Administration

- Assignment 2: just a programming assignment.
- Midterm: posted by next week (5), will cover:
    - Lectures
    - Readings

  A midterm review sheet will be posted on the website. A forum will be opened dedicated for questions about the midterm. Next week there will not be a lecture, but a session for reviewing question about the midterm. Midterm release date will be announced.

  The midterm will be posted as PDF, and we will have a week to complete it in our own time (spend no more than 3 hours on it!)

  We might get extra credit for submitting the midterm in Latex; that will be noted in the midterm instructions.

Overview

- Game theory:
- Games: adversarial games (aka adversarial search)

Game Theory

Game theory: a formal way to analyze interactions among a group of **rational** players / agents who behave strategically. The base assumption is that the players act rationally. GT has many applications (economics…)

Games are a form of multi-agent environment, where we have multiple players playing each with a different objective function. Each agent may have a different objective, which may not be compatible with another's - Corporative vs. competitive.

Games are generally used for making choices in situations where we don't have perfect information.

Each agent is completely self-interested, only maximizing their own personal objective.

Relation of games to search:

In search there's no adversary. The evaluation function is an estimate of the cost of the path to the goal.

In adversarial games:

- The solution is a **strategy**, e.g. in chess: if I see a certain configuration of pieces, I take this type of move – something defined before even start playing in most cases.
- The evaluation function is of the "goodness" of the game position

Types of games:

Can be classified based on:

- Is there randomness in the game / is it deterministic. For instance chess is deterministic, so the successor states are always known in advance, as opposed for instance to monopoly, where the dice add randomness.
- Perfect / imperfect information – in chess and monopoly we know everything about the world; in poker for instance, you can't see the cards in the opponent's hands.

Example of deterministic imperfect information game: battleship.

Assumptions:

Features of a game:

- At least two rational player
- Each player has more than one choice
- There are strategic interactions: outcome depends on strategies chosen by all players

Example: six people go to a restaurant; when each person pays for his own meal, that's a single agent decision problem. If before the meal they all agree to evenly split the game, each player has to decide whether to agree to that decision beforehand, depending on what they will benefit from that (how much food they plan to order).

More assumptions:

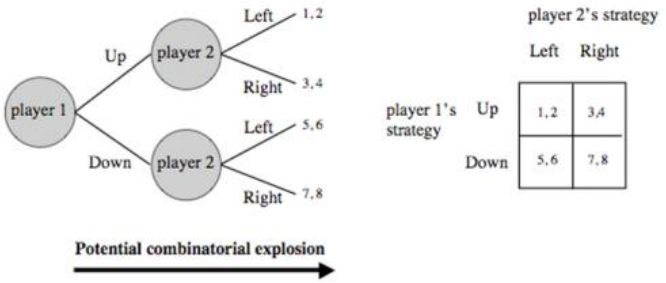
- Simultaneous move:
  - Everyone make a move at the same time – each player chooses his own strategy with no knowledge of the strategy of the other players; no cooperation.
  - Each player receives his payoff at the end of the game
- Complete information:
  - Strategies and payoffs are common knowledge among all players
- Assumptions on players:
  - Aim to maximize their payoff
  - They are rational and know the others are rational

Formal definitions:

- Players $\mathcal{P} = \{P_1, \dots, P_n\}$
- Actions $\mathcal{S} = \{S_1, \dots, S_n\}$ that the players can take
- Payoff matrix $\mathcal{M}$: for any assignment of actions to players, the payoff functions maps payoffs to players
  - Each player chooses an action $s_1 \in S_1, \dots, s_n \in S_n$
  - $\mathcal{M}(s_1, \dots, s_n) \rightarrow \{u_1, \dots, u_n\}$ where u_i is the payoff for player P_i

Game representation

- Extensive form: create a search tree that enumerates all possibilities of the game flow. There's a layer for each player.
- Matrix form: normal strategic form; player on each axis



The extensive form is more comfortable for multiplayer game representation.

When the sum of all possible actions is 0, it is a 0-sum game, otherwise it's just a general sum game.

Definition: strategy:

Specifies probabilistically the action the player should take. Let π be the strategy for player i:

- $\pi_i(s)$: the probability player i chooses action s by strategy π.
- If there exists a s such that $\pi_i(s) = 1$, it is a pure strategy, otherwise: a mixed strategy

The strategy is decided beforehand, the actual action selection is done during the game.

Definition: strategy profile:

Π is a collection of strategies for player i.

Definition: expected value:

Expected value or reward of a game for player i: $\sum_{s_i \in S_i} \sum_{s_j \in S_j} \Pr[s_i, s_j]\, u_i(s_i, s_j)$

We sum over the probability of a pair of actions being taken vs. a reward.

For instance, for player 1 in rock-paper-scissors with π_1, π_2 giving $\frac{1}{3}$ for each choice, that value would be 0, since this example is symmetric: $\frac{1}{3} \cdot \frac{1}{3} \cdot -1 + \cdots = 0$

Definition: best response strategy:

π_i is the best response for agent i if given strategies for other agents, π_i maximizes the expected value for agent i.

Example: $\pi_j(b_0) = 0.2, \pi_j(b_1) = 0.8$

		a0	a1
			Player i
Player j	b0	10 ,10	0 , 0
	b1	0 , 0	12 , 12

We can ignore the 0-valued cells as they don't contribute E_i, so:

$$E_i = 0.2 \cdot \pi_i(a_0) \cdot 10 + 0.8 \cdot \pi_i(a_1) \cdot 12 = 2\pi_i(a_0) + 9.6\pi_i(a_1)$$

And we need to maximize this term by selection of π_i. The constraint we add is:

$$\pi_i(a_0) + \pi_i(a_1) = 1 \Rightarrow \pi_i(a_1) = \big(1 - \pi_i(a_0)\big)$$

Denote $x \coloneqq \pi_i(a_0)$, so:

$$f(x) = 2x + 9.6(1 - x) = -7.6x + 9.6$$

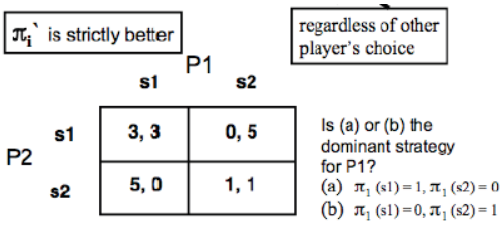
And $f(x)$ is maximized when $x = \pi_i(a_0) = 0$, therefore it is a pure strategy where $\pi_i(a_0) = 0, \pi_i(a_1) = 1$

Dominated strategies:

π_i is *strictly* dominated by π_i' if:

- $u_i(\pi_i, \pi_j) < u_i(\pi_i', \pi_j)$ for all π_j – the value for π_i is always better than π_i' for any opponent's strategy π_j

Example:



For instance if 1 always chooses s_2, then:

- If 2 chooses s_1, $u_i = 5$, and if he chooses s_2 then $u_i = 1$ – always guaranteed a value of at least 1

If 1 chooses s_1 always:

- If 2 chooses s_1, $u_i = 3 < 5$ and if he chooses s_2 then $u_i = 0 < 1$

So 1 choosing s_1 always dominates always choosing s_2.

Prisoner's Dilemma:

- 2 suspects held in separate cells
    - If both keep quiet, both sentenced to 1 month in jail
    - If both rat on the other, both sentenced to 3 months
    - If one rats on the other, he is freed and the other gets 5 months

What is the dominant strategy:

1\2	Quiet	Rat
Quiet	-1 \ -1	-5 \ 0
Rat	0 \ -5	-3 \-3

So the dominant strategy would be to always rat on the other guy because that will always give that player the better deal – either 0 vs. -1 if the other is quiet, or -3 vs -5 if the other guy rats.

Dominant strategy equilibrium:

Each agent picks his dominant strategy. Requires no counter-speculation, but doesn't always exist, so that's Nash Equilibrium.

Nash Equilibrium

		Player 2		
		L	C	R
Player 1	T	0 , 4	4 , 0	5 , 3
	M	4 , 0	0 , 4	5 , 3
	B	3 , 5	3 , 5	6 , 6

If player 2 assumes player 1 always choose B, player 2 has no incentive to deviate from choosing R, since the alternatives are less profitable. NE is when no player has an incentive to change his strategy, but that may not always be the case.

Or: a set of strategies, one per player, such that each player's strategy is best for him given all others play their equilibrium strategies.

Note: dominant strategy equilibria ⇒ Nash equilibria (but not vice versa).

Why study game theory:

Helps in:

- Agent design: design agent that reason strategically and perform optimally.
- Mechanism (game) design: assuming you know strategy of a player, MD tries to construct the rules of the game such that a certain property is obtained. For instance auction algorithms, where you try to create a competitive environment. Also used for routing, traffic conjunction etc.

Alternating move games:

These are games where the players' moves are sequential (player 1 moves, then player 2 and so on).

A game is considered *solved* if there is a strategy that taken guarantees to always win.

Game Trees

Games as search trees, where each player has a level in the tree; components:

- Initial state
- Successor function: (move, state) pairs
- Terminal test
- Utility function

For 2 players, they are called *max* and *min*.

- MAX: wants to maximize the final value of the final state
- MIN: minimize that value

Perfect play for deterministic games:

Assume that the opponent always makes the best possible move for them, then the solution is called the **minimax**: minimize the maximum possible loss I will make (from MAX's prespective; MIN wants to maximize the minimum possible game).

Theorem:

For every 2-player 0-sum game with finite strategies, there exists a value V and a mixed strategy per player such that:

- Given player 2's strategy, the best payoff possible for player 1 is V
- Given player 1's strategy, the best payoff possible for player 2 is $-V$

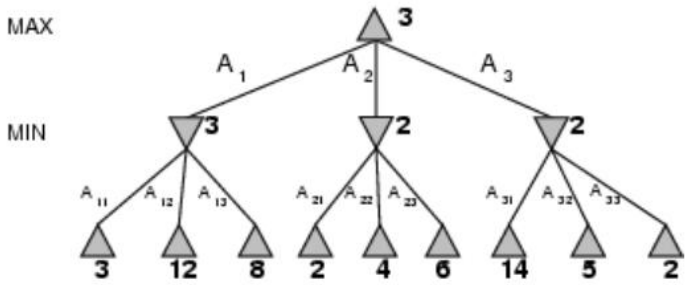
This is the same as mixed-strategy NE for 0-sum games.

It means that if each player knows the strategy of the other, there's always a mixed strategy for each player such that the payoff is symmetric.

Minimax value of a node:

Why in the game tree we want the MAX and MIN in different layers? In order to employ the minimax algorithm.

The minimax value of the node is the utility of MAX for reaching that state (MAX is always the root).



Think about the state of the world on MIN's rightmost node. He chooses, by looking forward, the rightmost node with the lowest value – 3. MIN is always going to choose that one. MAX will correspondingly choose 3, as it is highest among 3, 2 and 2. Minimax is recursively defined:

$minimax(x) =$

- If x is in MAX layer, it's the maximum of all $minimax(y)$ where y is a child of x
- If x is in MAX layer, it's the minimum of all $minimax(y)$ where y is a child of x
- If x is terminal, return $utility(x)$

Minimax algorithm:

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v
---------------------------------------------------------
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v
---------------------------------------------------------
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Properties:

- Complete: the tree is finite
- Optimal: yes, against an optimal opponent
- Time complexity: $O(b^m)$ – regular search
- Space: $O(bm)$ – depth first exploration

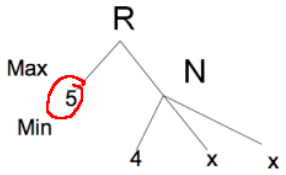
In chess $b \cong 35, m \cong 100$ for reasonable games, so exact solution completely infeasible

Alpha-Beta pruning:

Pruning out branches from the tree we don't need to search. It guarantees to return the same result as minimax but more efficient.

- α: minimum score of MAX player ($-\infty$)
- β: maximum score of MIN player (∞)
- $\beta < \alpha \Rightarrow$ no need to explore that branch any further

Example:



Here we know that once that 5 is found, $R \geq 5$ since R is a MAX node. We know that $N \leq 4$ therefore we know that R will always choose 5 anyway, so no need to search the two x nodes.

In this example: $\alpha = 5$ and $\beta = 4$ (the upper bound of the value of the solution for the MIN at that node).