2. Players and adversaries

III. Game Theory for Non-Cooperative Games

Giacomo Bergami

Newcaslte University

Objectives

- Modelling competing strategies as minimization vs. maximization of payoffs.
- Representing the combination of competing strategies as either payoff matrices or game trees.
- Determining rational strategies with MinMax and α-β pruning.
- Randomizing rational strategies and determining game difficulties with probabilities.

Part I

Game Theory for Non-Cooperative Games (1/2)

- Game Theory is the board of applied mathematics modelling players making interdependent decisions.
- The interdependencies are a consequence of both agents living on the same world and trying to achieve the same (or opposite) goal.
- Consequently, agents might try to anticipate each other's moves.
- This kind of theory is broadly adopted in economics and in biology.

Game Theory for Non-Cooperative Games (2/2)

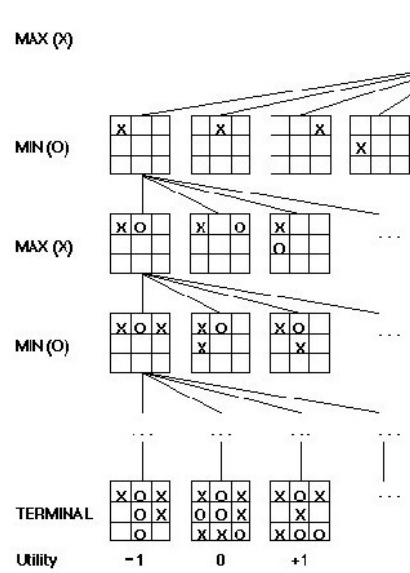
Each final state of the game has an *utility* value, or payoff:

- one player attempts a strategy maximizing the payoff (max),
- while the other tries to minimize it (min).

E.g., in *Rock-Paper-Scissors*, MAX wins when the resulting payoff is positive, while MIN wins when the resulting payoff is negative. If the payoff is zero, then we have a tie:

		MIN		
		Rock	Paper	Scissors
MAX	Rock	0	-1	1
	Paper	1	0	-1
	Scissors	-1	1	0

Modeling Multiple Turns in Tic-Tac-Toe



- f(n) = +1 if the position is a win for X.
- f(n) = -1 if the position is a win for O.
- f(n) = 0 if the position is a draw.

Guards and Thieves (1/2)

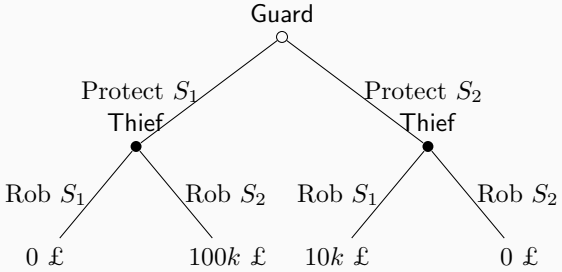
- Two locations have one safe each: №1 contains $10k$ £ and №2 contains $100k$ £.
- A guard is hired to protect both, but he can directly supervise only one place at a time.
- A thief must decide one of the two places.
- Nevertheless, they both must decide in advance, without knowing what the other party will do, which safe to rob and which safe to protect.

With respect to game theory:

- The thief is max.
- The robber is min.

Guards and Thieves (2/2)

		Guard: S_1	Guard: S_2
Thief	S_1	0 £	$10k$ £
	S_2	$100k$ £	0 £



Payoff Matrix (*left*):

- It describes two possible set of actions, one for each player. If we have more than two players, we will have **tensors**!
- This is the most known representation, but cannot correctly model only one-shot decisions for each of the players.

Game Tree (*right*):

- Game trees model each decision that could be associated to the player.
- Unlike Relational Learning, the reward is given only at the end of the process.

Game Theory for Game Balance

Randomization:

- If the human player is the robber and the NPC is the guard, the game shortly becomes uninteresting, as the player will easily know to go to place №1 and try to steal something.
- We can try to make the guard decision random: he will try to protect safe №1 with probability p and safe №2 with probability $1 - p$.

Difficulty:

- We could try to find p values making the choice non-deterministic ($p \neq 0, 1$).
- As we observed in Lesson `2.I`, players' expertise can be modelled with different probability values.
- We could set different difficulty levels as follows:
  1. Infer the strategy that makes the guard indifferent to what the thief does as the *average level* for the game.
  2. The easy level will make the guard take more frequently the most favourable strategy for the thief,
  3. and vice versa for the hard level.

MinMax algorithm for Deterministic Games

The overall algorithm can be summarized as a *post-visit* of the game-tree, where gc is the current Game Configuration after a given sequence of $\mathrm{nextMoves}$:

$$\textsc{MinMax}(gc) = \begin{cases} gc.\texttt{global\_utility} & \text{game ends at } gc \\ \min_{gc' \in \mathrm{nextMove}(gc)} \textsc{MinMax}(gc') & \text{turn in } gc \text{ is } \textsc{Min} \\ \max_{gc' \in \mathrm{nextMove}(gc)} \textsc{MinMax}(gc') & \text{turn in } gc \text{ is } \textsc{Max} \end{cases}$$

For our use case, this algorithm can be interpreted as follows:

1. The thief will always prefer the scenario allowing him to rob and not get arrested.
2. As a result, the guard wants to minimize the total loss, and therefore decides to supervise the second location.

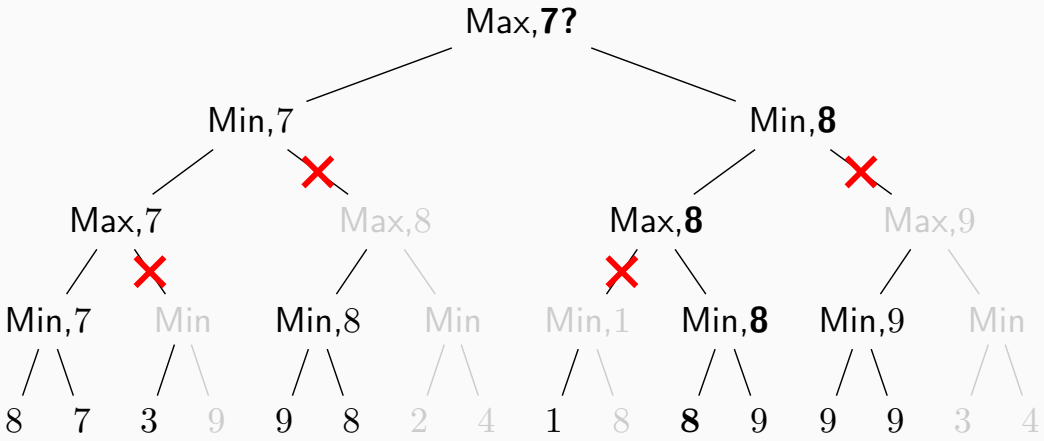
Given a branching factor of b and a maximum depth of m, the computational complexity for the exact solution is $O(b^m)$ and requires space $O(bm)$.

α-β pruning (1/2)

Given that the MinMax algorithm is an optimization problem, we can use a *Branch&Bound* optimal heuristic for reducing the amount of visited game states (*search space*). By assuming that Min and Max alternate their moves:

1. α and β are respectively the Max's and Min's bounds:
   - Initially, $\alpha = -\infty$ and $\beta = +\infty$.
   - α (and β) is the best payoff achievable for Max (and Min) in the current game state.
2. If a Max node picks a maximum value v from the j-th Min child, all the remaining Min children proposing a value $\nu < v$ are going to be pruned.
3. Dually, if a Min node picks a minimum value v from the j-th Max child, all the remaining Max children proposing a value $\nu > v$ are going to be pruned.

α-β pruning (1/2)



In practical games, we can achieve a computational complexity near to the optimal case scenario's computational complexity, $O(\sqrt{b^d})$.

IBM Deep Blue (1997)

Before the Deep-NN, this machine managed to beat the world chess player, Kasparov, by applying the following techniques:

1. MinMax
2. α-β pruning
3. More pruning strategies
4. Uneven tree expansion (trade-off with confidence in winning)
5. Parallel Computing
6. Well known Opening Moves
7. Well known Final Moves

Part II

Guards and Thieves: Average Level

The thief will get:

- $100k$ £ with probability p, and
- $10k$ £ with probability $1 - p$

The thief will get the following average gains:

- If he tries to rob S_1, 10,000$(1 - p)$ £
- If he tries to rob S_2, 100,000p £

The guard will indifferently choose one of the two safes to protects if the average amount stolen is the same:

$$10^4(1 - p) = 10^5 p$$

On the average level, the guard should protect the first safe with probability $p = {}^1/_{11}$, and protect the second with probability $1 - p = {}^{10}/_{11}$

- On average, the thief will gain $9090.\overline{90}$ £

Real Use Case: Backgammon (1/2)

We can oversimplify the games' rules as follows:

- Each player has a set of 15 checkers.
- The aim of the game is to move all the checkers to the player's *home board*. Then, the player has to move the checkers out of the whole board.
- A player rolls two dices, and the resulting value will tell how to move along the points (*long triangles*).
- A checker cannot move to a point containing at least 1 opponent's checker.
- Single checkers might be pushed by the opponent towards the beginning of the game.

Real Use Case: Backgammon (2/2)

This game mixes chance (actions performed by the player) with strategy (outcome of the dice roll):

- *Force Majeure*: The dice rolls restraint the set of the possible moves that a player can perform.
- *Free Will*: The player could choose which checker to move within the borad.

ExpectMinMax algorithm

When random events are introduced within the game, the best that we can do is to do an estimation of the expected value:

$$\textsc{ExpectMinMax}(\mathsf{gc}) = \begin{cases} gc.\texttt{global\_utility} & \text{game ends at } gc \\ \min_{gc' \in \text{next}(gc)} \textsc{ExpectMinMax}(gc') & \text{turn in } gc \text{ is } \textsc{Min} \\ \max_{gc' \in \text{next}(gc)} \textsc{ExpectMinMax}(gc') & \text{turn in } gc \text{ is } \textsc{Max} \\ \sum_{gc' \in \text{next}(gc)} \textsc{ExpectMinMax}(gc') \cdot p_{gc \to gc'} & gc \text{ generates } \textsc{Random Events} \end{cases}$$

under the assumption that $\sum_{y \in \mathsf{out}(x)} p_{xy} = 1$ for each x.

We will not discuss here pruning mechanisms, as they are advanced research topics! (E.g., Monte Carlo Sampling is often used)

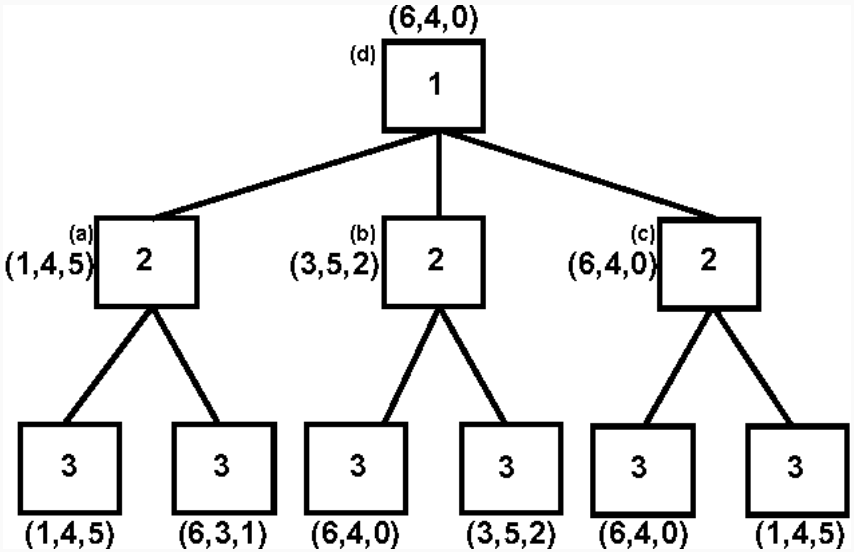
Max^n, i.e., Multi-Player Games (1/2)

When we have more than 2 players, any reduction to the MinMax algorithm cannot guarantee the solution's optimality. In such scenarios, we use a simpler solution:

- for each final configuration of the game (*leaf node*), we report the utility score associated to **each** i-th player of n in a n-dimensional score vector $\vec{v} = (v_1, \ldots, v_n)$.
- Each non-leaf node associated to the turn of the i-th player picks the child vector maximising the i-th component, and sets it as its utility value:

$$\textsc{Max}^n(gc) = \begin{cases} (gc.v_1,\ \ldots,\ gc.v_n) & \text{game ends at } gc \\ \arg\max_{gc' \in \text{nextMove}(gc)} \left(\textsc{Max}^n(gc')\right)_i & i\text{-th player turn} \end{cases}$$

Max^n, i.e., Multi-Player Games (2/2)



Max^n Pruning

- We are not going to discuss how to optimize such algorithm in full detail, as this requires that the total sum of the game's resources is constant in time.

A possible formal notation for the first statement is:

$$\forall game(n).\exists c.\forall gc \in game(n).\sum_{i=1}^{n} gc.v_i = c$$

Max^n Pruning

- We are not going to discuss how to optimize such algorithm in full detail, as this requires that the total sum of the game's resources is constant in time.
- This can be possibly done by setting the World as a "passive player", which performs no action but, still, the opponents' actions affect its utility value whenever one of the players varies its utility score within the game.

A possible formal notation for the first statement is:

$$\forall game(n).\exists c.\forall gc \in game(n).\sum_{i=1}^{n} gc.v_i = c$$