CS 540 Introduction to Artificial Intelligence
Review for Search, Game and RL

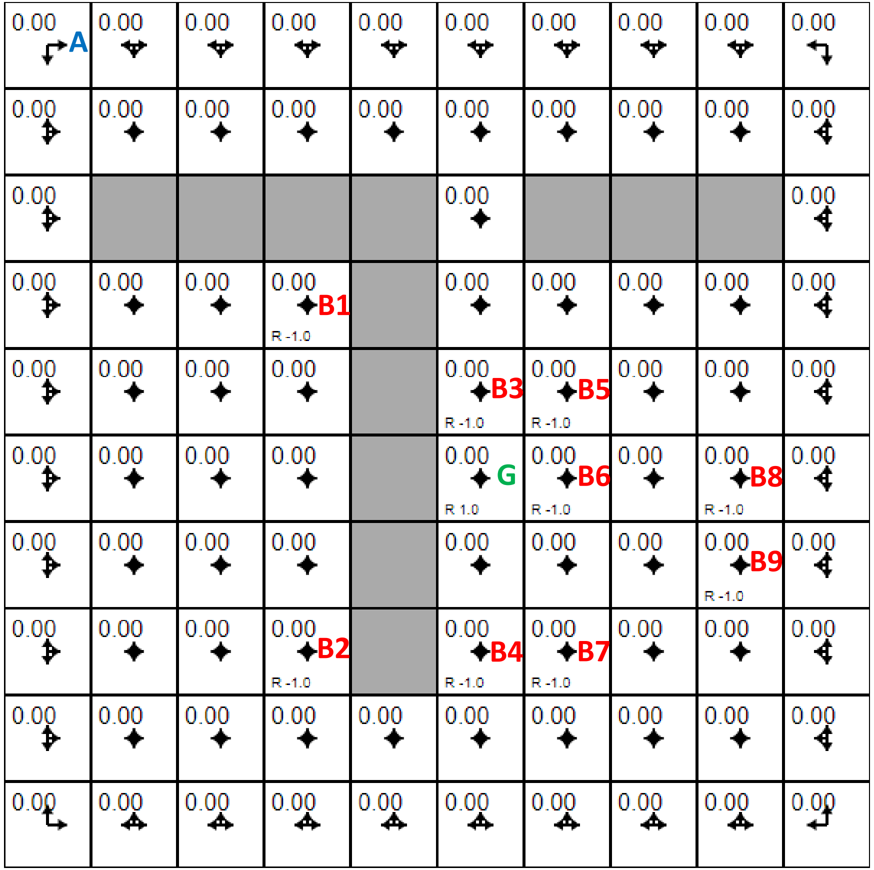
Yudong Chen
University of Wisconsin-Madison

Dec 9, 2021

Announcements

- **Please fill out couse evaluation survey**
- **Homework**:
  - HW10 due next Tuesday (before last class)
- **Final exam:** Dec 20, 2:45-4:45pm, online
- **Class roadmap:**
  - Today: **Demonstration for RL**; Review on search, games, RL
  - Next Tuesday: Ethics and Trust in AI

Demonstration: GridWorld

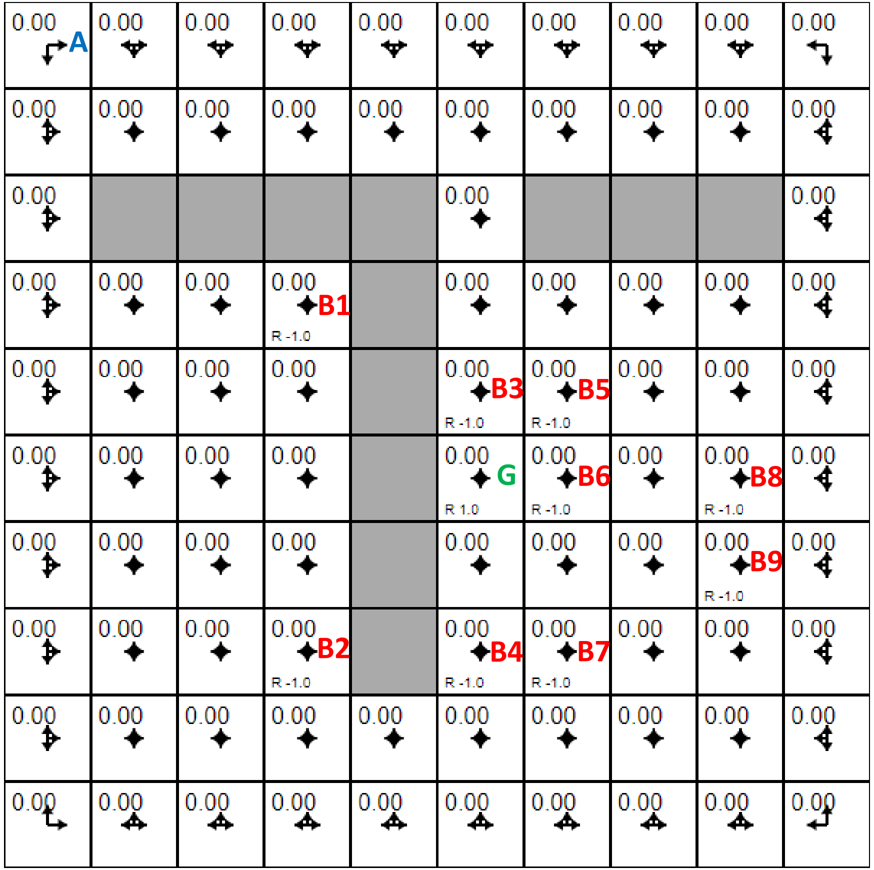


https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Note:

- Transition is **deterministic** (robot moves exactly as told)
- Game **does *not* terminate**
  - Reaching **B**'s: pay -1 and game continues
  - Reaching **G**: get +1 and robot teleports to initial state **A**
- Discount factor = 0.9

What do optimal value/policy look like?



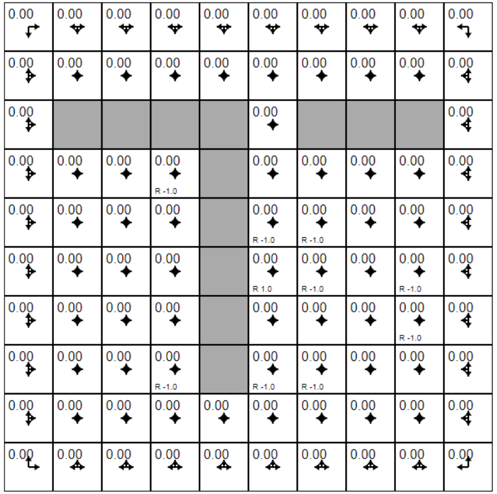
Let's guess:

- Optimal route?
- $V^*(\mathbf{B8})$ vs $V^*(\mathbf{B9})$?
- $V^*(\mathbf{A}) = 0.22$. Then $V^*(\mathbf{G}) = ?$
- $V^*(\mathbf{B3}) \approx ?$
- If reward(**B3**) changes to -0.5, should we go through it?

Visualization of Q Learning and ϵ-Greedy

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

- "Reinit agent" resets the board
- "Toggle TD Learning" starts or stops the agent running



Review: Outline

- Search
  - Uninformed vs Informed
  - Optimization
- Games
  - Game theory basics, dominant strategy, equilibrium
  - Minimax search
- Reinforcement Learning
  - MDPs, value iteration, Q-learning

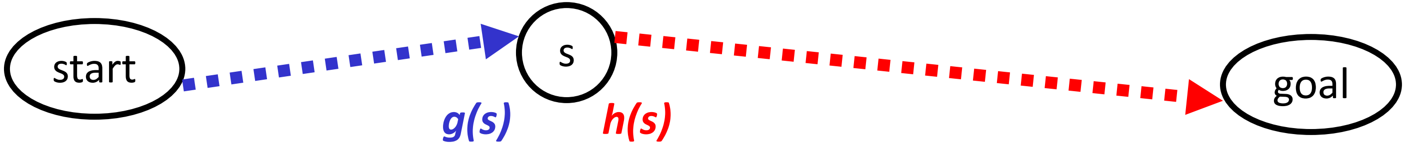
Uninformed vs Informed Search

Uninformed search (all of what we saw). Know:

- Path cost ***g***(*s*) from start to node *s*
- Successors.

Informed search. Know:

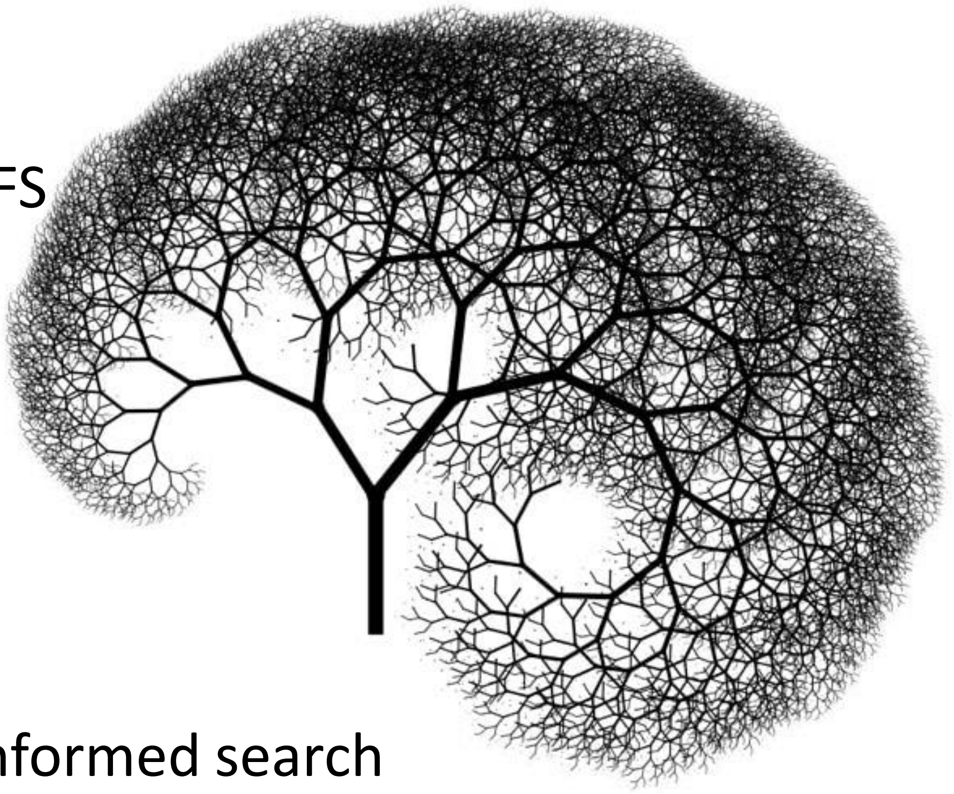
- All uninformed search properties, plus
- Heuristic **h(s)** from s to goal (recall game heuristic)



Uninformed Search: Iterative Deepening DFS

Repeated limited DFS

- Search like BFS, fringe like DFS
- **Properties**:
  - Complete
  - Optimal (if edge cost 1)
  - Time O(b^d)
  - Space O(bd)



- **Preferred algorithm** for uninformed search

Fractalsaco

Performance of Search Algorithms on Trees

b: branching factor (assume finite)   d: goal depth   m: graph depth

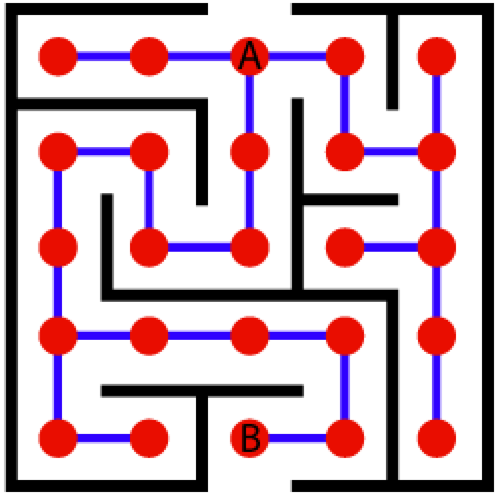
	Complete	optimal	time	space
Breadth-first search	Y	Y, if [1]	$O(b^d)$	$O(b^d)$
Uniform-cost search[2]	Y	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Depth-first search	N	N	$O(b^m)$	$O(bm)$
Iterative deepening	Y	Y, if [1]	$O(b^d)$	$O(bd)$

1. edge cost constant, or positive non-decreasing in depth
2. edge costs $\geq \varepsilon > 0$. C^* is the best goal path cost.

Informed Search: **A* Search**

A*: Expand best $g(s)$ + $h(s)$, with one requirement

- Demand that $h(s) \leq h^*(s)$
- If heuristic has this property, “admissible”
  - Optimistic! Never over-estimates
- Still need $h(s) \geq 0$
  - Negative heuristics can lead to strange behavior

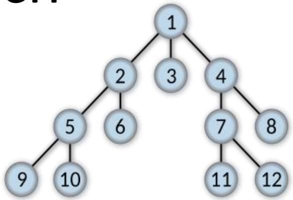


V. Batoćanin

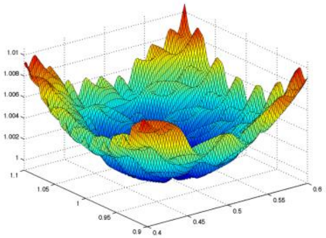
Search vs. Optimization

Before: wanted a path from start state to goal state

- Uninformed search, informed search



Wiki



TuringFin

New setting: optimization

- States *s* have values ***f***(*s*)
- Want: *s* with optimal value ***f***(*s*) (i.e, **optimize** over states)
- Challenging setting: **too many states** for previous search approaches, but maybe not a continuous function for SGD.

Hill Climbing Algorithm

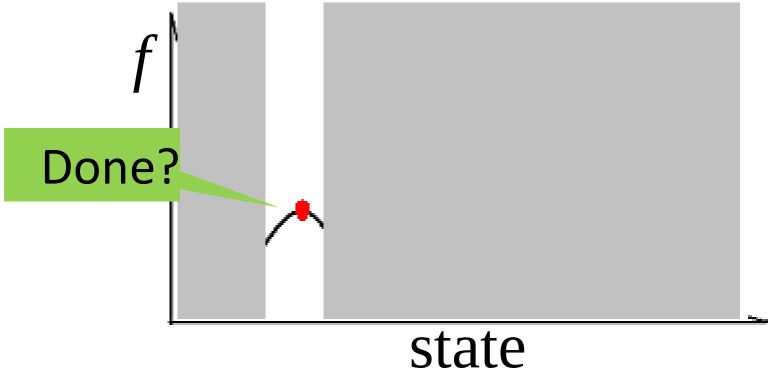
Pseudocode:

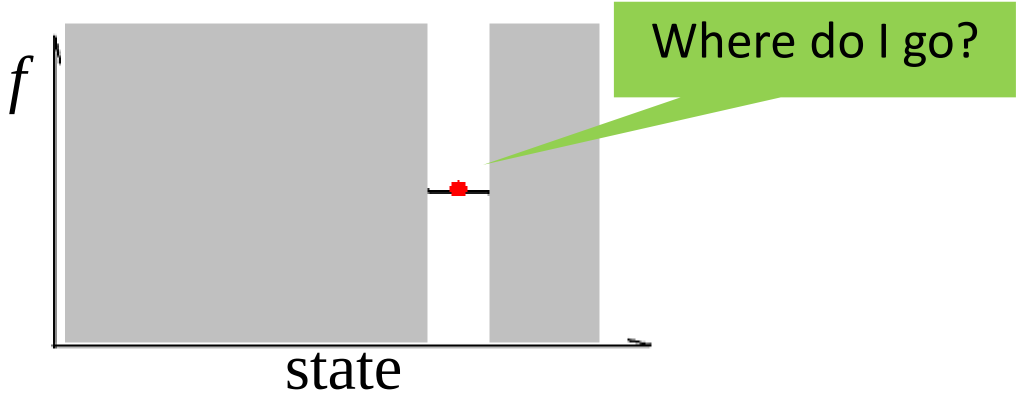
1. Pick initial state s
2. Pick t in **neighbors**(s) with the largest $\boldsymbol{f}(t)$
3. if $\boldsymbol{f}(t) \leq \boldsymbol{f}(s)$ THEN stop, return s
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

Hill Climbing: Local Optima

Note the **local optima**. How do we handle them?





Simulated Annealing

A more sophisticated optimization approach.

- **Idea**: move quickly at first, then slow down
- Pseudocode:

Pick initial state x
For k = 0 through k_{max}:
    Reduce temperature $\boldsymbol{T}$
    Pick a random neighbour, $y \leftarrow$ neighbor(x)
    If $\boldsymbol{f}$(y) ≥ $\boldsymbol{f}(x)$, then $x \leftarrow y$
The interesting bit
    Else, with prob. $P(\boldsymbol{f}(x), \boldsymbol{f}(y), \boldsymbol{T})$ then $x \leftarrow y$
Output: the final state x

Simulated Annealing: Picking Probability

How do we pick probability *P*?

$$P(x, y, \boldsymbol{T}) = \exp\left(-\frac{|f(x) - f(y)|}{\boldsymbol{T}}\right)$$

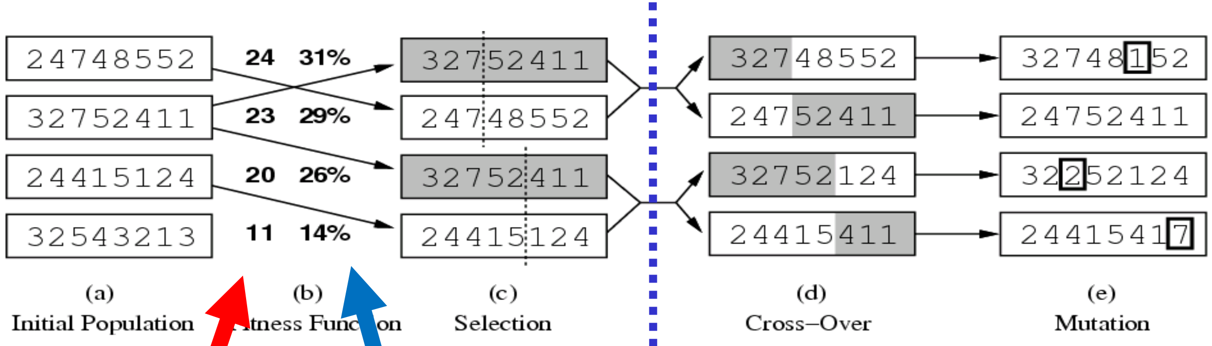
- Decrease with gap $|\boldsymbol{f}(x) - \boldsymbol{f}(y)|$
- Decrease with time *k*

- Temperature ***T*** cools over time
  - High temperature, accept any *y*
  - Low temperature, behaves like hill-climbing
  - Still, $|\boldsymbol{f}(x) - \boldsymbol{f}(y)|$ plays a role: if big, replacement probability low.

Genetic Algorithms

Goal of genetic algorithms: optimize using principles inspired by mechanism for evolution

- E.g., analogous to **natural selection, cross-over**, and **mutation**



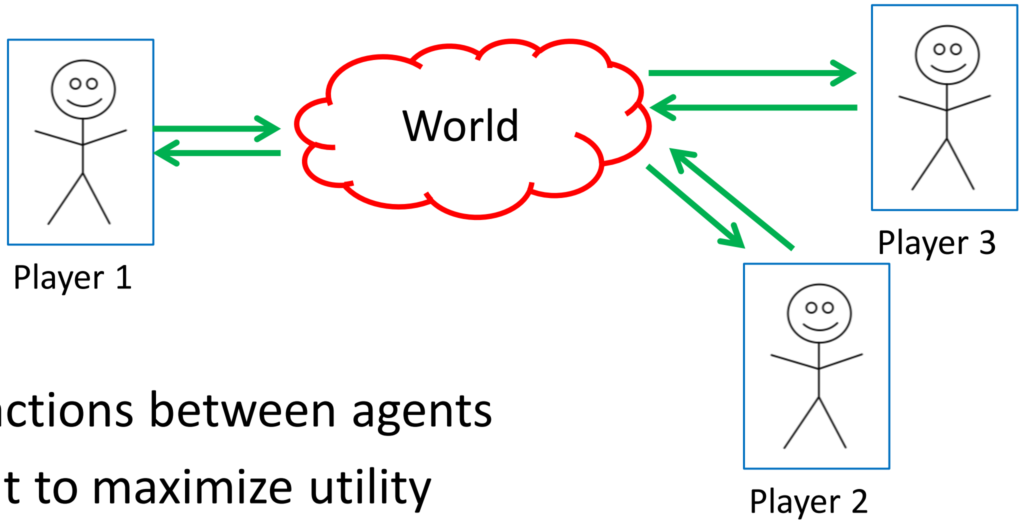
# of non-attacking pairs

prob. reproduction ∝ fitness

→ Next generation

Games Setup

Games setup: **multiple** agents



- Now: interactions between agents
- Agents want to maximize utility
- **Strategic** decision making.

Modeling Games: Properties

Let's work through **properties** of games

- **Number** of agents/players
- State & action spaces: **discrete** or **continuous**
- **Finite** or **infinite**
- **Deterministic** or **random**
- **Sum**: zero or positive or negative
- **Sequential** or **simultaneous**

Wiki

Simultaneous Games: Normal Form

- n players $\{1,2,\ldots,n\}$
- Player i strategy a_i from A_i.
  - Strategy of **all** players: $a = (a_1, a_2, \ldots, a_n)$
- Player i gets rewards $u_i(a)$ for any outcome
  - **Note**: reward depends on other players!

Player 2 Player 1	*Stay silent*	*Betray*
Stay silent	−1, −1	−3, 0
Betray	0, −3	−2, −2

Dominant Strategies and Equilibria

- a_i is **dominant** if a_i better than a_i' *regardless* of what other players do

$$u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i}) \forall a_i' \neq a_i \text{ and } \forall a_{-i}$$

- $a^* = (a_1^*, \dots, a_n^*)$ is an **equilibrium** if all the players do not have an incentive to *unilaterally deviate*

$$u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*) \quad \forall a_i \in A_i$$

- A *mixed strategy* $x^* = (x_1^*, \dots, x_n^*)$ is a **Nash equilibrium** if

$$u_i(x_i^*, x_{-i}^*) \geq u_i(x_i, x_{-i}^*) \quad \forall x_i \in \Delta_{A_i}, \forall i \in \{1, \dots, n\}$$

Dominant Strategies and Equilibria

- Dominant strategies $\Longrightarrow$ (Pure) Equilibrium $\Longrightarrow$ NE
  - Not the other way around

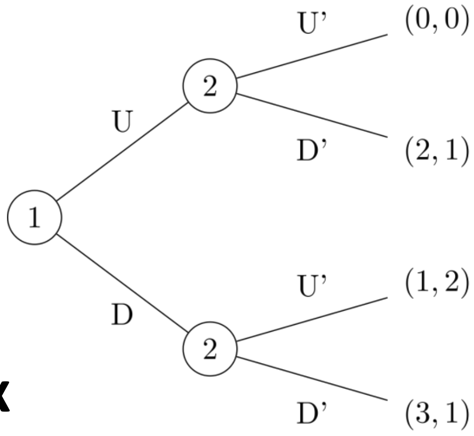
- NE always exists. Not necessarily for the other two

i

Sequential Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- Perform search over the tree



Wiki

- Can still look for Nash equilibrium
  - Or, other criteria like **maximin** / **minimax**

Minimax Value and Strategies

Let's stick to zero-sum two-player games

- Write down all the pure strategies (e.g., the big tree) and select the *s** and *t**

$$s^* = \arg\max_{s \in S} \min_{t \in T} u(s,t) \qquad t^* = \arg\min_{t \in T} \max_{s \in S} u(s,t)$$

- Can implement this as depth-first search: **minimax algorithm**

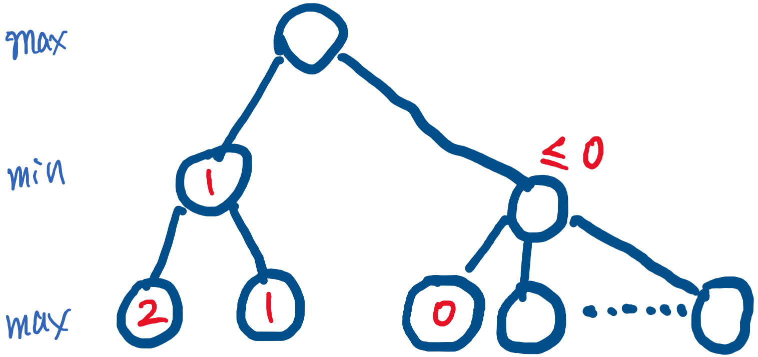
Minimax Search with α-β pruning

```
function Max-Value (s,α,β)
inputs:
    s: current state in game, Max about to play
    α: best score (highest) for Max along path to s
    β: best score (lowest) for Min along path to s
output: min(β , best-score (for Max) available from s)

    if ( s is a terminal state )
    then return ( terminal value of s )
    else for each s’ in Succ(s)
      α := max( α , Min-value(s’,α,β))
      if ( α ≥ β ) then return β   /* alpha pruning */
    return α
```

```
function Min-Value(s,α,β)
output: max(α , best-score (for Min) available from s )

    if ( s is a terminal state )
    then return ( terminal value of s)
    else for each s’ in Succs(s)
      β := min( β , Max-value(s’,α,β))
      if (α ≥ β ) then return α   /* beta pruning */
    return β
```

Starting from the root:

Max-Value(root, -∞, +∞)



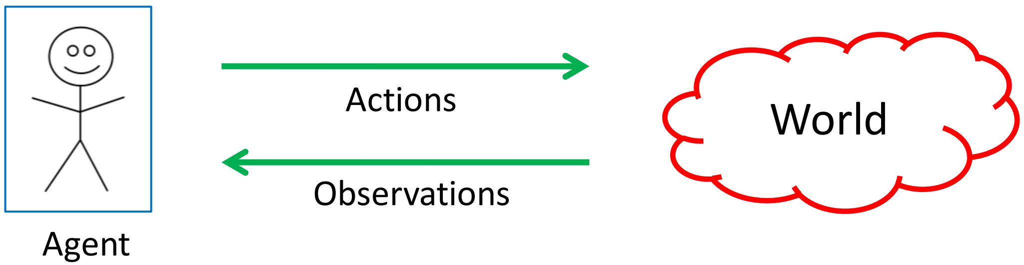
Minimax Search with Heuristics

- Long games are yield huge computation
- To deal with this: limit ***d*** for the search depth
- **Q**: What to do at depth *d*, but no termination yet?
  - **A**: Use a heuristic evaluation function *e(x)*

function MINIMAX(x, d) **returns** an estimate of x's utility value
  inputs: x, current state in game
      d, an upper bound on the search depth
  if x is a terminal state **then return** Max's payoff at x
  else if $d = 0$ **then return** $e(x)$
  else if it is Max's move at x **then**
    return $\max\{\text{MINIMAX}(y, d-1) : y \text{ is a child of } x\}$
  else return $\min\{\text{MINIMAX}(y, d-1) : y \text{ is a child of } x\}$

Credit: Dana Nau

Reinforcement Learning



Basic setup:

- Set of states, S
- Set of actions A
- Information: at time t, observe state $s_t \in$ S. Get reward r_t
- Agent makes choice $a_t \in$ A. State changes to s_{t+1}, continue

Goal: find a map from **states to actions** maximize rewards.

A “policy”

Markov Decision Process (MDP)

The formal mathematical model:

- **State set** S. Initial state s_0. **Action set** A
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \to A$ action to take at a particular state.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Value function

The **value function** for policy π at state s_0 is the **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(\text{sequence})U(\text{sequence})$$

where the utility of a sequence is its corresponding **discounted cumulative reward**:

$\gamma \in (0,1)$: discount factor

$$U(s_0, s_1 \ldots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t \geq 0} \gamma^t r(s_t)$$

Bellman Equation

- Set $V^*(s)$ to be value function for **optimal** policy.
- $V^*(s)$ satisfies the Bellman Equation: for all s,

$$V^*(s) = r(s) + \gamma \underbrace{\max_a \sum_{s'} P(s'|s,a)V^*(s')}_{\text{Discounted expected future } \textbf{rewards}}$$

Current state reward

Value Iteration

Q: How do we find $V^*(s)$?

- Know: reward ***r***(*s*), transition probability P(*s*'|*s*,*a*)
- Also know $V^*(s)$ satisfies Bellman equation

A: Start with $V_0(s)=0$, $\forall s$. Then for all *s*, update

$$V_{i+1}(s) = r(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a) V_i(s')$$

From Optimal Value to Optimal Policy

Now that $V^*(s_0)$ is known, what a should we take?

- What's the expected utility of an action a in state s?

$$\sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards

- So, to get the optimal policy, compute

$$\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

Q-Learning

What if we don't know transition probability P(s'|s,a)?

- **Q-learning**: get an action-value function Q(s,a) that tells us the value of doing a in state s
- How do we get Q(s,a)? Similar iterative procedure:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Learning rate

SARSA

An alternative:

- Just use the next action, no max over actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Learning rate

- Called state–action–reward–state–action (**SARSA**)

Epsilon-Greedy Policy

Need to balance **exploitation** and **exploration**

- With some 0<ε<1 probability, take a random action at each state, or else the action with highest Q(*s*,*a*) value.

$$a = \begin{cases} \mathrm{argmax}_{a \in A} Q(s, a) & \mathrm{uniform}(0,1) > \epsilon \\ \mathrm{random}\ a \in A & \mathrm{otherwise} \end{cases}$$

- Can be used in Q Learning and SARSA

Acknowledgements: Based on slides from Fred Sala, Yin Li, Jerry Zhu, Svetlana Lazebnik, Yingyu Liang, David Page, Mark Craven, Pieter Abbeel, Dan Klein