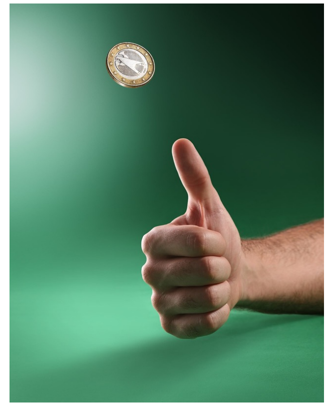
Expectiminimax example

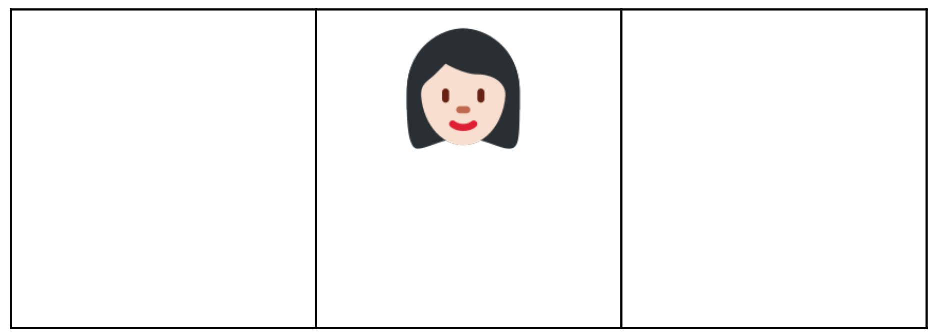
- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.



Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.

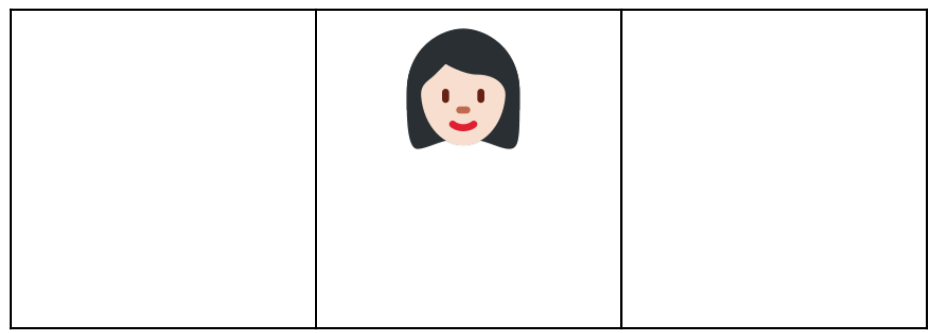




Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.

By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

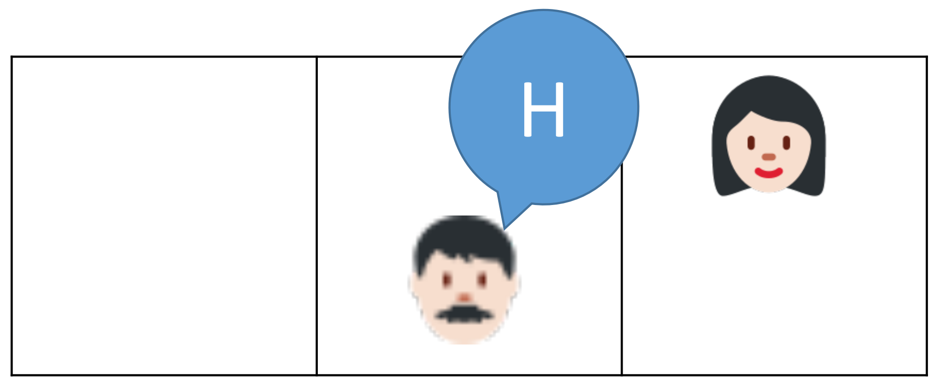


Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366

Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.

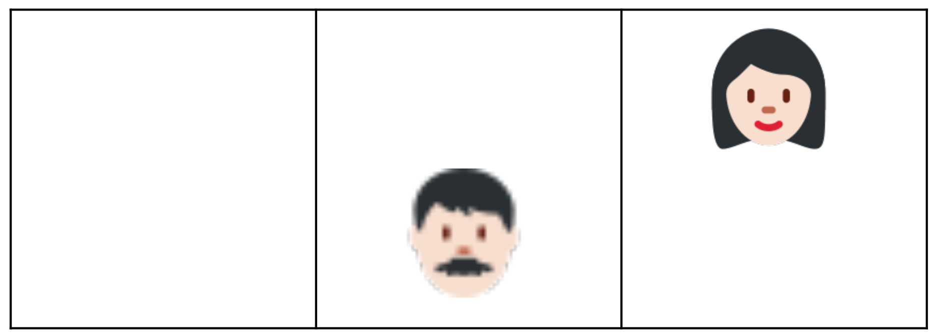
By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918



Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366

Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.



Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

Reward: $2 to the winner, $0 for a draw.

By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

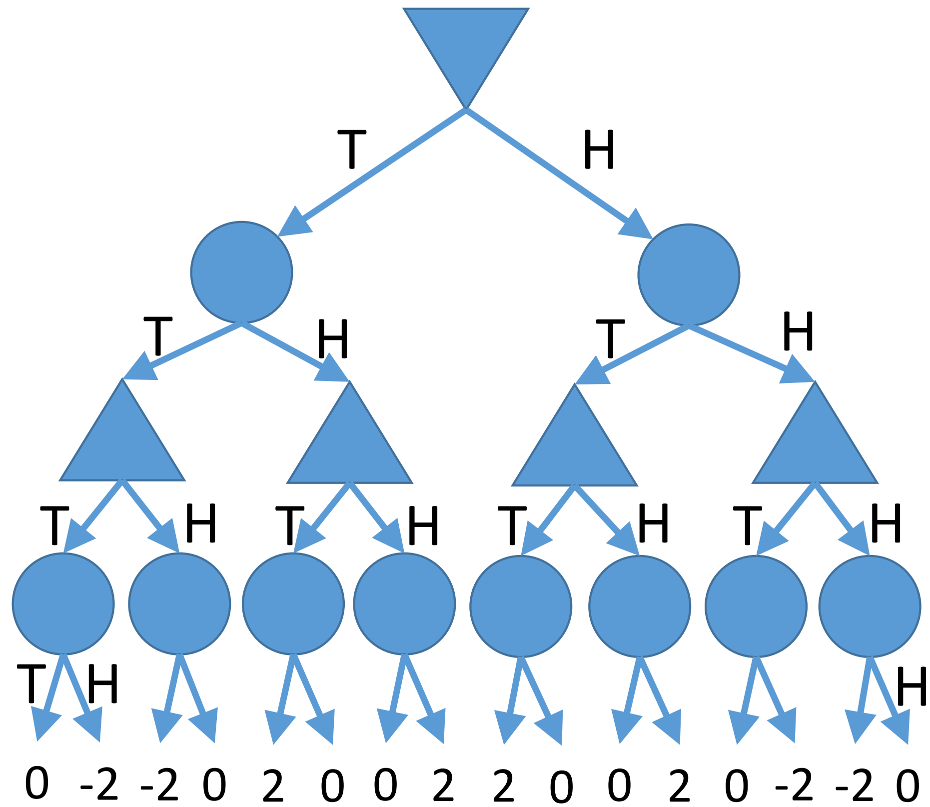
By NJR ZA - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4228918

Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366. $2 By Bureau of Engraving and Printing: U.S. Department of the Treasury - own scanned, Public Domain, https://commons.wikimedia.org/w/index.php?curid=56299470

Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

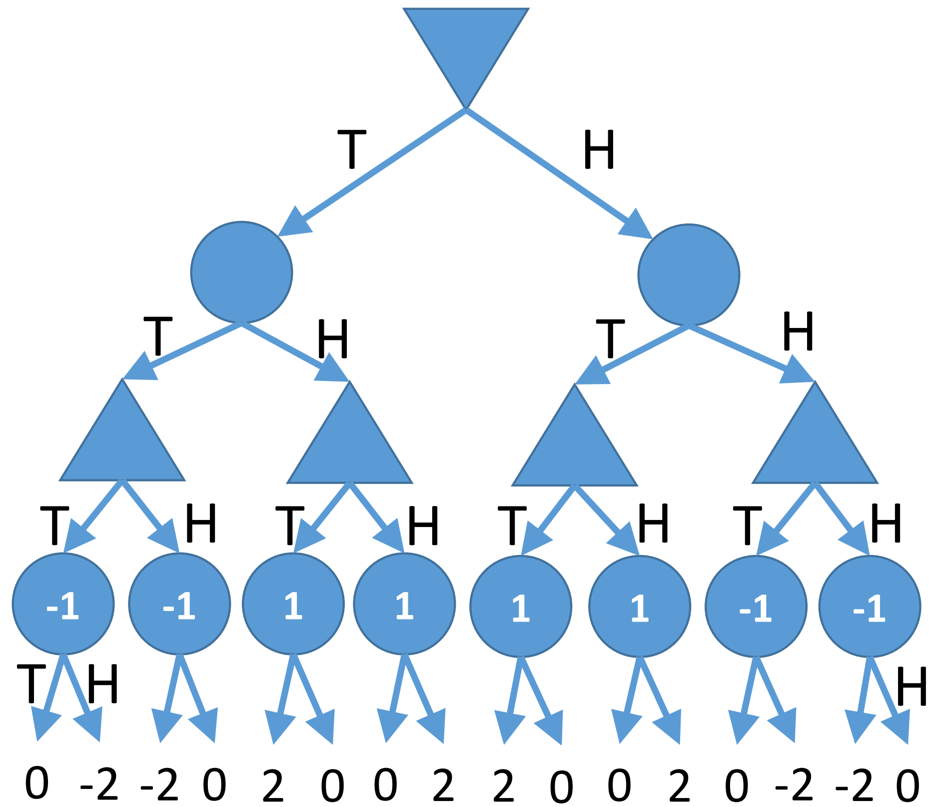
Reward: $2 to the winner, $0 for a draw.



Expectiminimax example

Chance node:

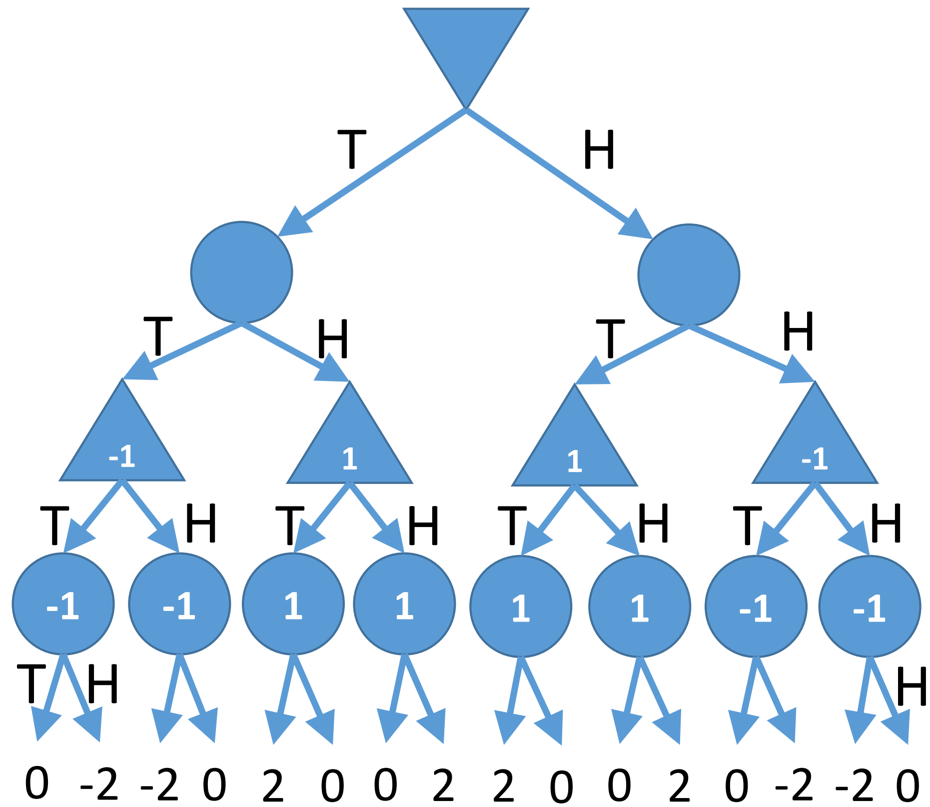
$$Q(s,a) = \sum_{s'} P(s'|s,a)U(s')$$



Expectiminimax example

Max node:

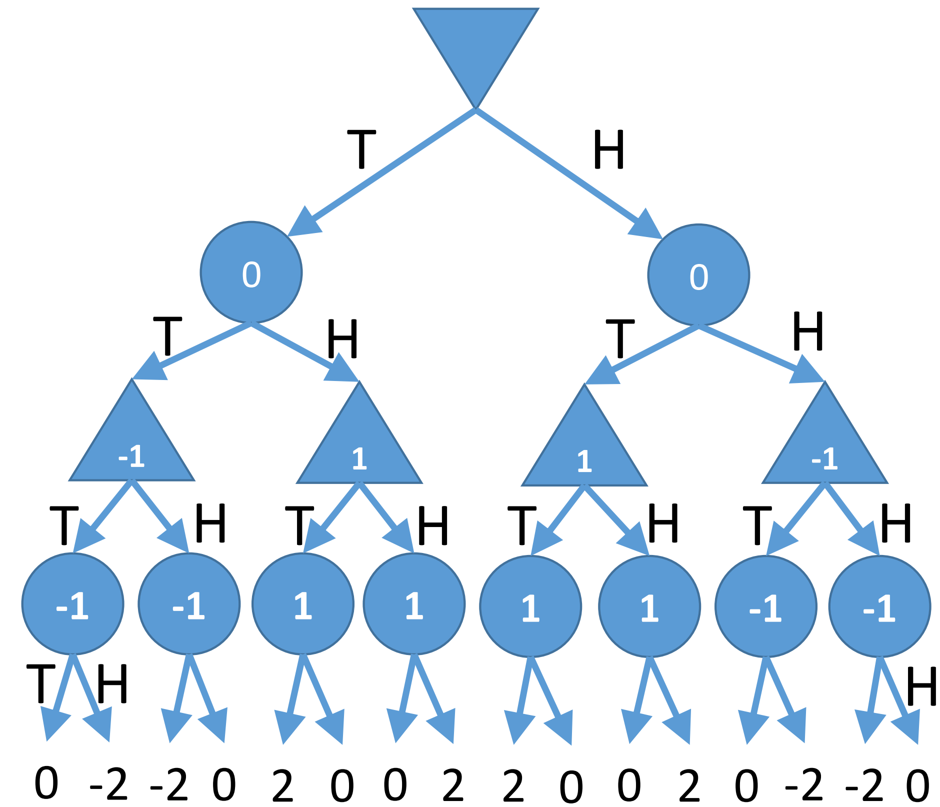
$$U(s) = \max_{a \in A(s)} Q(s, a)$$



Expectiminimax example

Chance node:

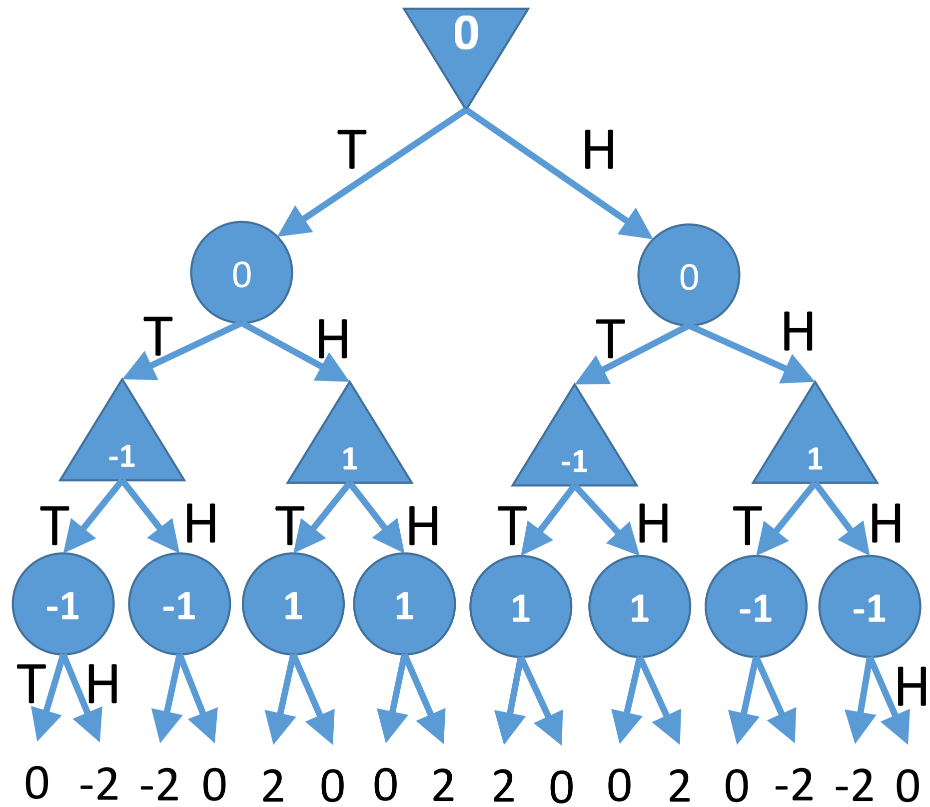
$$Q(s,a) = \sum_{s'} P(s'|s,a) U(s')$$



Expectiminimax example

Min node:

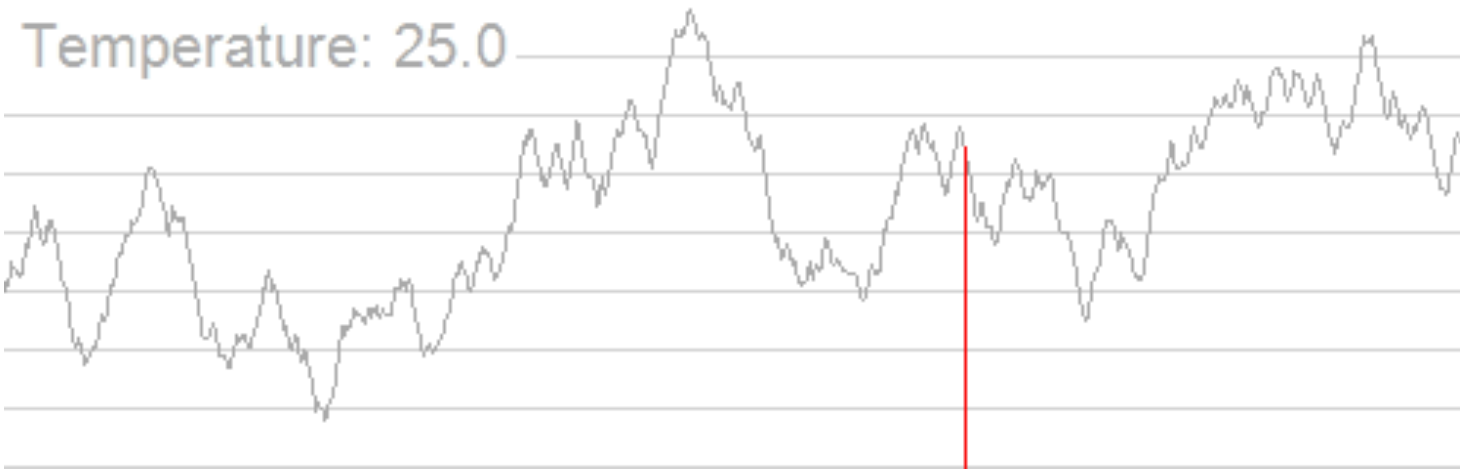
$$U(s) = \min_{a \in A(s)} Q(s,a)$$



Outline

- Stochastic games: the game itself is random
- Stochastic search: the game is deterministic, but we use randomness in the search, to find a fast approximate solution

Stochastic search



Hill_Climbing_with_Simulated_Annealing.gif, Public domain image, Kingpin13, 2013

Computational Complexity of Minimax & Alpha-Beta for Deterministic Games

- Computational complexity of minimax is $O\{b^d\}$
- Alpha-beta reduces the complexity, in the best case, to $O\{b^{d/2}\}$
- There is no way to do an exact search with better complexity, but…
- Stochastic search (a.k.a. Monte Carlo tree search) finds an approximate answer by randomly sampling from the possible moves

Stochastic search

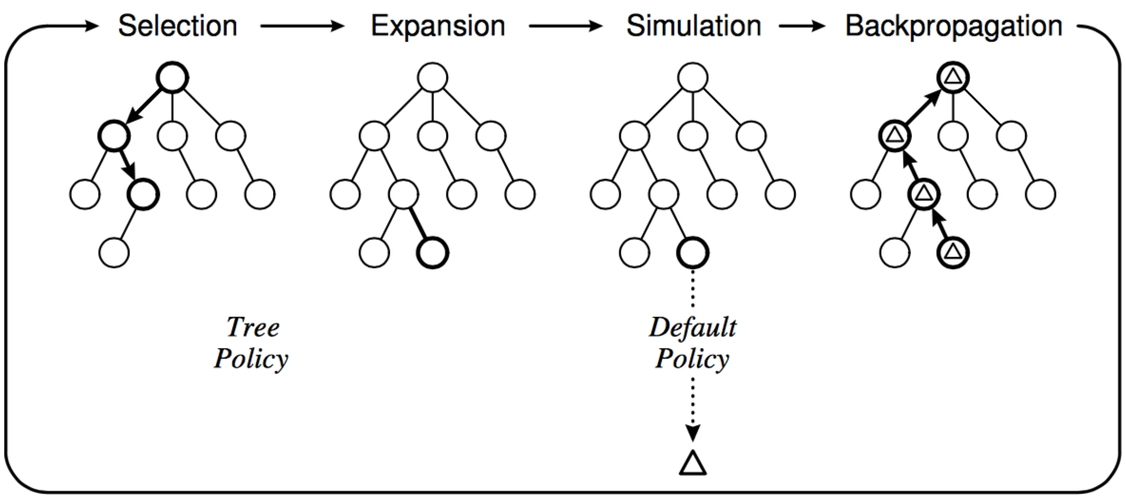
- An approximate solution: stochastic search

$$v(s) \approx \frac{1}{n}\sum_{i=1}^{n} v(i^{th} \text{ random game starting from } s)$$

- Asymptotically optimal: as $n \to \infty$, the approximation gets better.
- Controlled computational complexity: choose n to match the amount of computation you can afford.

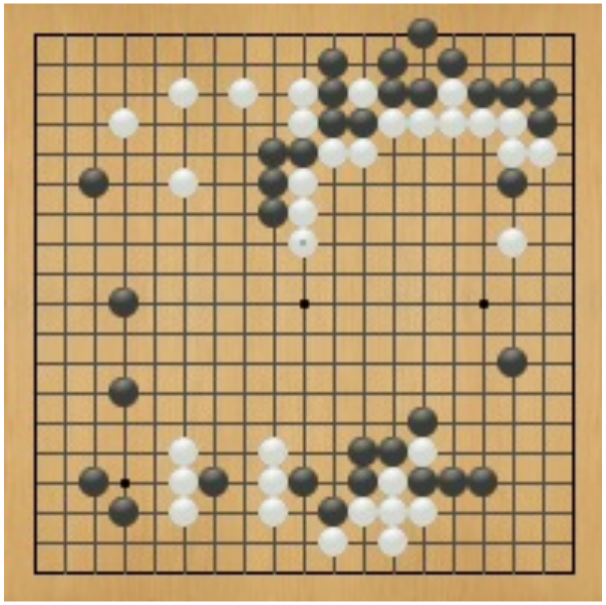
Stochastic search

- Depth-limited search out to level d, then random simulation for a few levels after that
- Starting at level d:
  - **Select**: choose the next state in the frontier
  - **Expand**: find all of its children
  - **Simulate**: play a random game from that node, to see what value results. Take that value to be the true value of this state
  - **Backpropagate**: use these values in a minimax search, over d levels, to find the best move



C. Browne et al., [A survey of Monte Carlo Tree Search Methods](), 2012

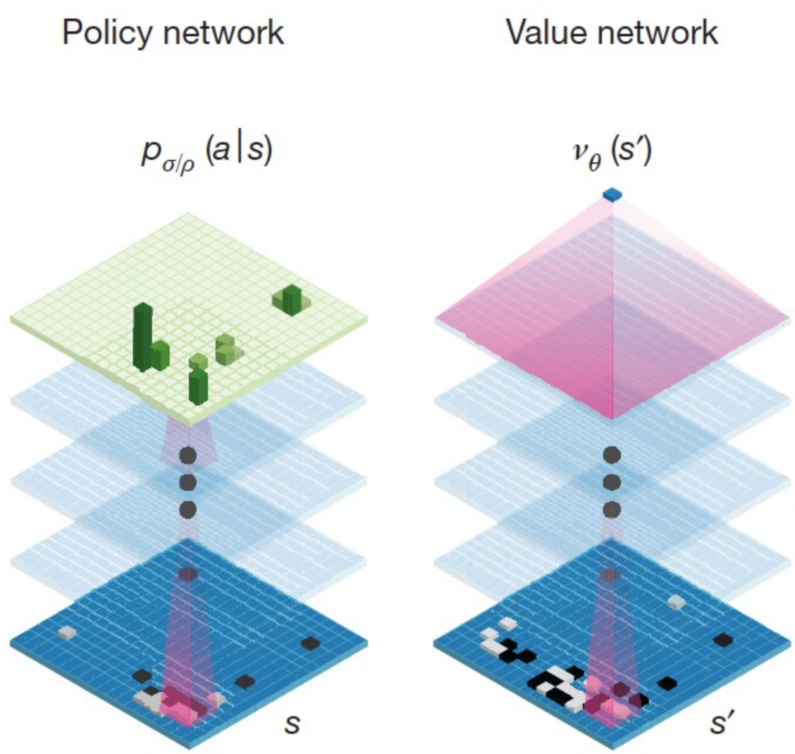
Case study: AlphaGo



Anton Ninno, Roy Laird, Ph.D.
antonninno@yahoo.com
roylaird@gmail.com
special thanks to Kiseido Publications

- **“Gentlemen should not waste their time on trivial games -- they should play Go.”**
- ***-- Confucius,***
- ***The Analects***
- ***ca. 500 B. C. E.***

AlphaGo

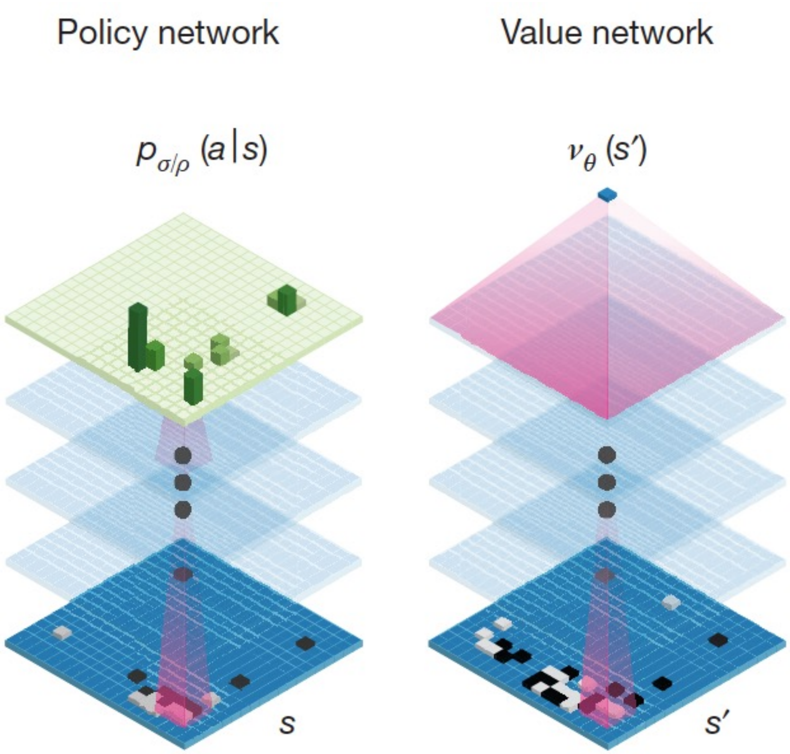


Deep convolutional neural networks

- Treat the Go board as an image
- Can be trained to predict distribution over possible moves (*policy*) or expected *value* of position

D. Silver et al., [Mastering the Game of Go with Deep Neural Networks and Tree Search](), Nature 529, January 2016

AlphaGo



- Policy network: Given a game state, s, predict what would be the best next move.
  - Input: game board as an image, s.
  - Output: $p(a|s)$, probability that action a is best.
- Value network: Given a game state, s, compute the expected value of the board for player 0 (MAX).
  - Input: game board as an image, s.
  - Output: $v(s)$, value of the game state.

D. Silver et al., [Mastering the Game of Go with Deep Neural Networks and Tree Search], Nature 529, January 2016

Stochastic Search in AlphaGo

- Each edge in the search tree has
  - Probabilities $p(a|s)$ computed by the policy network
  - State+Move values $Q(s, a)$ computed by the value network
  - Counts $N(s, a)$ specifying how many times that move has been tried
- Tree traversal policy selects actions randomly according to some combination of $p(a|s)$, $Q(s, a)$, and $N(s, a)$
- At the end of each simulation, values of the final boards are averaged in order to re-estimate the value of the initial move.

Stochastic Search in AlphaGo

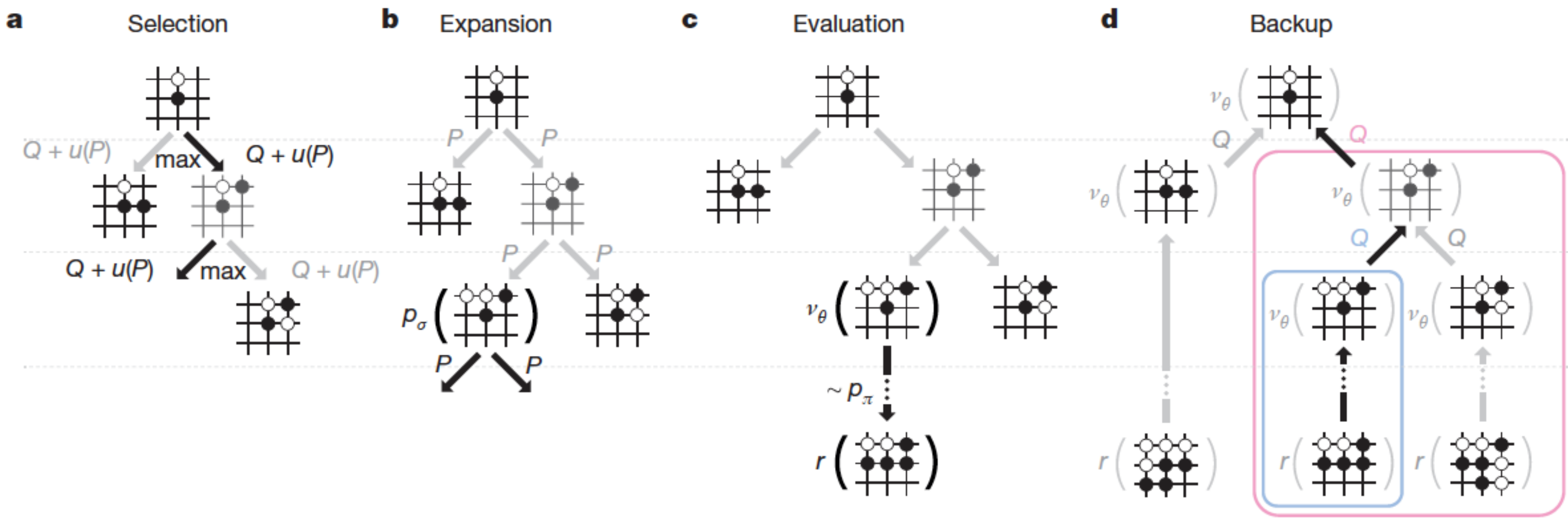


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

D. Silver et al., [Mastering the Game of Go with Deep Neural Networks and Tree Search](), Nature 529, January 2016

Conclusions

- Stochastic games: the game itself is random, so we need to use expectiminimax instead of minimax:

$$U(s) = \max_a \sum_{s'} P(s'|s,a)U(s')$$

$$U(s') = \min_{a'} \sum_{s''} P(s''|s',a')U(s'')$$

- Stochastic search: the game is deterministic, but we use randomness in the search, to find a fast approximate solution
  - Select and expand nodes as usual, using minimax
  - Simulate the leaf nodes: $v(s) \approx \frac{1}{n}\sum_{i=1}^{n} v(i^{th} \text{ random game starting from } s)$
  - Back-propagate using minimax