Identifying Market Arbitrage Opportunity using Pathfinding Algorithm

Yoel Susanto 13517014
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517014@std.stei.itb.ac.id

***Abstract*—In economics and finance, arbitrage is the activity of buying and selling similar financial instruments on different markets to gain margin from the difference price. Price difference on the same asset could occur in different markets due to various factors such as supply and demand, currency fluctuation and political stability, this situation is known as market inefficiencies. Arbitrage usually involve buying and selling of similar financial instruments simultaneously at the same time; therefore, arbitrage carries low to no risk for the perspective trader. While arbitrage promises to produce margin for low to no risk, it proves to be very hard to identify arbitrage opportunity. The use of computer in High Frequency Trading addresses market inefficiency in matter of seconds. This condition makes it impossible for human trader to gain margin from arbitrage. This paper will discuss on method of identifying arbitrage opportunity using pathfinding algorithm. This solution would enable traders to gain margin from arbitrage even in High Frequency Trading environments.**

Keywords—Arbitrage, Market, Riskless Activity, Pathfinding, Negative Cycle

I. Introduction

Trading has been a preferred way of gaining margin. There are many kinds of trading using various of method. Some involve low risk and some involved high risk. In this paper we will discuss about market arbitrage. A low risk method of gaining capital through selling and buying of similar instruments. We will also discuss the challenge and problems traders face when trying to find arbitrage opportunity. We will also propose some solutions to tackle the challenge and enable traders to gain profit from this low risk method.

II. Background On Market Arbitrage

Arbitrage is the activity of buying and selling similar financial instruments on different markets to gain margin from the difference price. Consider a simple example case of arbitrage as follow: the stock of a company is trading at $100.00 on the Indonesia Stock Exchange while the same stock of company X is trading at $100.5 at Singapore Exchange Centre. As a trader, we could buy the stock of company X in Indonesia Stock Exchange while at the same time selling it at Singapore Exchange Centre. We would gain a margin of $0.5 for every stock we trade. A more complicated market arbitrage would be triangle arbitrage. Imagine having three banks exchanging various kinds of currencies.

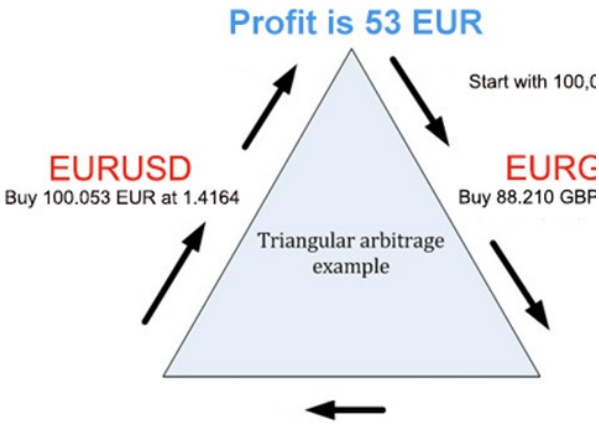

Fig 1. Example of Triangular Dependency

We could gain margin by selling and buying currency in the order of the transaction cycle. More complicated arbitrage would involve more than just three institutions and it poses a challenge to actually identify and exploit them.

III. Background On Graphs

A. Definition

Graphs are discrete structures consisting of vertices and edges that connect these vertices. There are different kinds of graphs, depending on whether edges have directions, whether multiple edges can connect the same pair of vertices, and

whether loops are allowed.[2]

Formally, *graphs* are denoted as

$$G = (V, E)$$

where V is a nonempty set of *vertices, nodes, or points* and E is a set of *edges, lines or arcs*. The vertices and edges inside a graph can represent different kind of objects and relations, their meanings differ significantly depending on the context where the graph is being used to represent information.

Vertices on graph are usually labelled using alphabet or numbers or the combination of both while the edges are usually labelled using e followed by a number. subscript Vertices can have multiple or no edges associated with it, while edges must have either one or two vertices associated with it. Edges are commonly denoted as

$$e = (u, v)$$

where u and v are the vertices connected by e.

Generally, graphs are visualized using points to represent vertices and line segments to represent edges.

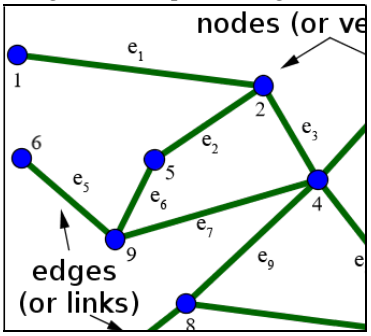

Fig 2. A graph consisting of 10 vertices and 11 edges V = {1, 2, …, 10}, E = {$e_1, e_2, \ldots, e_{11}$}

https://mathinsight.org/media/image/image/small_undirected_network_labeled.png (accessed December 8, 2018)

B. Types of Graph

There are many ways to classify graphs. Based on the existence of parallel edges, graphs can be classified into two kinds:

1. **Simple Graph**
   A simple graph is a graph with no loop or parallel edges connecting a pair of vertices. Edges in simple graph are concerned with the order of the vertices they are connected to. We can safely say that edge (u, v) is the same as edge (v, u).

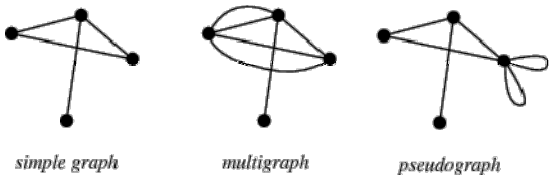

Fig 3. Simple graph(left), multigraph(middle) and pseudograph(right)

2. **Multigraph**
   Multigraph are graphs consisting of parallel edges. In addition, when a vertex in a graph has edge connecting to itself, the graph is classified as **pseudograph.**

Based on the whether the edge of a graph is associated with ordered pair of vertices or not, we classify graph into two kinds:

1. **Undirected Graph**
   Edges in directed graph are not associated with any order of the vertices they connected. Edge (u, v) is the same as edge (v, u).
2. **Directed Graph**
   On the other hand, directed graphs are graph with edges that have direction associated with it. In this case we say that edge (u, v) *start* at u and *end* at v. This means that edge (u, v) is not the same as edge (v, u).

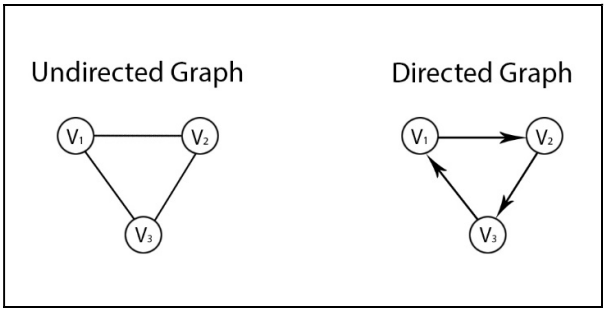

Fig 4. Undirected and directed graph

https://www.e-education.psu.edu/geog597i_02/sites/www.e-education.psu.edu.geog597i_02/files/Lesson8/Geog597i_Lesson8_directedgraph.jpg (accessed December 8, 2018)

C. Graph Terminology

1. Adjacent
   If two vertices u and v in a graph G is connected by an edge, u and v are adjacent in G.
2. Incident
   An edge is incident with both of the vertex it

connected.

3. Isolated Vertex
   The isolated vertex in a graph is a vertex which doesn't have any edge incident with it.
4. Null Graph
   Null graph is a graph which doesn't contain any edges. Nonetheless the null graph may contain any number of vertices.
5. Degree
   Degree of a vertex inside a graph is the sum of edges associated to it.
6. Path
   Path in a graph is a sequence of edges that travels from one vertex to the other vertices.
7. Circuit
   Circuit is path that begins and ends at the same vertex.
8. Connected
   Two vertices *a* and *b* inside a graph are said to be connected if there exist a path that begins at *a* and ends at *b*.
9. Subgraph
   A subgraph *H* of graph *G* which contain a subset of vertices that G has and also contain a subset of the edges that G has.
10. Spanning Subgraph
    A subgraph of a graph which contains all the vertices inside the original graph is called the spanning subgraph.
11. Cut-Set
    A cut-set of a graph is a set of edges which would cause a graph not to be connected if it is removed.
12. Weighted Graph
    A graph *G* is called as weighted graph if edges inside graph *G* has a value assigned to it instead of just lines.
13. Special Graph

There are several kinds of special graph such as: complete graph, regular graph and bipartite graph.

D. Representing Graphs

It is easy for human to represent graph as depicted in pictures using dots and lines, but it is not such in the case of representing graph in computers. As computer store data using ones and zeros, we cannot simply draw a graph into graph. We need to a systematic and reliable way of representing graph such that the data could be meaningful. There are several ways we could represent graph:

1. Adjacency Matrix
   When we use adjacency matrix to represent graph, we define a matrix *A* the size of *N x N* where *N* is the number of nodes inside a graph. Suppose a_{ij} is an element of A, a_{ij} is 1 if there exist an edge connecting v_i and v_j. Otherwise, a_{ij} will equal to 0.

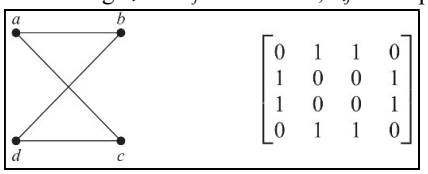

Fig 5. Example of a graph and its corresponding adjancency matrix

Discrete Mathematics and Its Applications, Seventh Edition, Kenneth H Rosen, Page 669

2. Incidence Matrix
   Incidence matrix focus on vertex and which edges is it incident with. If we have a graph $G = (V, E)$ consisting of vertices $v_1, v_2, \ldots, v_n$ and edges $e_1, e_2, \ldots, e_m$, then the incidence matrix will have a size of *n x m*. Suppose m_{ij} is an element of M, m_{ij} is 1 if e_j is incident with v_i and 0 otherwise.

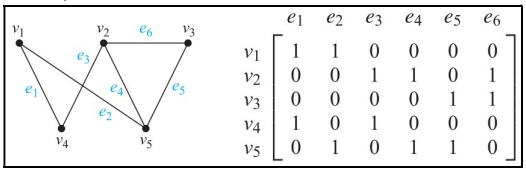

Fig 6. Example of a graph and its corresponding incidence matrix

Discrete Mathematics and Its Applications, Seventh Edition, Kenneth H Rosen, Page 669

E. Negative Cycle

Negative cycle is a special cycle in directed graph whose summed values of edge weight is less than zero. This cycle cause problem for various mainstream path finding algorithm. Generally, path finding algorithm update the distance of a node if it could find a path with less weight than the previously known. The negative cycle causes the algorithm to never stop because it could get cheaper route each time it goes through the negative cycle, getting negative infinity for the shortest path.

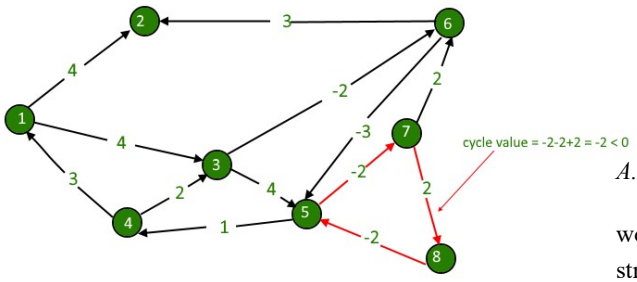

Fig 7. Example of negative cycle

IV. Path-Finding Algorithm

Pathfinding algorithm are usually utilized for finding the shortest path possible from a given point A to point B. Different pathfinding algorithms try to achieve this in very different ways, but generally the algorithms consider the weights of each edge in the graph and try to find the smallest sum for each node in the graph.

A. Bellman Ford Shortest Path

The Bellman Ford Shortest Path algorithm provide solution to single-source shortest-paths problem. Given a node *s* in directed graph *G containing |V| nodes*, the algorithm will calculate the shortest distance from *s* to *e, distance[e]*, for every edge *e* in the graph provided there is no negative cycles reachable from *s*. Here is the pseudocode for Bellman Ford SP:

```
for (int v = 0; v<G.V; v++)
{
    distTo[v] = double.PositiveInfinity;
}

distTo[s] = 0.0;

for (int i=0; i<|V|; i++)
{
    foreach (DirectedEdge e in G.Adj(v))
    {
        if (distTo[e.To] > distTo[v] + e.Weight)
        {
            distTo[e.To] = distTo[v] + e.Weight;
            edgeTo[e.To] = e;
        }
    }
}
```

Using Bellman Ford Algorithm, we could allow the graph to have negative weight of edge. Even though the algorithm is considered slower than Dijkstra, Dijkstra algorithm does not work with graph containing negative weight edges. In addition, a slight modification could be implemented to the Bellman Ford Algorithm to enable it to detect negative cycles in a graph.

Time Complexity = O (|V| |E|)

Space Complexity = O (|V|)

V. Identifying Arbitrage Opportunity Using Negative Cycle Detection

A. Modelling the Market as Directed Edge-Weighted Graph

To solve this problem using path finding algorithm, we would first need to convert the data from the text form into a structure we could easily process using path finding algorithm. Below is the code for reading the input file and construct a graph model of the market:

```
public EdgeWeightedDigraph(TextInput input) :
this(input.ReadInt())
{
    int E = input.ReadInt();
    for (int i = 0; i < E; i++)
    {
        int v = input.ReadInt();
        int w = input.ReadInt();
        // validation wil be done from within
AddEdge
        double weight = input.ReadDouble();
        AddEdge(new DirectedEdge(v, w, weight));
    }
}
```

```
EdgeWeightedDigraph G = new EdgeWeightedDigraph(V)
for (int v = 0; v < V; v++)
{
    name[v] = StdIn.ReadString();
    for (int w = 0; w < V; w++)
    {
        double rate = StdIn.ReadDouble();
        DirectedEdge e = new DirectedEdge(v, w, -
Math.Log(rate));
        G.AddEdge(e);
    }
}
```

In this model of the data, the node inside the graphs represent each kind of currency while the weight of each edge represents the exchange rate between currency. The weight of the edge is calculated using the minus of the logarithm of original value. The reason behind using this method of calculation is to reduce computational cost of computing the path weight. In the original problem, we would need to do the calculation by using multiplication operation. In the transformed problem we could achieve the same result using addition instead of multiplication thus making the algorithm faster.

B. Identifying Negative Cycle in the Graph Model

To identify arbitrage opportunity in the market model, we would need to identify negative cycle in graph model. A

negative cycle is reachable from the source of a graph if and only if the LIFO Queue of the algorithm is not empty after |V|th pass through all the edges. We first do this by identifying the cycle in a subgraph of edges in the original graph. The code for the creating the subgraph and finding the cycle:

```
private void findNegativeCycle()
{
    // edgeTo[v] = last edge on shortest s->v
path
    int V = edgeTo.Length;
    EdgeWeightedDigraph spt = new
EdgeWeightedDigraph(V);

    for (int v = 0; v < V; v++)
        if (edgeTo[v] != null) {
            spt.AddEdge(edgeTo[v]);
        }

    EdgeWeightedDirectedCycle finder = new
EdgeWeightedDirectedCycle(spt);
    cycle = finder.GetCycle();
}
```

```
private void dfs(EdgeWeightedDigraph G, int v)
{
    onStack[v] = true;
    marked[v] = true;
    foreach (DirectedEdge e in G.Adj(v))
    {
        int w = e.To;

        // short circuit if directed cycle found
        if (cycle != null) return;
        else if (!marked[w])
        {
            //found new vertex, so recur
            edgeTo[w] = e;
            dfs(G, w);
        }
        else if (onStack[w])
        {
            // trace back directed cycle
            cycle = new LinkedStack<DirectedEdge>();
            DirectedEdge eTemp = e;
            while (eTemp.From != w)
            {
                cycle.Push(eTemp);
                eTemp = edgeTo[eTemp.From];
            }
            cycle.Push(eTemp);
            return;
        }
    }
    onStack[v] = false;
}
```

C. *Calculating Margin of Arbitrage Opportunity*

After we determine that there is an arbitrage opportunity in the market, we will need to calculate the potential margin to be gained from the market. We do this by making a stake of an amount of money and traverse each edge in the negative cycle of the graph as if we are doing the trading of the financial instruments. In doing the calculation, we reverse the effect of the negation and logarithm by doing another negation and exponentiation of the edge weight:

```
// find negative cycle
BellmanFordSP spt = new BellmanFordSP(G, 0);
if (spt.HasNegativeCycle)
{
    double stake = 1000.0;
    foreach (DirectedEdge e in spt.GetNegativeCycle())
    {
        Console.Write("{0,10:F5} {1} ", stake,
name[e.From]);
        stake *= Math.Exp(-e.Weight);
        Console.Write("= {0,10:F5} {1}\n", stake,
name[e.To]);
    }
}
else
{
    Console.WriteLine("No arbitrage opportunity");
}
```

D. *Test Data and Results*

After the exposition of the method and algorithm we are going to be using, we are going to test how the method will work on the test data. First, we have the test data as follow:

```
5
USD 1      0.741  0.657  1.061  1.005
EUR 1.349  1      0.888  1.433  1.366
GBP 1.521  1.126  1      1.614  1.538
CHF 0.942  0.698  0.619  1      0.953
CAD 0.995  0.732  0.650  1.049  1
```

The first number in the documents define the number of currencies involved in the data. Each row defines the exchange rate of a currently toward the other currency in the data. For example, the first row, third column has value 0.741 means 0.741 euro for each US dollar. Here we present the results of the calculation:

```
5
USD 1      0.741  0.657  1.061  1.005
EUR 1.349  1      0.888  1.433  1.366
GBP 1.521  1.126  1      1.614  1.538
CHF 0.942  0.698  0.619  1      0.953
CAD 0.995  0.732  0.650  1.049  1
Results:
1000.00000 USD =  741.00000 EUR
 741.00000 EUR = 1012.20600 CAD
1012.20600 CAD = 1007.14497 USD
```

Fig 8. Results

The results show that using a capital of 1000 USD we could gain profit around $7.

CONCLUSION

Arbitrage is a low to no risk trading activity. Although arbitrage poses very little risk, the process of identifying arbitrage opportunity is not trivial. To tackle this problem, we could utilize the help of computer by building a graph model of the market situation and using algorithm to find negative cycle. This kind of algorithm enables personal trader to gain profit by taking part in a low risk trading activity.

ACKNOWLEDGMENT

First, the author would like to thank God for giving the author the ability to finish the paper. The author would also like to thank the lecturers of Algorithm Strategies for the guidance given during this semester. In addition, the author would also like to thank his parents, family and friends who have supported the author to finish the paper.

REFERENCES

[1] K. H. Rosen, *Discrete Mathematics and Its Applications*, 7th Edition, 2013.

[2] "Shortest Path", https://algs4.cs.princeton.edu/44sp/, diakses pada tanggal 26 April

[3] "Dijkstra's Algorithm", http://web.stanford.edu/class/archive/cs/cs161/cs161.1176/Lectures/CS161Lecture11.5.pdf, diakses pada tanggal 26 April

[4] S. Robert, K, Wayne, *Algorithms*, 4th Edition, 2011.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

Yoel Susanto - 13517014