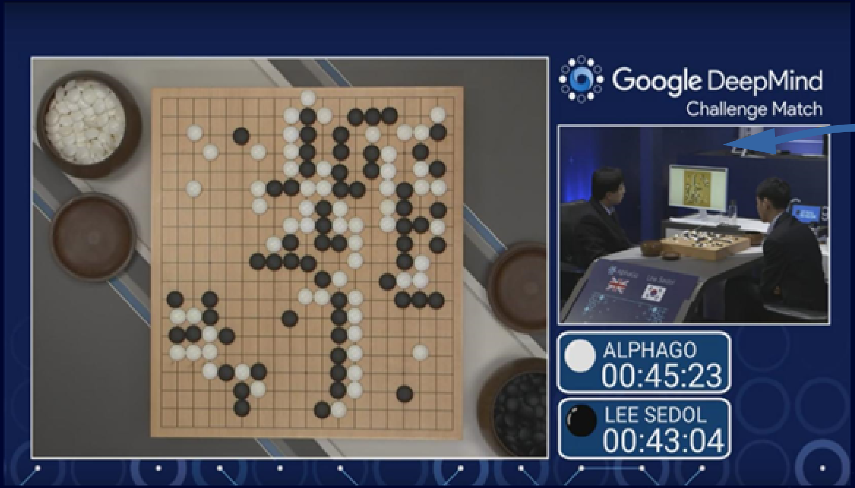


$$(\boldsymbol{p}, v) = f_\theta(s) \quad \text{and} \quad l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

AlphaZero

Last modified on 2022/05/21
by f.maire@qut.edu.au

Slides based on the article
Mastering the game of Go without human knowledge by David Silver et al. Published in Nature in 2017

Outline

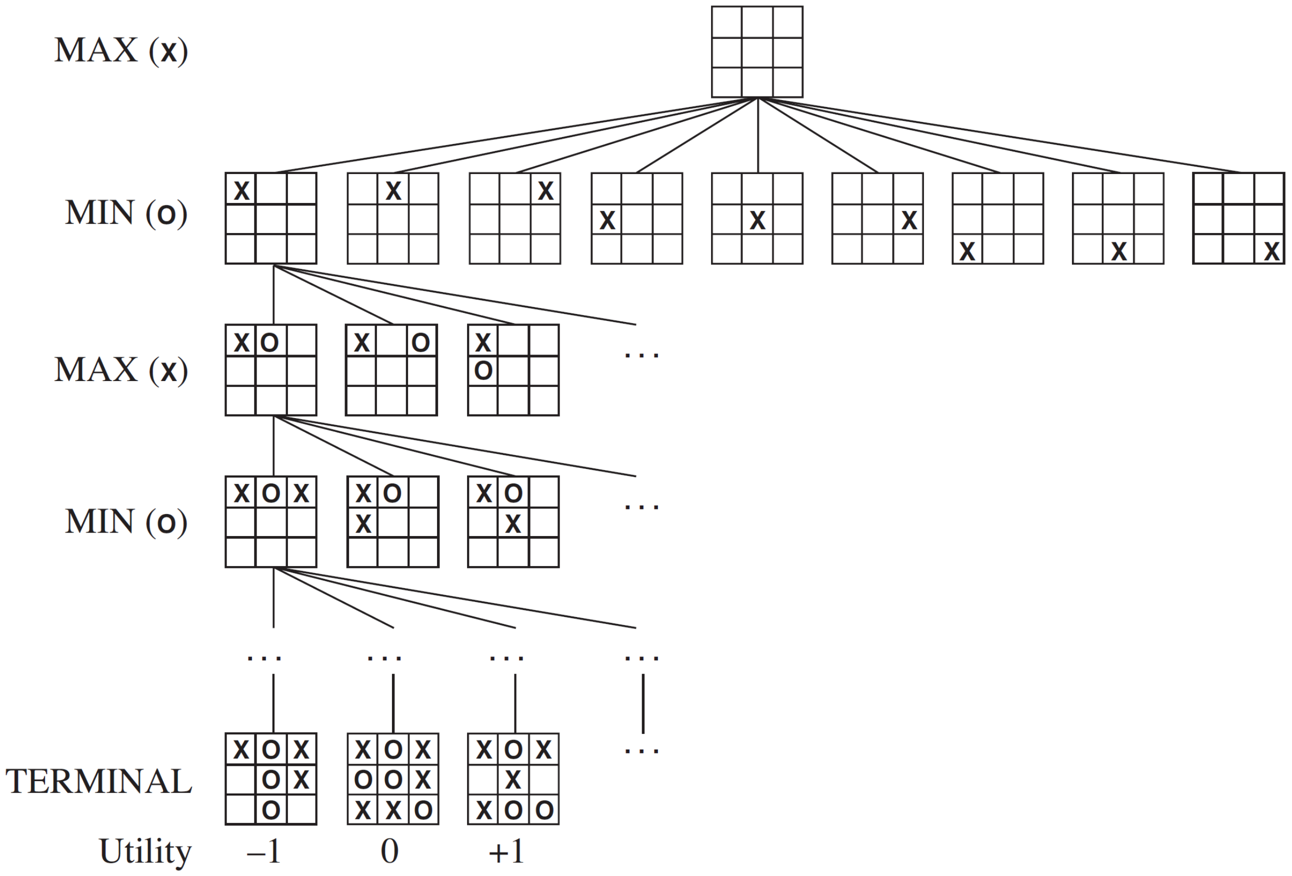
- Game trees
- MinMax
- TD leaf
- Monte Carlo Tree Search
- AlphaZero

Learning to Optimize Rewards

- In Reinforcement Learning, a software agent makes **observations** and takes **actions** within an **environment**, and in return it receives rewards.
- Its objective is to learn to act in a way that will **maximize** its **expected rewards** over time.

Refresher from last lecture

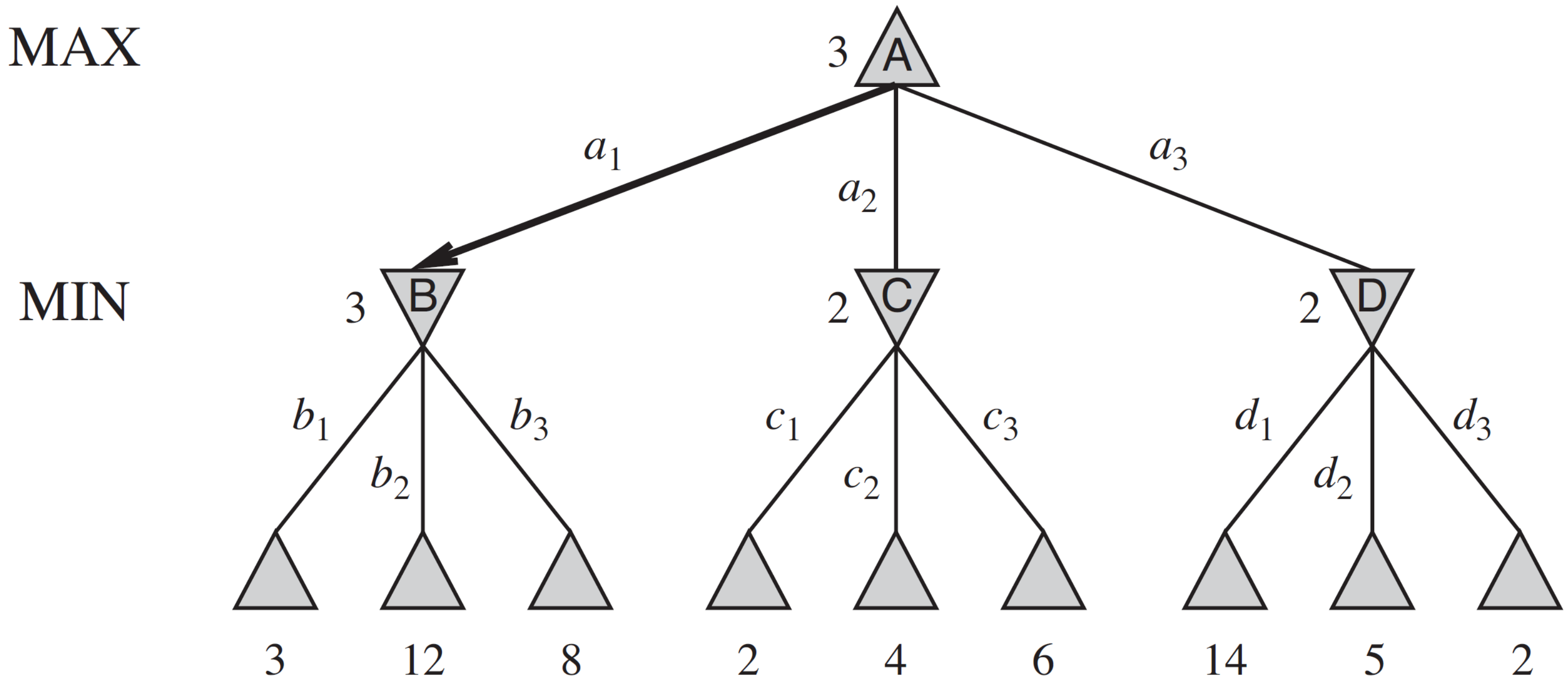
A (partial) game tree for Tic-Tac-Toe



Minimax Algorithm

- How to find the optimal action for a given possible game state?
  - Minimax value of a state: the value of the best outcome possible for the player who needs to move at the state assuming both players will take the best action in the corresponding states after the current move
  - Minimax Algorithm
    - Labeling the minimax value of each state given the minimax values of its successors
    - For player , take the max value of the values of its successors
    - For player , take the min value of the values of its successors
    - Essentially a backward induction algorithm (von Neumann & Morgenstern 1944)
    - Often implemented in a depth-first manner (recursive algorithm)

Minimax Algorithm



Minimax Algorithm

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

Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each a in ACTIONS(state) do
        $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow \infty$
    for each a in ACTIONS(state) do
        $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

Minimax Algorithm

- Intractable for large games
  - Chess(nodes), Go (nodes)
  - Cannot even represent the optimal strategy profile in space
- For many problems, you do not need a full contingent plan at the very beginning of the game
  - Can solve the problem in a more “online” fashion, just like how human players play Chess/Go: My opponent takes this action and what should I do now?

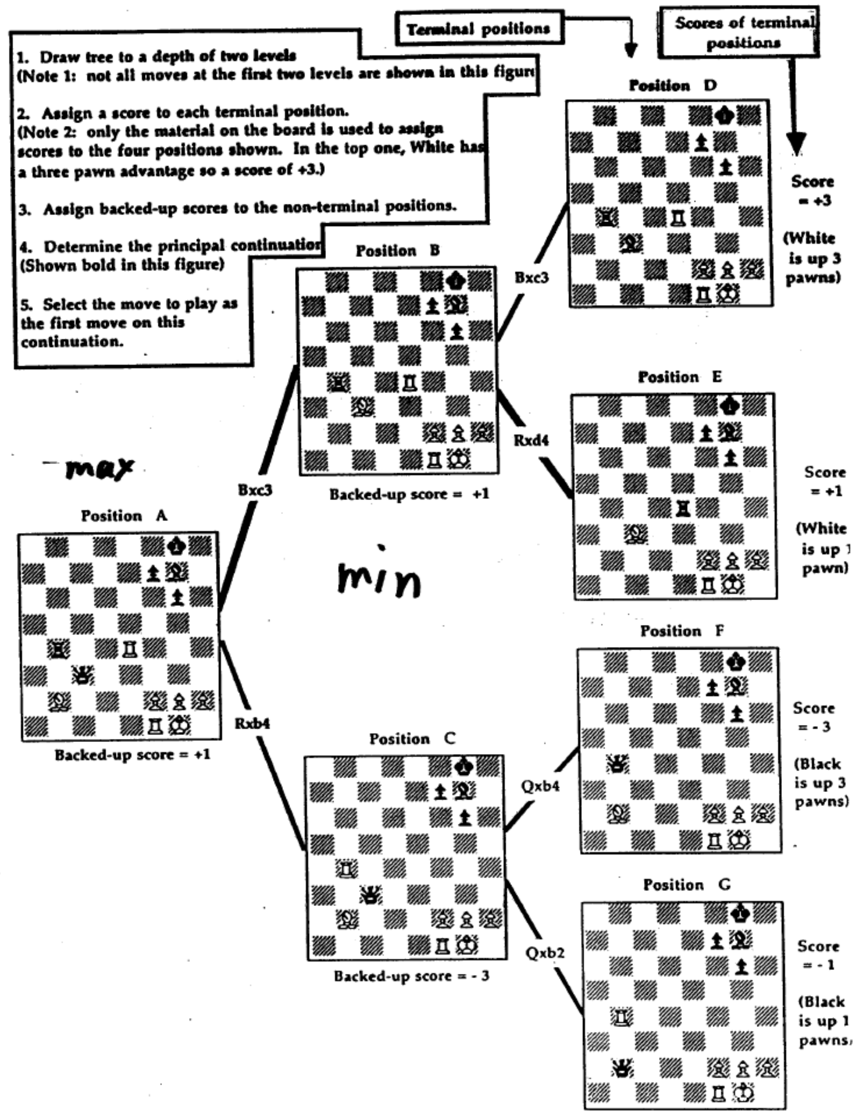
Minimax Algorithm

- If we only care about the game value, or the optimal action at a specific game state, can we do better?
  - Trick: Depth-limited search (Limit the depth)
- Minimax algorithm (Shannon, 1950):
  - Set (an integer) as an upper bound on the search depth
  - the static evaluation function, returns an estimate of minimax value of state

Minimax Algorithm

Claude Shannon, Alan Turing: Minimax search with scoring function (1950)

Only show a few branches here


1. Draw tree to a depth of two levels
(Note 1: not all moves at the first two levels are shown in this figure

2. Assign a score to each terminal position.
(Note 2: only the material on the board is used to assign scores to the four positions shown. In the top one, White has a three pawn advantage so a score of +3.)

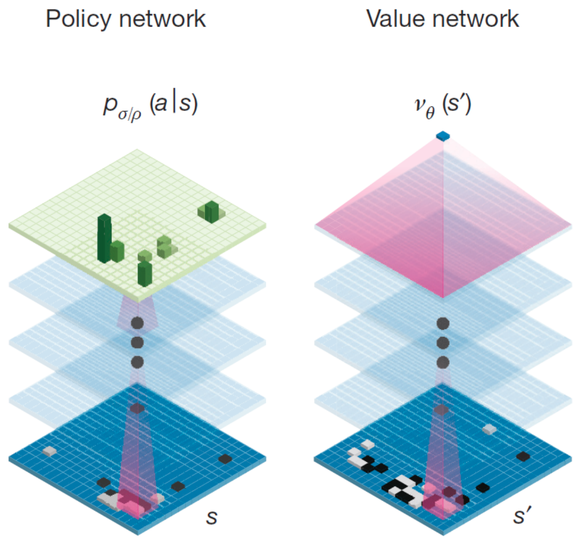
3. Assign backed-up scores to the non-terminal positions.

4. Determine the principal continuation
(Shown bold in this figure)

5. Select the move to play as the first move on this continuation.

Connections with RL

- Learning / Estimating the **state value function** will help the MAX player find the optimal policy against a perfectly rational MIN player



TDLeaf(λ): Combining Temporal Difference Learning with Game-Tree Search

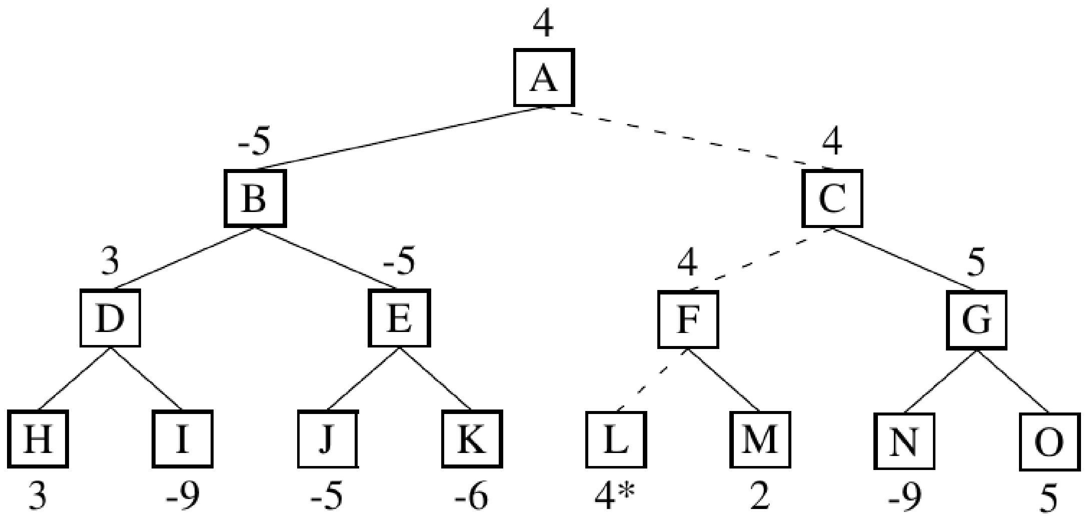


Fig. 1: Full breadth, 3-ply search tree illustrating the minimax rule for propagating values. Each of the leaf nodes (H–O) is given a score by the evaluation function, $\tilde{J}(\cdot, w)$. These scores are then propagated back up the tree by assigning to each opponent's internal node the minimum of its children's values, and to each of our internal nodes the maximum of its children's values. The principle variation is then the sequence of best moves for either side starting from the root node, and this is illustrated by a dashed line in the figure. Note that the score at the root node A is the evaluation of the leaf node (L) of the principal variation. As there are no ties between any siblings, the derivative of A's score with respect to the parameters w is just $\nabla \tilde{J}(L, w)$.

MCTS tree growth

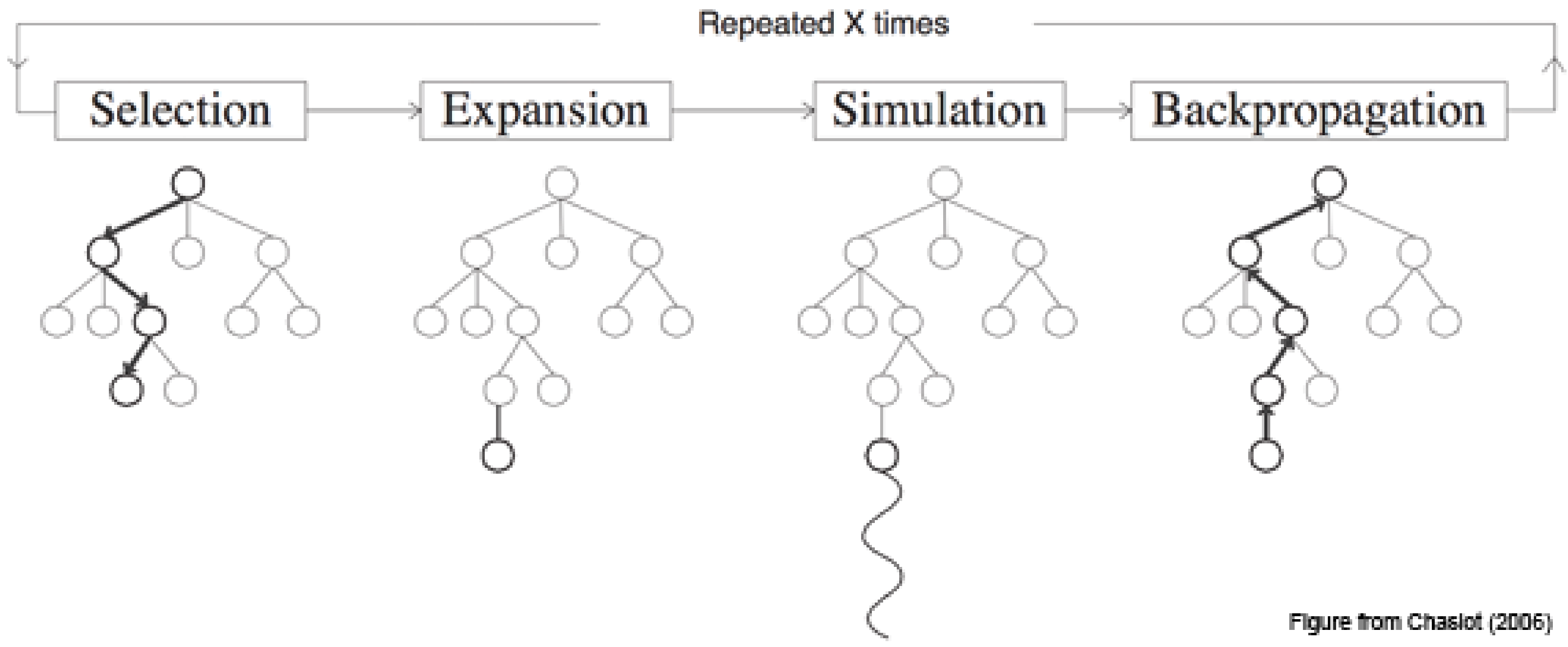
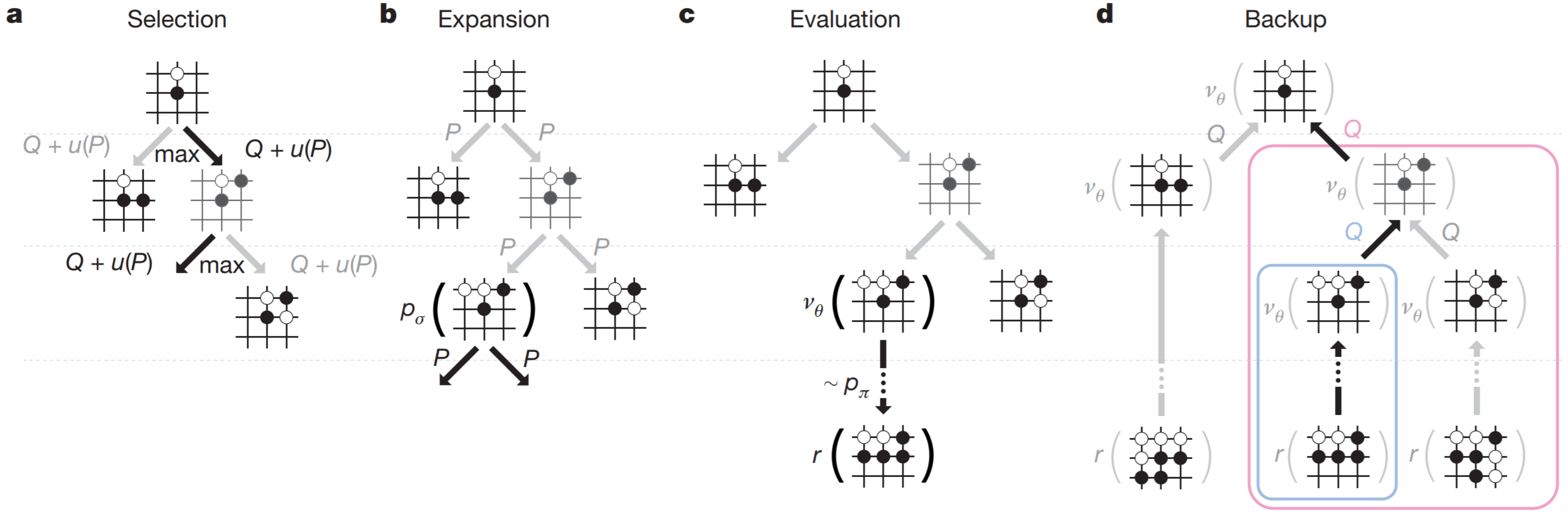


Figure from Chaslot (2006)

How AlphaGo works

- Supervised learning + Reinforcement learning
- Monte-Carlo Tree Search



upper confidence bound $Q(s, a) + U(s, a)$, where $U(s, a) \propto P(s, a) / (1 + N(s, a))$