On the application of Bellman-Ford algorithm

---

1 Introduction

Arbitrage is a practice of taking advantage of a difference in prices in two or more markets.
Example: You get your senpai's homework solution for an onigiri (100 yen) and sell it to your classmates for 500 yen.

In case of homework arbitrage graph theory may not be of great use, but for more complex case one can consider a graph representation for arbitrage schemes. Namely, consider vertex is an object that you sell or buy, edge is an operation with this object and weight on an edge is the cost for such an operation. With the notion from above, consider currency arbitrage. It can be conveniently represented with directed weighted graphs as follows:

1. You have your initial vertex, for example 100 yen
2. You go to rubles vertex and 100 yen becomes 50 rubles as you go through yen-ruble edge.
3. You do some other selling-buying N times.
4. (Hopefully) In the end you get more than your starting 100 yen.

But how can one find a specific sequence of exchanging currency such that you obtain more than you have started with?

2 Exploration

To begin with, one observes that we have to go through a cycle (as we begin with the currency V and end at currency V). Now one has to determine the expression for edge-weight.
Define R for rate of the exchange, $R[v_i, v_j]$ is rate from vertex i to vertex j.
Now to determine edge weights, one observes that we want

$$R[v_1, v_2] \cdot R[v_2, v_3] \cdots R[v_{k-1}, v_k] \cdot R[v_k, v_1] > 1$$

if and only if

$$\frac{1}{R[v_1, v_2]} \cdot \frac{1}{R[v_2, v_3]} \cdots \frac{1}{R[v_{k-1}, v_k]} \cdot \frac{1}{R[v_k, v_1]} < 1$$

If one takes log of both sides of the inequality, one can express this condition as

$$\log\left(\frac{1}{R[v_1, v_2]}\right) + \log\left(\frac{1}{R[_{2,3}]}\right) + \cdots + \log\left(\frac{1}{R[_{k-1,k}]}\right) + \log\left(\frac{1}{R[_{k,1}]}\right) < 0$$

Now one can define the weight of edge (v_i, v_j) as

$$w(v_i, v_j) = \log\left(\frac{1}{R[i, j]}\right) = -\log\left(R[v_i, v_j]\right)$$

then the goal is to find whether there exists a negative-weight cycle in a given graph G with edge weights as proposed above.
After researching for an algorithm to determine negative-weight cycles one finds that Bellman-Ford algorithm allows to detect negative-weight cycles.

3 Bellman-Ford algorithm

Bellman-Ford algorithm is an algorithm used to find the shortest paths from a single source vertex to all other vertices in a weighted graph, even if the graph contains negative weight edges. It is similar to Dijikstra's algorithm which was covered during the lectures.

Explanation of the algorithm
The Bellman-Ford algorithm's principle is that it starts with a single vertex and calculates the distance to each vertex. The distance is initially unknown and assumed to be infinite, but as time goes on, the algorithm relaxes those paths by identifying a few shorter paths.
The algorithm works by iteratively relaxing all edges n-1 times, where n is the number of vertices. Relaxation means attempting to improve the shortest known distance to each vertex by considering all possible edges (basically forgetting about paths that are longer).

How does it detect negative cycles?
After n-1 iterations, the algorithm has theoretically computed the shortest paths that do not contain any cycles (positive or negative weight). However, if we can still relax an edge in the n-th iteration (i.e., find a shorter path), this indicates the presence of a negative weight cycle in the graph.

Why does this indicate a negative weight cycle?
In a graph without negative weight cycles, the shortest path from the source vertex to any other vertex will not change after n-1 iterations. This is because the longest possible path in any graph with n vertices has at most n-1 edges. If after n-1 iterations, we can still find a shorter path to some vertex v from vertex u by relaxing an edge (v_i, v_k), this means there is a cycle where the sum of the edge weights is negative.

4 Application

Now, with all the knowledge from above, one can use Bellman-Ford algorithm to try can find if there exists an opportunity for currency arbitrage.
Python implementation of Bellman-Ford algorithm for the purpose of currency arbitrage was found online, credit is given in the appendix section.
Results After inserting today's exchange rates (13.07.24, data obtained from google finance) for six currencies JPY, USD, EUR, RUB, AUD, GBP and obtaining the following matrix

```
rates = [
    [1, 0.00633, 0.0058, 0.56, 0.0094, 0.00488],
    [157.8450, 1, 0.9157, 87.8038, 1.4789, 0.7705],
    [172.3763, 1.0905, 1, 95.8949, 1.6139, 0.8415],
    [1.7997, 0.0114, 0.0104, 1, 0.0168, 0.0088],
    [106.8007, 0.6766, 0.6196, 59.6, 1, 0.5213],
    [204.8319, 1.279, 1.1885, 113.9604, 1.9194, 1],
]

currencies = ('JPY', 'USD', 'EUR', 'RUB', 'AUD', 'GBP')
```

Figure 1: Matrix with exchange rates. Explanatory screenshot how to read the matrix is given in the appendix.

-log was taken of each rate, as explained in the section exploration, and the results are presented in the photo below

```
Arbitrage Opportunity:

USD --> RUB --> GBP --> USD
Arbitrage Opportunity:

JPY --> RUB --> GBP --> JPY
Arbitrage Opportunity:

JPY --> RUB --> GBP --> JPY
Arbitrage Opportunity:

JPY --> RUB --> GBP --> JPY

=== Code Execution Successful ===
```

Figure 2: Code results

Now we shall manually check the validity of the results.

1. USD $\rightarrow$ RUB $\rightarrow$ GBP $\rightarrow$ USD
   $1 \rightarrow 87.8038 \rightarrow 0.7726664 \rightarrow 0.988249$
   Well, this is wrong. That may be due to some code issues.

2. JPY $\rightarrow$RUB $\rightarrow$ GBP $\rightarrow$ JPY
   $1 \rightarrow 0.56 \rightarrow 0.004928 \rightarrow 1.01$
   Indeed we have obtained 0.01 from 1 yen. It means for each 100 yen we obtain 1 yen, which is not that bad, as this is basically money out of nothing.

5 Conclusion

It was quite surprising that graph theory can be used in economics, as I did not think about that, but when this example came to my mind I was really curious to check if it would really work, as theoretically it should.
One can make a real life application of that by considering other arbitrage options which are actually available (as google finance is some ideal exchange rates, as you will not find such rates in any currency exchange agency) and one with better programming skills can make the code better.
But the idea works, this is what important.

6 Appendix

```
[1, 0.5, 2, 0.1],   # Exchange rates from currency A to [A, B, C, D]
[2, 1, 4, 0.2],     # Exchange rates from currency B to [A, B, C, D]
[0.5, 0.25, 1, 0.05], # Exchange rates from currency C to [A, B, C, D]
[10, 5, 20, 1],     # Exchange rates from currency D to [A, B, C, D]
```

Figure 3: How to read matrix from figure 1

1.

2. https://gist.github.com/anilpai/dd00a9671c062390c848952eaddbbe1e - source to the code used