Cryptocurrency Trading Algorithm White Paper

Patrick Nercessian

Evolutionary computation methods should be able to evolve an algorithm which can consistently make profit by buying cryptocurrencies at low prices and selling them at higher prices. Genetic programming (GP), or something similar, seems like a suitable option to create a profitable algorithm [1]. After gathering pricing data and technical indicators [2][3], the program should call a function *decision()* which takes the pricing data and technical indicators as parameters. This function will be created through GP evolution.

Specifically, *decision()* will be created from a tree structure with a Terminal set T and a Function set F. Depending on a probability variable, new offspring will be created from either mutation or crossover (the probability of mutation should be very low to avoid massive trees). Mutation will involve replacing a random subtree (leaf nodes should have a higher chance of being selected) with a randomly generated subtree. The root node of the new subtree will be probabilistically chosen from either T or F. If a function is chosen from F, the algorithm will recursively generate a new subtree for each argument needed. Crossover will involve exchanging a random subtree from each parent.

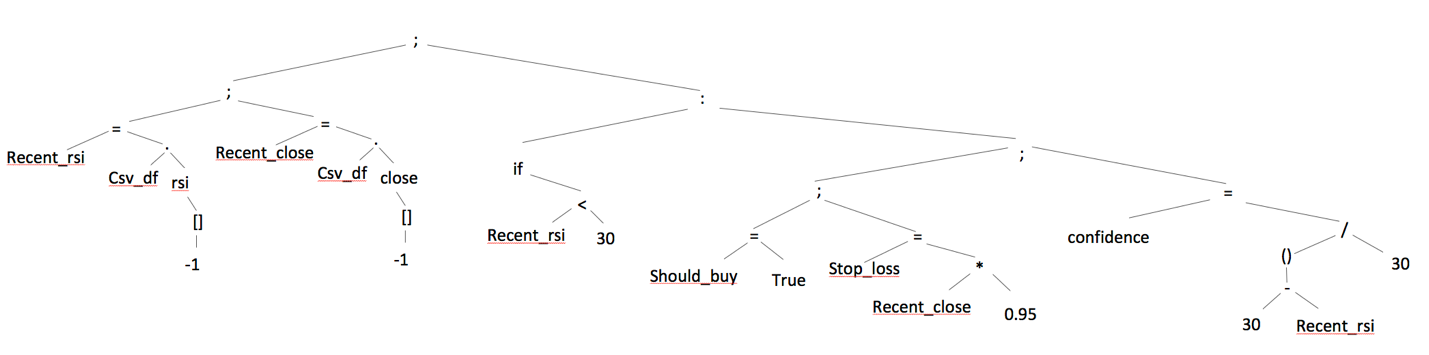
Other than a few necessary manual constraints, the algorithm should be free to explore any syntactically correct tree function. These constraints will include ensuring that a variable signifying Buy, Hold, or Sell, a float variable representing a stop loss value, and a float variable representing a confidence percentage (which will inform the manually coded section how much to invest) are declared and returned by the function. If the first variable is not Buy, the other two variables are ignored. **Further research needs to be conducted to determine suitable Terminal and Function sets T and F. Should also re-research dynamic search spaces for evolutionary algorithms from class.**

To determine the fitness of an individual, it needs to be tested on historical data with a predefined starting balance. First, a random historical timestamp will be generated which will act as the present. Next, the *decision()* function will receive pricing data and technical indicators up until that point, and whatever is returned by the function will be the decision made by the individual. If a Buy decision is returned, the algorithm will continue to run on each subsequent timestamp until it either closes the position or is forced to sell after a predetermined length of time (e.g. 24 hours). Now the process will repeat for a new randomly generated timestamp until it has performed at least *x* (e.g. 3) buys. If a predetermined number of timestamps are evaluated and 3 buys are not executed, a fitness of 0 is returned.

Example Algorithm:

def decision\_example1(csv\_df):  
 should\_buy = True  
 stop\_loss = 0  
 confidence = 1.0  
  
 # This should be what results from the tree  
 recent\_rsi = csv\_df.rsi[-1]  
 recent\_close = csv\_df.close[-1]  
 if recent\_rsi < 30:  
 should\_buy = True  
 stop\_loss = recent\_close \* 0.95  
 confidence = (30 - recent\_rsi) / 30  
  
 return should\_buy, stop\_loss, confidence

Correlated Tree Structure



* Function Set F
  + Basic operands: +, -, \*, /
  + :
    - Left child is conditional or loop
      * if, while
        + Right child is Boolean:

==, !=, >, >=, <, <=, is, is not (two children)

‘and’ or ‘or’ (two Boolean children)

* + - * for
        + Only child is ‘in’

Left child is temp variable

Right child is list/column

* + - Right child is enclosed block
  + Assignment: =
    - Left child is variable
    - Right child is expression

[1] <https://algotrading101.com/learn/binance-python-api-guide/>

[2] https://tradingbrowser.com/best-cryptocurrency-indicator/

[3] <http://jonathankinlay.com/2018/09/developing-trading-strategies-with-genetic-programming/>

[4] https://zhanggw.wordpress.com/2009/11/08/a-simple-genetic-programming-in-python-4/