Poker Bot

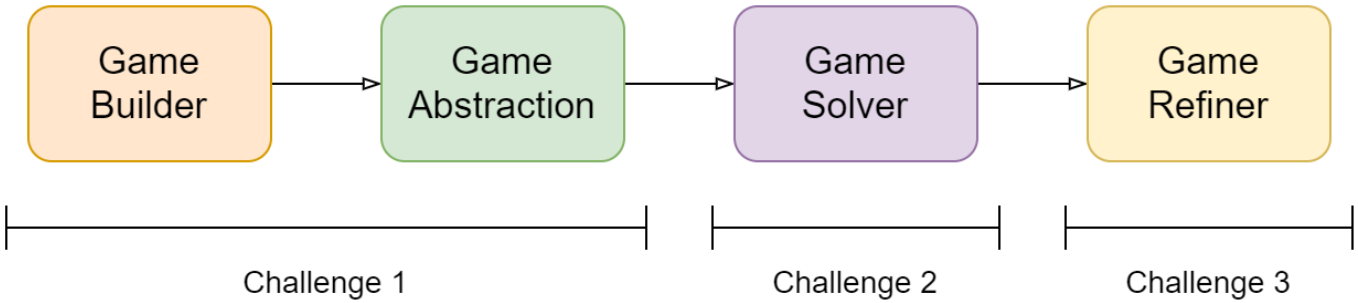
Project for the Economics and Computation Class
PoliMi 2019-2020

Corda Francesco - 920212
Damato Davide - 920616
Fiorentino Alessio - 920488
Franchin Luciano - 921093

July 10, 2020

Framework

- The following diagram shows the general framework of our project.
- The architecture is organized in four high-level modules:



Game Builder

- This module is responsible for the construction of the game tree in extensive form
- The Game Builder can parse files that comply with the format established by the efg_lib

Game Builder

- The parsing is an iterative process that, for each line of the file:
  1. Distinguishes the different classes using a set of regular expressions.
  2. Creates a node instance and attach it to the tree.
  3. The resulting tree is a recursive data structure of nodes.

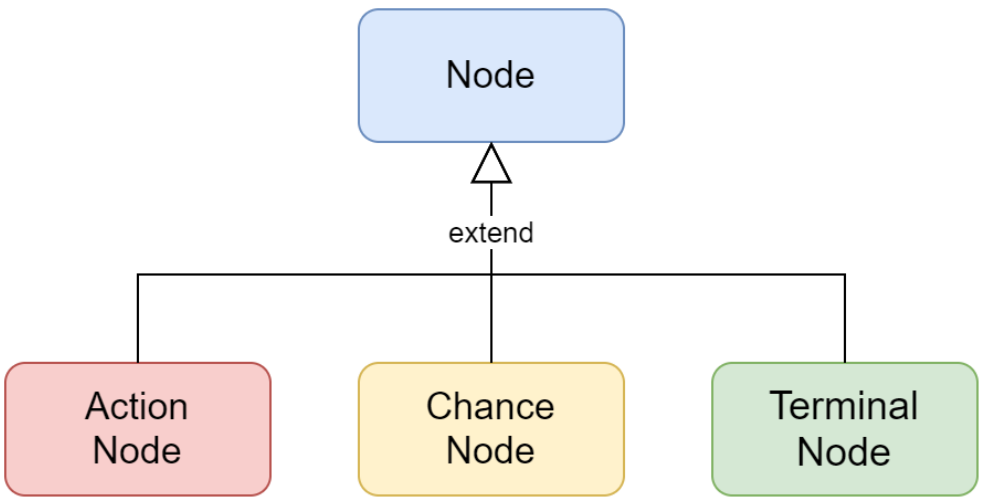


Figure: Model of the game nodes.

Game Builder

- The game builder also parses the information sets, grouping nodes that belong to the same set together.
- The information sets are organized in lists of nodes, where each node maintains a reference to the others in the group.

Figure: Model of the information set structure.

Game Abstraction

- To perform the game abstraction, we adopted a top-level approach that visits recursively the tree starting from the root.
- This choice allowed us to avoid occurrences of imperfect recall during the process.
- We also designed the algorithm to abstract one player at a time, to be able to use different parameters for each player.
- The abstraction procedure, for each level:
  1. Identifies information sets that are eligible to be unified, based on the sequence of moves that preceded them
  2. Produces a clustering

Game Abstraction

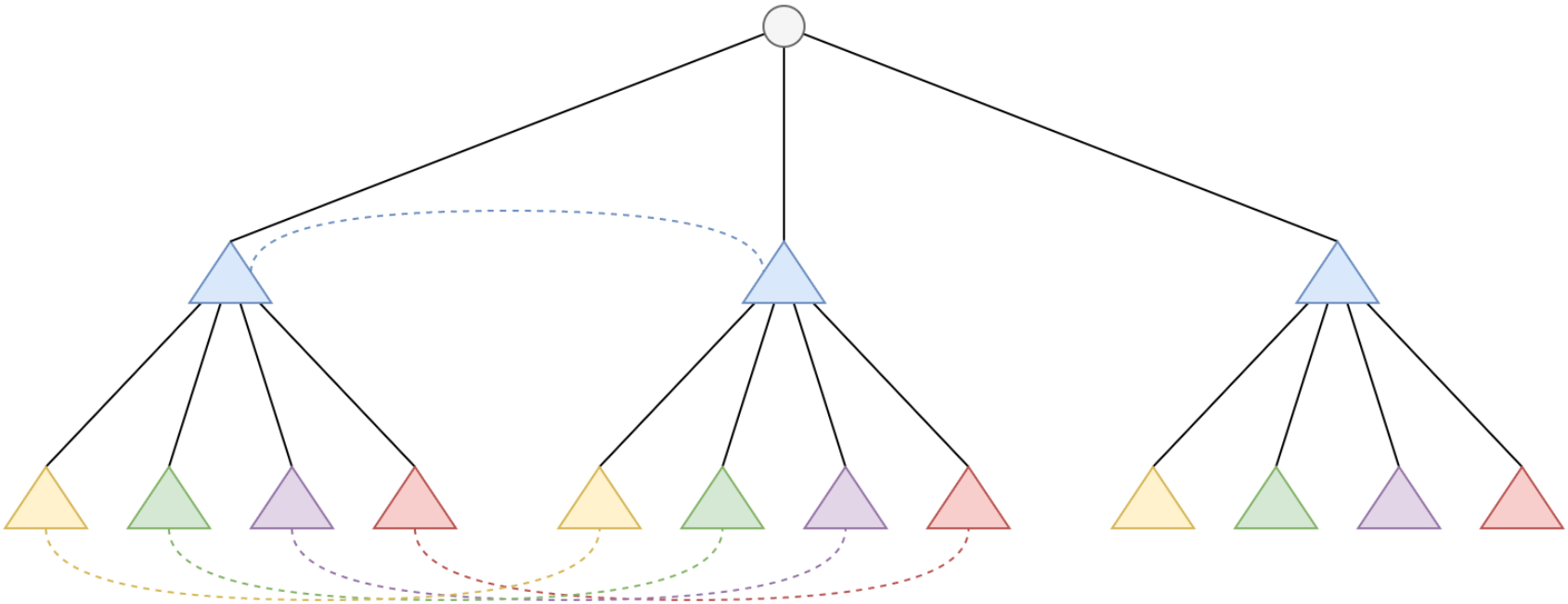


Figure: Visualization of a top-down abstraction.

Game Abstraction

- The clustering task is responsible of grouping information sets that satisfy the following properties:
  - Same player
  - Same level in the tree
  - Same set of actions / sequence of moves that preceded them
- Each node of an information set is associated with a point in a d-dimensional space, where d is the number of terminal nodes that derive from the node. The coordinates in the dimensions are given by the payoffs of the terminal nodes.

Game Abstraction

- To perform the clustering task, we adopted the K-means algorithm.
- K-means is one of the most popular clustering algorithm for its simplicity and good performance.
- As is well known, k-means is not guaranteed to converge to the optimal solution and is strongly dependent on the initialization. To cope with this limitations we tried different random seeds and evaluated the results.
- This portion of the project was the only one that we didn't implement, choosing instead to use the implementation provided by the scikit-learn library.

Game Solver

- After the abstraction, we developed the Game Solver module.
- This module takes as inputs a game tree and a game abstraction to produce a set strategies for the game.
- After some research, we decided to adopt a Monte Carlo CFR approach, that guarantees good convergence properties and a faster performances than the standard version of Counterfactual Regret.
- Also, after some comparisons, we opted for the CFR+ version of the algorithm, because it seemed to provide more stable results.

Game Solver

- Monte Carlo CFR can be implemented in different ways, depending on the technique used to perform the sampling.
- We opted for the External Sampling approach, that draws the samples only for the nodes belonging to the opponent and to the nature.
- Our solution is almost entirely based on the pseudo code developed by Marc Lanctot in his PhD thesis, with the following addition provided by Professor Gatti slides:
  - Each payoff in the terminal nodes is weighted by the inverse of the probability of reaching it, starting from the tree root.

Game Solver

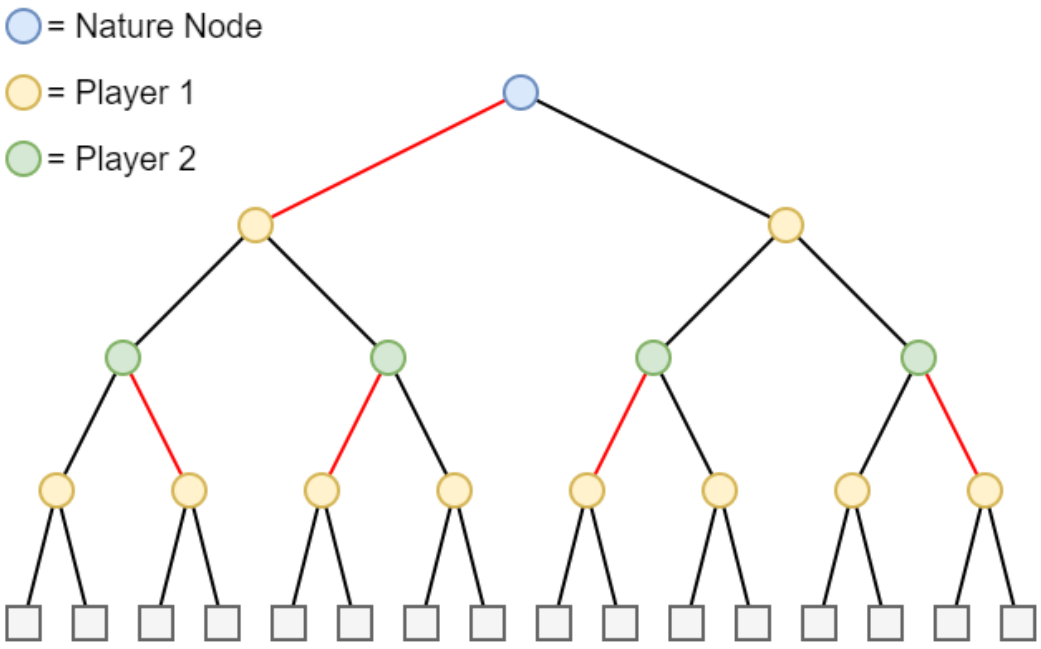


Figure: Monte Carlo CFR with External Sampling, the sampling (edges in red) is done only for the chance and opponent nodes.

Game Refiner

- The objective of this module is the resolution of an imperfect information game.
- Starting from a blueprint strategy produced in the previous step, the Game Refiner improves the strategy at run time by solving an auxiliary tree.
- To solve the task we adopted again a top down approach, since the optimal strategy at a given level depends on the results from the previous level.

Game Refiner

- The algorithm traverses the game tree starting from the root.
- The main steps for each level are:
  1. Identification of the sub-games in the level.
  2. For each sub-game, compression of the sub-tree generated by it. This operation consists in the removal of all the information sets in the sub-tree belonging to the player that we are trying to refine.
  3. Creation of an auxiliary tree that associates the sub-games with the probabilities of reaching them from the root node.
  4. Solution of the auxiliary tree, using the Game Solver.
  5. Remapping of the updated blueprint strategy into the original tree.

Game Refiner

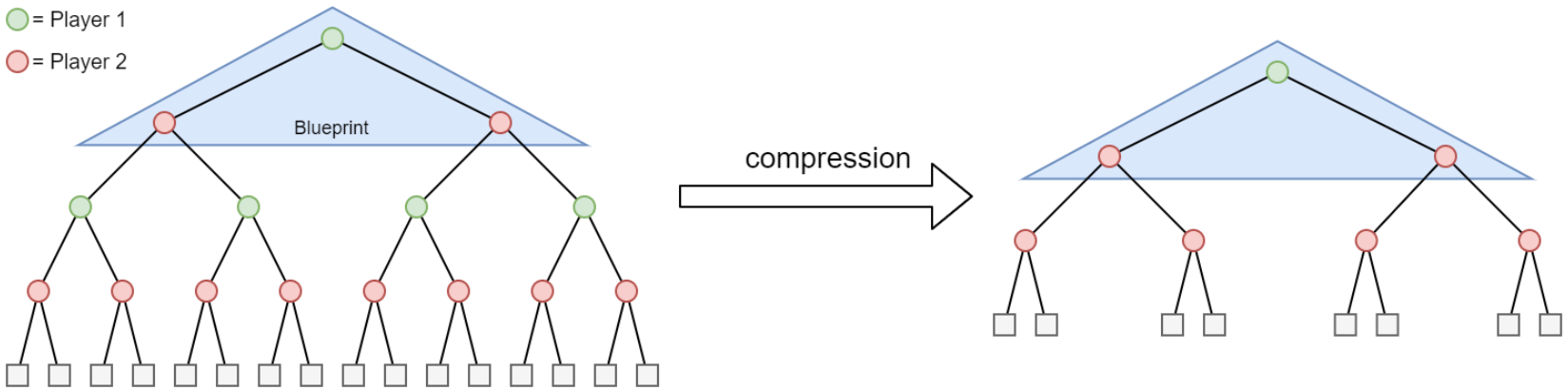


Figure: Compression of the auxiliary tree. This step is followed by the solution of the compressed tree and the remapping of the strategy into the game tree.

Performances

- To assess the performances of our models, we observed the evolution of the cumulative regret during training.
- To evaluate the quality of an abstraction, we compare its performance with a baseline (represented by the non-abstracted game).
- The following plots shows the regret accumulated by an abstraction, compared to a baseline trained with the same parameters.

Performances

- Insert regret plots...

Game Simulator

- We built a game simulator to have a closer look at the choices made by the strategies we produced.
- Given a game tree and a strategy for each player, the Game Simulator plays an arbitrary number of Poker games and outputs:
  1. The expected return for each player, computed as the empirical means of the expected rewards collected in each game.
  2. The list of moves chosen by the players in each turn, to let us observe the behaviors we were obtaining.

References

- GitHub repository: `https://github.com/francescocorda/pokerbot`
- Slides and videos provided from the Economics and Computation classes
- Tuomas Sandholm, (2015). Abstraction for solving large incomplete-information games.
- Marc Lanctot, (2012). Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision-Making in Large Extensive Form Games.