

Application of Bellman-Ford Algorithm to Find Arbitrage Condition in Forex Trading

Taufiq Husada Daryanto 13518058
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
taufiqhd@students.itb.ac.id
13518058@std.stei.itb.ac.id

Abstract—In this paper we discuss the use of Bellman-ford algorithm on finding negative cycle to find an arbitrage condition in forex trading. Also, in this paper, we will discuss about ideas to improve the efficiency of graph implementation on finding arbitrage situation. Lastly, we also discuss about some advantages and disadvantages on using Bellman-ford algorithm to find arbitrage condition in forex trading

Keywords—Arbitrage, Bellman-Ford Algorithm, Forex Trading, Graph.

I. INTRODUCTION

In our life, we usually face several problems. When we deal with problems, we usually use our intuition mainly, with a little amount of calculations in our head. Sometimes, with just an intuition it brings out good solution while it does not take us long time to think about the calculations. But on the other hand, doing less calculations also can lead us to wrong solution that maybe can make the problem even worse. Maybe it is true that calculations in real life problem solving will make things complicated, but it will give us a clear and correct decision.

One example topic that we can use in our daily life is graph theory. By modelling problems that we faced into a graph, and solve it using some methods in graph theory, it can help us to make an important decision. For example, finding shortest distance, finding minimum cost, and so on.

We can apply graph theory in forex trading. I know that some people say forex trading is *haram* because it is like speculation to gain profit similar with gambling. But in this paper, what I want to highlight is not the forex trading itself, but how we can apply methods in graph for real life condition.

II. THEORIES

A. Graph Introduction

A graph is a structure that is defined by two components, which are node and edge [1]. Node can represent some information. Edge is a connection between two nodes, and it also can represent something that give a meaning to relation between two connected nodes.

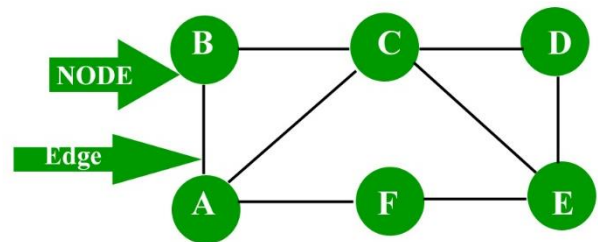


Figure 1. Example of graph, picture from <https://www.geeksforgeeks.org/mathematics-graph-theory-basics-set-1/>

There are some classifications of graph. From edge characteristic, graph can be classified into two types, which are weighted graph and unweighted graph. Weighted graph is a graph that for every edge, there is a weight on it.

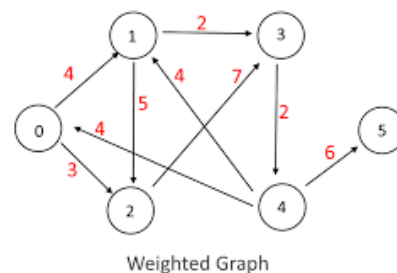


Figure 2. Example of weighted graph, picture from algorithms.tutorialhorizon.com

Those weights can represent something for example cost, distance, etc. On the other hand, unweighted graph has no weight in every edge, so edges only represent the connectivity between nodes.

From the directivity of edges, graph can be classified into two types, which are direct graph and undirect graph. Direct graph is a graph that for every edge it has direction.

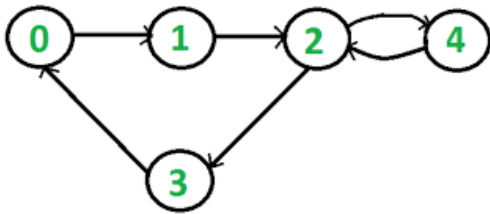


Figure 3. Example of directed graph, picture from geeksforgeeks.com

Those direction can represent the path between nodes. The opposite, undirect graph has no direction in the edges.

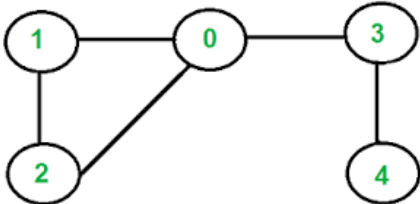


Figure 4. Example of undirected graph, picture from geeksforgeeks.com

B. Graph Implementation

There are some representations of graph implementation. The two most common representations are adjacency matrix and adjacency list.

Adjacency matrix is a way to represent the connectivity between nodes as a element in matrix, with a "1" or "0" in position (v_i, v_j) according to their connectivity [2].

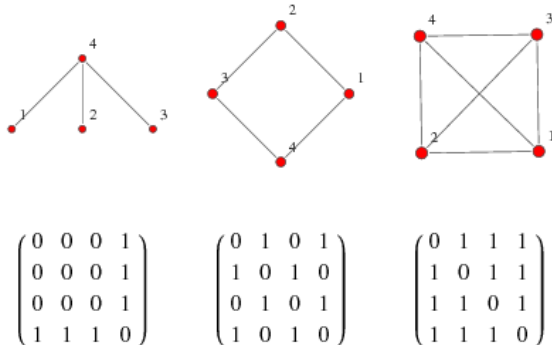


Figure 5. Example of adjacency matrix, picture from <http://mathworld.wolfram.com/AdjacencyMatrix.html>

In weighted graph, those "1" or "0" can be replaced by the weight of edge itself if two nodes are connected or with infinity if there is no edge connect the two specific nodes.

Adjacency list is array of list, that every node become array elements, and node that connected with will be the element of the list.

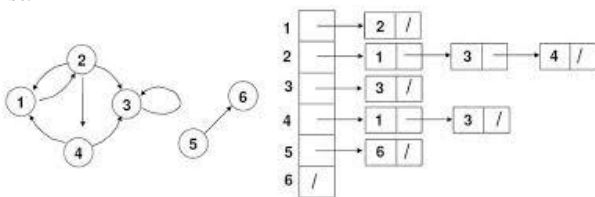


Figure 6. Example of adjacency list, picture from researchgate.net

In weighted graph, list elements can be represented as pair of nodes and its edge weight.

From those two implementations there are several benefits and disbenefits. In term of memory efficiency, adjacent list is better than adjacency matrix because in adjacency list, program will only create element on list from node if that element is connected to that node. On the other hand, adjacency matrix will create all space between nodes, because that is how matrix is represented. In term of time efficiency on accessing the connectivity or weight between nodes, adjacency matrix is better than adjacency list. To access the connectivity between nodes in adjacency matrix, program can just call the element on coordinate (v_i, v_j) , so it only gives $O(1)$ time complexity. On the contrary, in the adjacency list, program must traverse the list to find specific node that connected to, so it gives $O(m)$ time complexity, where "m" is number of edges.

C. Negative Cycle in Graph and Bellman-Ford Algorithm

Cycle in graph is a way that from a node, there will be edges that will make a path to come back to that node again. Negative cycle is a cycle that sum of all edges in that cycle is negative.

Bellman ford algorithm is a way to find shortest path from source node to all nodes in the given graph, but the graph may contain negative edges [3]. The idea of this algorithm is to relax all edges $V-1$ times, where V is number of nodes. The idea why relaxing $V-1$ times is because the shortest path from a node to the other node proved that it does not involve more than $V-1$ edges. Here is the pseudocode to find shortest path from source to other nodes.

Algoritma 65 Implementasi algoritma Bellman-Ford.

```

1: function BELLMAN-FORD(s)
2:   dist ← new integer[V + 1]

3:   FILLARRAY(dist, ∞)
4:   dist[s] ← 0
5:   for i ← 1, V - 1 do
6:     for ⟨u, v⟩ ∈ edgeList do
7:       if dist[v] > dist[u] + w[u][v] then
8:         dist[v] = dist[u] + w[u][v]
9:       end if
10:    end for
11:  end for

12:  return dist
13: end function

```

Figure 7. Pseudocode to find shortest path using Bellman-ford algorithm, picture from Pemrograman Kompetitif Dasar, - Aji & Gozali.

To find the negative cycles we can run one more relax, if there is the path become shorter then there is a negative cycle. Here is the pseudocode

```

1: function HASNEGATIVECYCLE(s)
2:   dist ← BELLMAN-FORD(s)

3:   for (u,v) ∈ edgeList do
4:     if dist[v] > dist[u] + w[u][v] then
5:       return true
6:     end if
7:   end for
8:   return false
9: end function

```

Figure 8. Pseudocode to find negative cycles using Bellman-ford algorithm, picture from Pemrograman Kompetitif Dasar, - Aji & Gozali.

The time complexity of this algorithm is $O(VE)$ which V is number of nodes and E is number of edges.

D. Forex and Arbitrage

Forex is an abbreviation from foreign currency exchange. Foreign exchange is the process of changing one currency into another currency for a variety of reasons, usually for commerce, trading, or tourism [4]. Forex trading is the process of getting profit from currency exchange. The principle of forex trading is simple that is getting profit from the difference between the buying and the selling price by making a buy transaction at a low price and a selling transaction at a high price [5].

Forex arbitrage is a trading strategy that seeks to exploit price discrepancy [6]. The method is simple that is finding if there is a way from one currency, traded one into another, then get back at that currency but with higher amounts as our profit. Here is an example of arbitrage situation in forex trading

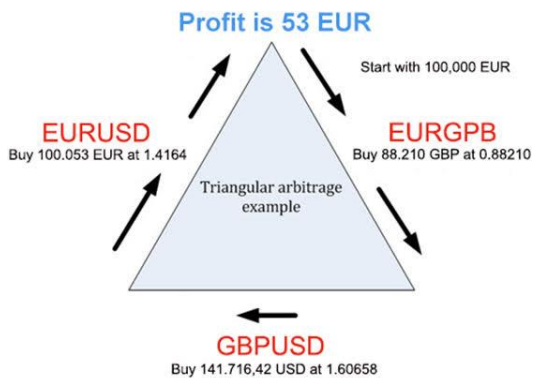


Figure 9. Example of arbitrage in forex, picture from alparsi.com

To exploit arbitrage, we have to do rapid execution, so we have to use automated program to find arbitrage condition and exploit it.

There are several methods to detect arbitrage, which are cycle detection with Bellman-Ford shortest path algorithm, the famous Black-Scholes option pricing formula, and statistical arbitrage algorithm [7]. The simplest method is to use Bellman-ford algorithm.

III. APPLICATION OF BELLMAN-FORD ALGORITHM ON DETECTING ARBITRAGE SITUATION

A. The methodology

First, we have to model the currency rates data as a graph. Currency as a node, and exchange rates as their edges. To detect arbitrage, for example if we go in cycle and get weighted edge path like $a \rightarrow b \rightarrow c \rightarrow d$, arbitrage is a condition when $a * b * c * d > 1$, which means that we have to find a cycle that product of their edges is greater than 1.

In order to use Bellman-ford algorithm, we have to design it not as a product but as a sum and we look for negative sum. Because we know that $\log(a * b) = \log(a) + \log(b)$, first we have to turn all edges as a negative log. For example, edges path $a \rightarrow b \rightarrow c \rightarrow d$ turns into $-\log(a) \rightarrow -\log(b) \rightarrow -\log(c) \rightarrow -\log(d)$. The total cost of this path is $-(\log(a) + \log(b) + \log(c) + \log(d)) = -\log(a * b * c * d)$. As if $-\log(x) < 0$ means that x is greater than 0, so that arbitrage condition is when the total cost path of negative log edges is less than zero.

So, here is the step for detecting arbitrage

1. Data preparation
2. Graph modelling and convert all the rates into negative log
3. Find a negative cycle using Bellman-ford algorithm

B. Data preparation and preprocessing

This is the example of the currency data

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.011
EUR	1.35	1	0.889	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.943	0.698	0.62	1	0.953
CAD	0.955	0.732	0.65	1.049	1

Figure 10. Currency rates sample data, picture from globalsoftware.com

Elements of row and column are exchange rates between currency in specific row into currency in specific column

C. Graph modelling

I will implement the graph using adjacency list model in C++. Here is the code

```

#include<bits/stdc++.h>
using namespace std;

std::vector<vector<pair<int,double>>> adj(5);
string currency [] = {"USD","EUR","GBP","CHF","CAD"};

int main(){
    // from USD
    adj[0].push_back({0,1});
    adj[0].push_back({1,0.741});
    adj[0].push_back({2,0.657});
    adj[0].push_back({3,1.061});
    adj[0].push_back({4,1.011});
}

```

```

// from EUR
adj[1].push_back({0,1.35});
adj[1].push_back({1,1});
adj[1].push_back({2,0.889});
adj[1].push_back({3,1.433});
adj[1].push_back({4,1.366});

//from GBP
adj[2].push_back({0,1.521});
adj[2].push_back({1,1.126});
adj[2].push_back({2,1});
adj[2].push_back({3,1.614});
adj[2].push_back({4,1.538});

//from CHF
adj[3].push_back({0,0.943});
adj[3].push_back({1,0.698});
adj[3].push_back({2,0.62});
adj[3].push_back({3,1});
adj[3].push_back({4,0.953});

//from CAD
adj[4].push_back({0,0.955});
adj[4].push_back({1,0.732});
adj[4].push_back({2,0.65});
adj[4].push_back({3,1.049});
adj[4].push_back({4,1});

```

Figure 11.A Implementation of Bellman ford algorithm on finding arbitrage condition, initial set up

I modelled the adjacency list using vector of vector of pair in C++. I modelled the nodes as number from zero to four and save the currency name in array of string with related index.

The next step is to convert all the edges into negative log, here is the procedure

```

void convertIntoNegativeLog(){
    for (int i = 0; i<5; ++i){
        for (int j = 0; j<5; ++j){
            adj[i][j].second = -log(adj[i][j].second);
        }
    }
}

```

Figure 11.B Implementation of Bellman ford algorithm on finding arbitrage condition, convert to negative log

After that call that procedure in main program after inserting nodes and edges, then check the resulting edges by printing them into terminal

```

convertIntoNegativeLog();
for (int i = 0; i<5; ++i){
    for (int j = 0; j<5; ++j){
        cout<<i<<" -> "<<adj[i][j].first<<" : "<<adj[i][j].second<<endl;
    }
}

```

Figure 11.C Implementation of Bellman ford algorithm on finding arbitrage condition, main program converting to negative log

The result is like this

```

0 -> 0 : -0
0 -> 1 : 0.299755
0 -> 2 : 0.420071
0 -> 3 : -0.0592119
0 -> 4 : -0.0109399
1 -> 0 : -0.300105
1 -> 1 : -0
1 -> 2 : 0.117658
1 -> 3 : -0.35977
1 -> 4 : -0.311887
2 -> 0 : -0.419368
2 -> 1 : -0.118672
2 -> 2 : -0
2 -> 3 : -0.478716
2 -> 4 : -0.430483
3 -> 0 : 0.058689
3 -> 1 : 0.359536
3 -> 2 : 0.478036
3 -> 3 : -0
3 -> 4 : 0.0481404
4 -> 0 : 0.0460439
4 -> 1 : 0.311975
4 -> 2 : 0.430783
4 -> 3 : -0.0478373
4 -> 4 : -0

```

Figure 11.D Implementation of Bellman ford algorithm on finding arbitrage condition, result after converting to negative log

D. Applying Bellman-ford algorithm

First initialize all cost as infinity, we use $1e9+7$ as infinity number. Take node 0 (USD) as a source node. Then we apply Bellman-ford algorithm by relaxing edges $V-1$ times (4 times, as number of nodes is 5), and calculate the minimum cost.

```

void findNegativeCycle(){
    // initialize cost with infinity
    for (int i = 0; i<5; ++i){
        cost[i] = INF;
    }
    // take node 0 (USD) as a source
    // initialize cost source with 0
    cost[0] = 0;

    // bellman ford algorithm, relax edges v-1 times
    for (int relax = 0; relax<4 ; ++relax){
        for (int i = 0; i<5; ++i){
            for (int j = 0; j<5; ++j ){
                if (cost[j] > cost[i] + adj[i][j].second){
                    cost[j] = cost[i] + adj[i][j].second;
                }
            }
        }
    }
}

```

Figure 11.E Implementation of Bellman ford algorithm on finding arbitrage condition, calculate shortest distance

Find if there is cycle by relaxing edges one more time, and also find the nodes that in the negative cycle. To find nodes that in the negative cycle, we have to save a node that in negative

cycle and get the data of all parents of nodes that in negative cycle.

```

// find the if there is negative cycle
int x = -1;
for (int i = 0; i < 5; ++i){
    for (int j = 0; j < 5; ++j){
        if (cost[j] > cost[i] + adj[i][j].second){
            x = j;
            parent[j] = i;
        }
    }
}
if (x == -1) {
    cout << "No negative cycle found.";
} else {
    for (int i = 0; i < 5; ++i)
        x = parent[x];

    vector<int> cycle;
    for (int v = x; v = parent[v]) {
        cycle.push_back(v);
        if (v == x && cycle.size() > 1)
            break;
    }
    reverse(cycle.begin(), cycle.end());

    cout << "Negative cycle: ";
    for (int v : cycle)
        cout << currency[v] << ' ';
    cout << endl;
}
}

```

Figure 11.F Implementation of Bellman ford algorithm on finding arbitrage condition, finding the negative cycle

After calling that procedure in main program we got the result of arbitrage situation below.

```
Negative cycle: GBP EUR GBP
```

Figure 11.G Implementation of Bellman ford algorithm on finding arbitrage condition, result of arbitrage condition

That is one example of arbitrage situation, that is occurred when converting currency from GBP to EUR and back to GBP again.

If we check the result by manual calculation, we got result like this

GBP -> EUR: 1.126

EUR -> GBP: 1.126 x 0.889 = 1.001014

We got result 1.001014 which is greater than 1 so it is an arbitrage situation.

IV. IDEAS ON IMPROVING THE GRAPH IMPLEMENTATION TO BE MORE EFFICIENT AND SCALABLE

Based on my implementation in C++ in previous section I realize some inefficiency and here are my ideas to improve the graph implementation.

1. Scrap the currency exchange data from internet

From my implementation, I put the currency data one by one by pushing it into vector. As a result, this cannot be used in a large scale because it will take a lot of time and can caused a wrong arbitrage condition because the currency values already change. So, what I think of as an alternative is that scrap the currency exchange data from

internet in a real time so that it will not miss any change overtime and it will save a lot of times also.

2. Make a script to perform the trading after searching for arbitrage condition

To get advantage of arbitrage condition, we have to trade the currency in real time after we find such condition. To do so, I have an idea to use script to perform such action to do simultaneous task which are searching for the arbitrage condition using bellman-ford algorithm then execute the forex trading based on arbitrage condition.

V. ADVANTAGES AND DISADVANTAGES FROM USING BELLMAN FORD ALGORITHM TO FIND ARBITRAGE

From section 4, we have shown that bellman ford algorithm on finding negative cycle can be used to find arbitrage situation. But now let us analyze more about the advantages and disadvantages of finding arbitrage condition using Bellman-ford Algorithm.

These are the advantages of using Bellman-ford algorithm to find arbitrage condition.

1. Easy to understand and implement

Bellman-ford algorithm is a simple algorithm that just doing node relaxation $V-1$ times and do one additional relaxation to find if there is cycle or not. If it is compared to other method of finding arbitrage such as Black-Scholes option pricing formula and statistical arbitrage algorithm, it is the easiest algorithm to understand and to implement

2. Modelled in graph

From the illustration in previous section, we know that this algorithm is based on graph, which is we modelled the graph of currency exchange first, then do the algorithm. If it is compared to other method of finding arbitrage such as Black-Scholes option pricing formula and statistical arbitrage algorithm, those two algorithms are based on heavy statistic and math calculations, so it is not really give clear image as compared to graph modelling.

These are the disadvantages of using Bellman-ford algorithm to find arbitrage condition.

1. Not really gives accurate numbers

When we use Bellman-ford algorithm, we have to convert all the currency exchanges into negative log, then do some summation to calculate the total in some cycles. When we convert it into logarithmic form, usually it will do some rounding off so what we get is not the actual number. As a result, the summation usually do not give the real number, so it can lead to wrong arbitrage condition compared to real condition.

2. Use a lot of memory, compared to other methods

Because bellman-ford algorithm is based on graph, so that, first we must save all the edges and nodes data into a graph. We also have to save the data of parent nodes and total cost when we traverse in cycle. To save all of those data, it needs more space compared to other method that used statistical model.

3. Not really used in real life

Usually traders do not use Bellman-ford algorithm to find arbitrage situations instead of statistical based arbitrage algorithm because statistical method gives more accurate condition.

VI. ACKNOWLEDGMENT

I would like to thank all the lecturers of the knowledge that is shared regarding discrete mathematics, especially Mr. Rinaldi Munir as my discrete mathematics class lecturer. This subject given me the chance to explore more regarding several topics in the subject and its application as well as given me the chance to practice my English writing skill. I would also like to thank my friends and families who support me in the process of learning and also making this paper.

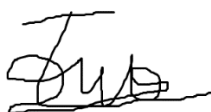
REFERENCES

- [1] <https://www.geeksforgeeks.org/mathematics-graph-theory-basics-set-1/>, 25 November 2019.
- [2] <http://mathworld.wolfram.com/AdjacencyMatrix.html>, 25 November 2019.
- [3] Gozali, William and Aji, Alham. *Pemrograman Kompetitif Dasar*. Jakarta: IA TOKI, pp. 126-127
- [4] <https://www.investopedia.com/articles/forex/11/why-trade-forex.asp>, 25 November 2019.
- [5] <https://www.seputarforex.com/belajar/forex/pengertian-dasar-forex/>, 25 November 2019.
- [6] <https://www.investopedia.com/terms/f/forex-f/forex-arbitrage.asp>, 25 November 2019.
- [7] <https://www.globalsoftwaresupport.com/forex-arbitrage-bellman-ford/>, 25 November 2019

STATEMENT

I hereby declare that the paper I wrote is my own writing, not an adaptation, or a translation of someone else's paper, and not plagiarism.

Bandung, 4 December 2019



Taufiq Husada Daryanto 13518058