Department of Computer Sciences
University of Salzburg

PS Natural Computation
SS 15/16

Evolution of a Poker Player Using Genetic Programming

July 13, 2016

Project Members:

Thomas Samy Dafir, 1331483, tdafir@cosy.sbg.ac.at
Laurentiu Vlad, 1423336, lvlad@cosy.sbg.ac.at
Dominik Baumgartner, 0920177, dbaumgart@cosy.sbg.ac.at
Sebastian Strumegger, 1420277, sstrumegg@cosy.sbg.ac.at

Academic Supervisor:

Helmut MAYER
helmut@cosy.sbg.ac.at

Correspondence to:

Universität Salzburg
Fachbereich Computerwissenschaften
Jakob–Haringer–Straße 2
A–5020 Salzburg
Austria

Abstract

The goal of this project is to evolve a GP poker player by means of genetic programming techniques. The GP Players evolve over generations through selection, mutation and cross mutation of their chromosome tree.
A good computer player has to emerge after a sufficient amount of generations have been played in a "heads up" texas-hold-em no limit poker game. Good players are determined by their fitness value which is determined through assessing a player's success after each poker round.
The focus of our work lies in the chromosome tree. Although the current API provides some basics functions for that tree, it is likely that they are not sufficient for the evolution of a good player. The second focus lies on the poker game itself. We have to determine what kind and how much information a player needs in order to evolve properly without interfering with its evolution. This means, that we don't want to "tell" the player what the next best action is, but rather let evolution decide. Additionally we will do some fine-tuning of evolutionary-parameters like population size or number of generations to achieve the best possible result.

1 Genetic Programming

As mentioned, a GP Player is represented by a so-called chromosome tree. Or more exactly, the poker player's strategy is represented by this tree.
This means, that each player's tree has a root node and more or less child nodes, where nodes with children are called functions and leave nodes are called terminals. From the very beginning there is a fixed set of functions and terminals available. At first, a population of GP Players with random nodes are created which play a predetermined number of games. In each generation there is a chance for every player that it's tree is changed by mutation or crossover.

1.1 Chromosome Tree

1.1.1 Functions

Functions play the most important role in a GP Player's strategy, since functions are the decision-makers.
They can have one or more child nodes and always are of a specific return type. Further they need to have a parent node (except the root function). Child nodes can either be terminals or functions, where it is also very important that the child nodes return the very type, that the function expects.
As an example we could define a lower-than function that expects floating point values from both children and returns a boolean value to it's parent node.

1.1.2 Terminals

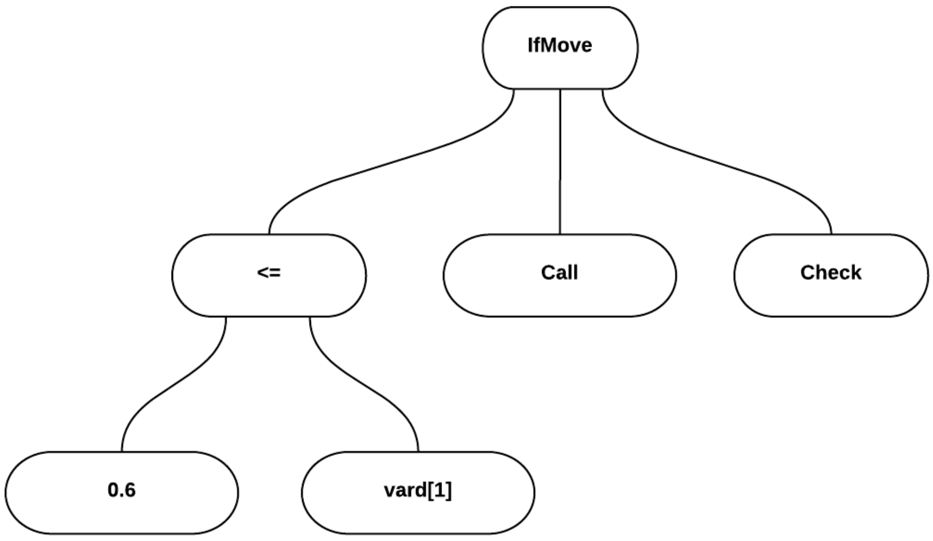
Terminals are the leave nodes in a player's tree. They cannot have child nodes but return a certain type themselves.
This does not necessarily mean that a terminal's value has to be static. They can have dynamic values like "pot size" or "higher hand card".

1.1.3 Root

The root node it a special function, since it does not have any parent nodes.
For a proper evaluation of the player's tree it needs to be rooted, so we can start making further decisions at one point. For GPoker we need to have a root node that allways returns a poker move which is returned to the game.

A GPlayer, for example, uses the root node "IfMove" which has three children:
The very left child is the if-branch, that needs to return a boolean type.
The middle child is the then-branch and returns a move.
And at last, the right child represents the else-branch, which also returns a move.

1.1.4 Example



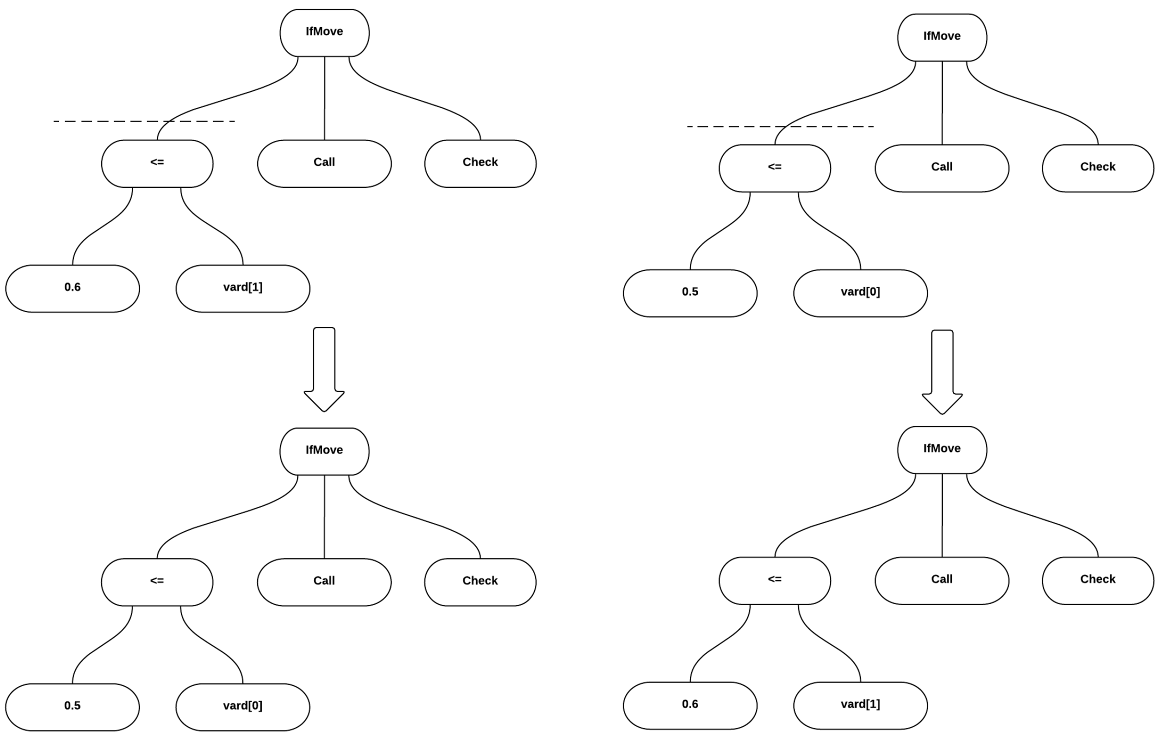
This very simple tree represents the following strategy:
If 0.6 (Two to Ace normalized in 0 to 1) is less than or equal to the player's lower hand card, make a call.
Else, check the game. Note that checking can also mean folding, in case there has been a raise.

1.2 Evolution

There are two fundamentally different ways for evolution to change a tree, namely crossover and mutation.

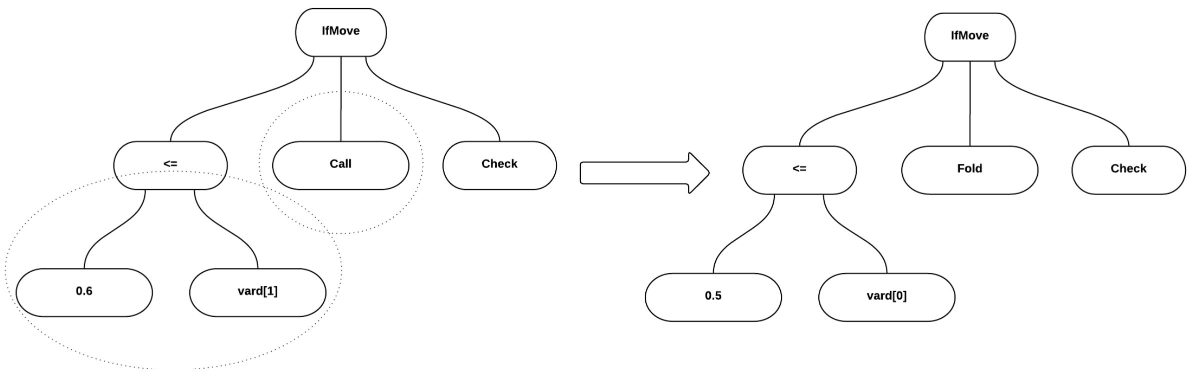
1.2.1 Crossover

Crossover takes two individual players and swaps subtrees in a random point. Of course both subtrees need to have the same return type. This method is more common, since good players that were not sorted out by evolution usually have good subtrees. So the the well-playing trees are not completely destroyed by the process of evolution, but there is much room for new combinations.



1.2.2 Mutation

In contrast to Crossover, mutation does not even need two individual trees for change. Either a single node is chosen randomly and changed to some other random node that returns the same type. Or even a randomly chosen subtree can be changed. Mutation has to be used carefully, since random changes can change a very good player to be useless.



2 Links

- Project Page: `https://student.cosy.sbg.ac.at/~tdafir/nc/`
- PS Page: `http://www.cosy.sbg.ac.at/~helmut/Teaching/NaturalComputation/proseminar.html`

References

[1] E. Ebensperger, "Implementation of a general poker framework validated by evolution of computer player strategies," Sept. 2014.