Market depth analysis of the cryptocurrency market

Rune K. Svendsen

IT University of Copenhagen

runs@itu.dk

May 15, 2019

Abstract

*The current practice of measuring the value of a cryptocurrency by "market cap" is inaccurate — particularly for the smallest (least liquid) cryptocurrencies. By simply multiplying the latest price of a cryptocurrency coin with the number of coins in existence, a theoretical figure is obtained that has little relevance to the real world. By instead measuring the "market depth" of a cryptocurrency, we gain insight into real world circumstances, by answering the question "**how many coins, of a given cryptocurrency, can be sold/bought while moving its price by no more than "x" percent?**". This paper describes an algorithm that answers this question, as well as a Haskell implementation of it. Using the implementation, the paper presents a ranking of cryptocurriencies against each other, based on applying this algorithm to order book data from five large cryptocurrency exchanges.*

1. Introduction

Market capitalization is often used as a measure of the value of cryptocurrencies. This paper critiques the use of market capitalization as a measure of value of cryptocurrencies, and offers an alternative measure: the amount of cryptocurrency that can be bought/sold while moving its price by no more than *x*%. By focusing on *actual* market data (the order books of cryptocurrency exchanges), we can see how much a cryptocurrency can be sold for — right now — while not exceeding a given change to its price.

2. Background

2.1. *The foreign exchange market*

2.1.1 Market example

The *foreign exchange market* is the market in which currencies (such as the US dollar, the euro, and the British pound) are sold in exchange for each other. To better explain the terminology used in the foreign exchange market, we start by considering a very simple market: the market in which *tomatoes* are exchanged for *US Dollars*. The *name* of a market is composed of the *symbols* of the two assets that are exchanged in the given market. Symbols are a sequence of 3-6 capital letters that uniquely identify a specific asset. If we define the symbol of tomatoes to be TOMATO and the symbol of US Dollars to be USD, then a market in which tomatoes are echanged for US Dollars is called *TOMATO/USD*. In other words, a market name is two symbols (that are traded in this market) separated by a forward slash (/).

2.1.2 *Quote* and *base* currency

The symbol preceding the slash (in this case TOMATO) is — using foreign exchange market terminology — the *base* currency, and the symbol after the slash (USD) is the *quote* currency (we ask the reader to bear with us, and pretend that the tomato is a currency). The unit used to specify *quantities* in a given market is the base currency and the unit of *prices* is quote currency per base currency. Consequently, in the TOMATO/USD market, we might say that we want to buy 5 tomatoes (the quantity) for 0.25 USD per tomato (the price). We do *not* say that we want to sell 1.25 USD for 4 tomatoes per USD — although the two statements exactly equivalent. Quantities are measured in the base currency, and prices are measured in quote currency per base currency.

2.1.3 Buy and sell orders

A market is a place for buyers and sellers to meet. Buyers and sellers express interest in entering into an exchange through an *order*. An order has two components: *quantity* and *price*. Two separate types of orders exist: *buy* orders and *sell* orders. Generally, people with base currency — who wish to obtain quote currency — issue *sell* orders, while people with quote currency — who wish to obtains base currency — issue *buy* orders. In the TOMATO/USD market, people with tomatoes — who are interested in acquiring USD — issue *sell* orders to express this interest, while people with USD — who are interested in acquiring tomatoes — issue *buy* orders.

2.1.4 Order books / exchanges

An order book is a collection of sell and buy orders for the same market. An order is only in the order book if it hasn't been *matched* with another order. A buy order and a sell order match if the buy order's price is greater than or equal to the price of the sell order. Exchanges maintain order books for many markets, and receive orders from clients. When the exchange receives an order from a client that matches with an order already in the order book, an exchange happens. If the quantity of the received order is *less than* that of the order with which it has matched, the quantity of the received order is subtracted from the matched order book order, and it remains in the order book. If the quantity of the received order is *greater* than that of the matched order, the matched order is removed from the order book, its quantity subtracted from the incoming order's quantity and the process repeats until the received order's quantity has been matched completely.

2.2. *Liquidity*

2.2.1 Market liquidity / slippage

A *liquid* market is one in which the price moves slowly in response to buying and selling. For example, if selling $10mm worth of silver moves the silver price down by *0.1%*, and selling the same dollar amount worth of copper moves the copper price down by *2%*, then silver is more liquid than copper. Liquidity is also called *market depth*.

How much the price of an asset moves in response to a trade is called the *slippage* of that trade. In the above example, executing the $10mm silver sell order results in a slippage of 0.1%.

2.2.2 Measuring liquidity

This paper presents the following measure of buy/sell liquidity:

- *Buy* liquidity: the quantity that can be *bought* until the price has *increased* by a certain percentage
- *Sell* liquidity: the quantity that can be *sold* until the price has *decreased* by a certain percentage

In other words, we continue to exchange a specified currency through *all* available markets, until the price of the next trade — in *any* market (see: Mispriced markets) — would change by more than some predefined percentage (slippage).

2.3. *Market capitalization*

2.3.1 Overview

Market capitalization (or "market cap" for short) is a measure used to value companies in the stock market. This figure is obtained by multiplying the price of a single share (as reflected by the most recent trade) with the number of shares issued. For example, at some point in time, if the last traded price of the Intel stock[1] is 46.2 USD[2], and there are 4,497,000,000[3] issued Intel shares, then the Intel stock has a market cap of $46.2 * 4,497,000,000 = 207,761,400,000$ USD — ie. roughly 210 billion US Dollars.

The market cap figure is relevant in the stock market because most company shares are issued under rules that allow a majority holder to buy the shares of the minority if a majority of the minority agrees with the offer. This means that the market cap figure is relevant as a measure of the price of a company's oustanding shares, as it roughly reflects what a majority shareholder would have to pay for the remaining shares.

2.3.2 Ranking currencies by market capitalization

Cryptocurrencies — like all other assets except company stock — are not issued under rules that allow majority holders to force a minority to sell their holdings. It is not possible to acquire all issued/created coins for a pre-defined price, by having a majority of the remaining holders agree to an offer. Attempting to buy all of a given cryptocurrency will cause its price to increase — the less liquid it is the faster.

Let's consider an example, where we are dealing with a very illiquid cryptocurrency, of which there are 100 trillion issued coins (currency units). If the only order book that exists for this cryptocurrency contains only a single sell order offering to sell a one coin for 10 cents USD, and a buyer matches this order, then the market cap of the cryptocurrency in question would be 10 trillion USD. This is despite the fact that there are no orders at all in the only order book for this currency, meaning the 100 trillion issued coins cannot be exchanged for USD at all. This is obviously an extreme example, but this principle applies to illiquid cryptocurrencies, albeit resulting in less extreme figures.

Applying the market *depth*/liquidity measure — proposed in this paper — to the above example currency would value the it at 10 cents before the order is matched, and at 0 USD after. The market depth measure tells us how much a given quantity of a cryptocurrency can sold/boght for right now.

2.4. Challenges

2.4.1 Many exchange paths

Given an order book for a currency/market, calculating the liquidity for this currency is simple. However, there exists thousands of cryptocurrency markets, spread out over dozens of exchanges. For example, if we wish to sell Ether (ETH) to obtain US Dollars (USD), there are a multiple "paths" through which this can be done:

- **The simple one:** sell ETH directly for USD (in the ETH/USD market)
- **The slighly more complex one:** sell ETH for BTC (bitcoin) (in the Bitfinex[4] ETH/BTC market), then sell the acquired BTC for USD (in the Bitstamp[5] BTC/USD market)

and even more complex paths, going through multiple markets and exchanges.

In short, there are dozens -- if not hundreds — of ways to obtain USD starting from ETH, and this applies to any currency.

2.4.2 Mispriced markets

When answering the question "*how much, of a given cryptocurrency, can be bough/sold while moving the price less than x%?*", there are at least two ways to define the term *price*:

1. The final price of moving from some currency to e.g. USD through multiple markets on the way

   In this case, the full order book is used, and the price of the first matched order is recorded. The algorithm runs until the price difference between the first matched order and next order is greater than the specified percentage. It may involve prices in single markets changing more than the specified percentage.

2. The price in *all the markets* moved through (ie. for all order books in all markets)

   Here, all order books are trimmed before being used, by removing all buy and sell orders that are further away, than the specified percentage, from the best-priced order. The algorithm runs until no more paths can be found to the given currency. It gives the guarantee that the no price in *any* market will change by more than the specified percentage.

The first option is problematic in the presence of mispriced markets — markets in which the prices are in disagreement with those of all other markets. The first order matched by the algorithm might match an order in this mispriced market, giving it e.g. a very low price compared to all other markets. This price will be used as the initial price for calculating the price change in percent, and it's possible that no orders from any other markets will match, since the price in this mispriced market is far away from the specified percentage. To solve this potential issue, the second option is used.

3. Algorithm

3.1. Overview

The purpose of the algorithm is to enumerate all *edge-disjoint* shortest paths in a graph, in ascending order of path length. Two paths are considered edge-disjoint if they have no edges in common. Given the data model explained below, this results in an algorithm that is able to enumerate, in ascending order of price, all ways to move from some currency to another.

The steps of the algorithm are as follows.

1. Build a graph in which:
    1. A vertex is a *currency*
    2. An edge is a *sell order*:
        1. "from"-vertex is the sell order's *quote* currency
        2. "to"-vertex is the sell order's *base* currency
        3. weight is the logarithm of the sell order's *price*
2. Remove negative-weight cycles from the graph
3. Repeatedly:
    1. Apply the Bellman-Ford[1] algorithm to obtain a shortest path
    2. Calculate the *capacity* of this shortest path (see: Path capacity) and modify the graph to obtain a new graph in which this capacity has been subtracted from the sequence of edges comprising the shortest path
    3. Add the obtained shortest path along with its capacity to the result set
    4. Apply 3.1 to the newly obtained graph (the algorithm terminates when 3.1 does not find a shortest path)

3.2. Data model

A graph is used to represent currencies and markets. The vertices of the graph are currencies (e.g. *USD*, *EUR*, *BTC*). Sell orders comprise the edges of the graph.

Edges are directed, with their *from*-vertex being the sell orders' *quote* currency, and the *to*-vertex being the sell order's *base* currency. The reason for this is that the seller has some quantity of base currency that it wants to exchange for quote currency, which means the seller wants to "move" from base currency to quote currency. The buyer moves in the opposite direction of the seller, by giving quote currency to the seller and obtaining base currency. This results in a graph whose edges are ways for a buyer to exchange currency denominated in the *from*-vertex for the currency of the *to*-vertex, by matching with a seller who moves in the opposite direction.

Edges are also weighted, with the weight being the sell order's *price*.

The following figure shows a graph created from a BTC/USD order book containing the two following orders:

- Sell order
  - Price: 4500 USD/BTC
  - Quantity: 1 BTC
- Buy order
  - Price: 4450 USD/BTC
  - Quantity: 1 BTC

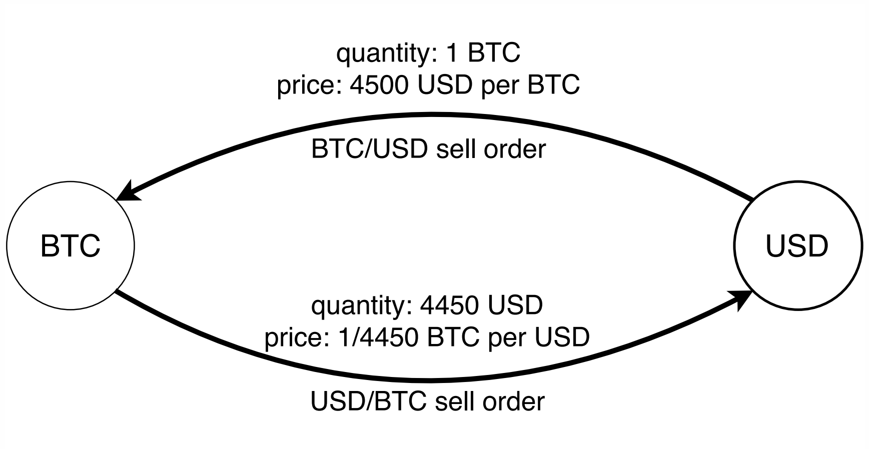

Figure 3.2.1: Simple sell order graph

Buy orders are converted into sell orders because we're interested in obtaining currency for the *lowest* possible price, which fits a shortest path algorithm (by lowest *price* becoming lowest *weight*). Therefore, the buy order in the BTC/USD order book has been converted to a sell order in the fictitious USD/BTC market. The USD/BTC sell order is exactly equivalent to BTC/USD buy order, where the only difference is the unit of quantity and price.

As a way to increase performance of the algorithm, an edge is not a *single* sell order, but rather a *sorted list* of *all* sell orders with the same base and quote currency. This is not a necessity, since it would be possible to simply add all sell orders to the graph, which would end up having thousands of edges between two given vertices. However, since we're applying a shortest path algorithm to the graph, we know that only the edge with the lowest weight/price will be selected by the shortest path algorithm in the end. Thus, we reduce the number of edges considered by the algorithm by simply keeping a list of sorted sell orders as edges, and only focusing on the sell order in this list with the lowest price/weight.

3.3. Negative cycles

When multiple order books from different exchanges are combined, *arbitrage opportunities* may appear. Arbitrage opportunities, in terms of the edges of the graph, is a sell order with the same base and quote currency and a price of less than 1. This means it's possible to buy e.g. 1000 USD for less than one USD per USD. It's a so-called "risk-free profit", at least on paper. In practice, fees and delays in transferring currency from one exchange to another makes it difficult to earn money on arbitrage opportunities. The reason the algorithm finds these is not that they're interesting, but because the Bellman-Ford algorithm does not work in the presence of negative edge weights. Hence, these are removed before the shortest path-part of the algorithm is executed.

3.4. *Path capacity*

A shortest path, from some currency to another, has the form of a sequence of sell orders. For any two adjacent sell orders in this sequence it holds that the base currency of the left order is equal to the quote currency of the right order. Example sell order sequence:

- BTC/USD
- ETH/BTC
- XRP/ETH
- BNB/XRP

From a buyer's perspective, this is equivalent to a path from USD to BNB (going through BTC, ETH, and XRP — in that order).

The quantity of a sell order is denominated in the base currency of the order. In order to calculate how much volume this sequence of orders can carry, we start by multiplying the quantity of the second order with its price, in order to obtain the quantity of the second order denominated in its quote currency — the same as the base currency of the order that precedes it. Once this has been done, the *minimum* of the two quantities is selected, and used to compare to the quantity of the third order's quantity converted to its quote quantity. This repeats until the last order, at which point the maximum capacity of the path — denominated in BNB — is obtained.

By going in the opposite direction — converting BNB to XRP, and then XRP to ETH and so on — we can subtract the BNB-denominated maximum capacity from the quantity of all sell order-edges of the given path. This results in a path whose capacity has been reduced by the edge(s) with the lowest capacity. The edges left with a capacity of zero are removed, resulting in the next sell order — in the sorted list of sell orders that comprise an edge — replacing the zero-quantity sell order.

3.5. *Time complexity*

Given that the time complexity of the Bellman-Ford-Moore algorithm is $O(V(V+E))$, then enumerating the capacity of all paths (of which there are P) from some currency (e.g. USD, EUR) to all cryptocurrencies (of which there are V) is bounded by $O(PV^2(V+E))$.

This is because the time complexity of obtaining a *single* path is $O(V(V+E))$ (one iteration of Bellman-Ford), which needs to be done for each path (of which there are P), which — again — needs to be done for all cryptocurrencies (of which there are V).

3.6. Scalability

Given a time complexity of $O(PV^2(V+E))$, an increase in the number of *distinct* markets ($|E|$) (ie. markets whose base and quote currency don't match that of any existing markets) will not slow down the algorithm as much as an increase in the number of cryptocurrencies $|V|$.

Also, the addition of a market that already exists (one that has the same quote and base currency as an existing market) will increase the asymptotic running time in a linear fashion, such that enumerating e.g. 10 paths will have an upper bound 10 times that of enumerating a single path.

The bottom line is that the algorithm scales very poorly — $O(V^3)$ — as the number of cryptocurrencies increase. There are a bit more than two thousand cryptocurrencies listed on coinmarketcap.com[6]. If the number of cryptocurrencies were to increase by a factor of five (from 2000 to 10000), the algorithm would become 125 times slower (5^3).

4. Implementation

4.1. Details

The implementation[7] is written in Haskell. It uses a Haskell translation[8] of Sedgewick & Wayne's `BellmanFordSP` algorithm[9]. `BellmanFordSP` features negative cycle-detection with a running time of $O(V(V+E))$.

4.2. Experiments

A graph containing orders based on all order books[10] from the exchanges *Bitfinex*[4], *Bittrex*[11], *Binance*[12], *Bitstamp*[5], and *Coinbase*[13], results in a graph with 401 vertices and 1912 edges. Calculating the liquidity for *bitcoin* in terms of *US Dollars* for this data set takes, on average, 2118 seconds[14] with a standard deviation of 28 seconds.

In addition, the below table contains running time for reduced data sets, with the above (full) data set as the last row. The size of the data sets have been reduced by fetching order books from the same exchanges as for the full data set, but limiting the number of order books fetched per exchange.

|V|	|E|	Mean time[14] (s)	Stddev (s)
26	74	12.98	0.06677
50	146	37.76	0.3350
86	274	109.2	4.763
144	490	227.6	0.403

167	584	309.6	0.823
251	1028	861.1	3.318
401	1912	2118	28.03

Table 4.2.1: Algorithm running times for *BTC* at 50% slippage

For performing the measure for *all* cryptocurrencies, it was necessary to use a 32-core VM in order to complete the task in roughly four hours at 1% slippage. Running time increases roughly linearly with an increase in slippage.

4.3. Improvements

4.3.1 Don't relax all vertices again after removing a path

After removing a shortest path from the graph (and adding it to the result set), the currency implementation relaxes *all* vertices in the graph again. This is not necessary, as only the vertices of the removed path need to be relaxed again. Further research is needed to design an algorithm that only relaxes the minimum number of vertices per iteration of the algorithm.

4.3.2 Stop checking for negative cycles once removed

The first phase of the algorithm removes negative cycles in the graph. Once these have been removed there's no reason to keep checking for them in the next phase of the algorithm.

4.4. Limitations

The order books used to calculate liquidity cannot currently be fetched at the same time. The exchange APIs, from which order books are fetched, impose rate limiting. Given that many exchanges have hundreds of different markets, the first order book fetched from an exchange may reflect how the market looked 10-15 minutes earlier than the last order book fetched from the same exchange.

This can be fixed by exchanges providing an API through which a snapshot — for some specified point in time — of *all* order books can be fetched.

5. Results

5.1. Top 20 cryptocurrency ranking — volume at 1% slippage

The table below shows, for a given cryptocurrency, the USD amount that can be:

1. *Sold into* the market while *decreasing* the price by no more than 1% ("**sell liquidity**" column)

2. *Bought from* the market while *increasing* the price by no more than 1% ("**buy liquidity**" column)

It also shows the sum of the above two figures ("**buy+sell liquidity**" column), and how large buy+sell liquidity is, for each cryptocurrency, in relation to the sum of this figure for *all* the cryptocurrencies analyzed ("**buy+sell part of total**" column).

Cryptocurrency	**Buy liquidity (USD)**	**Sell liquidity (USD)**	**Buy+sell liquidity (USD)**	**Buy+sell part of total (%)**
BTC	7,856,850	6,570,925	14,427,775	18.2
ETH	5,088,901	6,476,265	11,565,166	14.6
EOS	6,623,476	3,735,853	10,359,329	13
USDT	3,145,397	2,527,873	5,673,269	7.14
UST	3,388,687	927,040	4,315,727	5.43
USDC	1,998,350	1,411,262	3,409,612	4.29
LTC	1,426,271	1,236,876	2,663,147	3.35
BNB	874,800	1,710,430	2,585,230	3.25
NEO	936,152	1,227,629	2,163,781	2.72
XRP	898,224	1,226,613	2,124,837	2.67
TUSD	1,186,190	873,300	2,059,490	2.59
PAX	757,808	787,437	1,545,245	1.95
USDS	124,132	962,232	1,086,364	1.37
ETC	432,594	464,753	897,346	1.13
XMR	329,579	540,479	870,057	1.1
BCH	485,909	289,213	775,121	0.976
BCHABC	332,226	364,213	696,439	0.877
BAB	426,860	247,006	673,865	0.848
ADA	332,556	331,880	664,437	0.836
IOT	362,534	282,944	645,479	0.812
Others	5,840,554	4,404,672	10,245,211	12.9
Total	**42,848,050**	**36,598,895**	**79,446,927**	**100**

Table 5.1.1: Buy and sell quantities at 1% slippage (*Apr 18 2019 09:20 UTC*)

The pie chart below (figure 5.1.1) shows the data from the "**Buy+sell part of total**"-column of table 5.1.1 above.

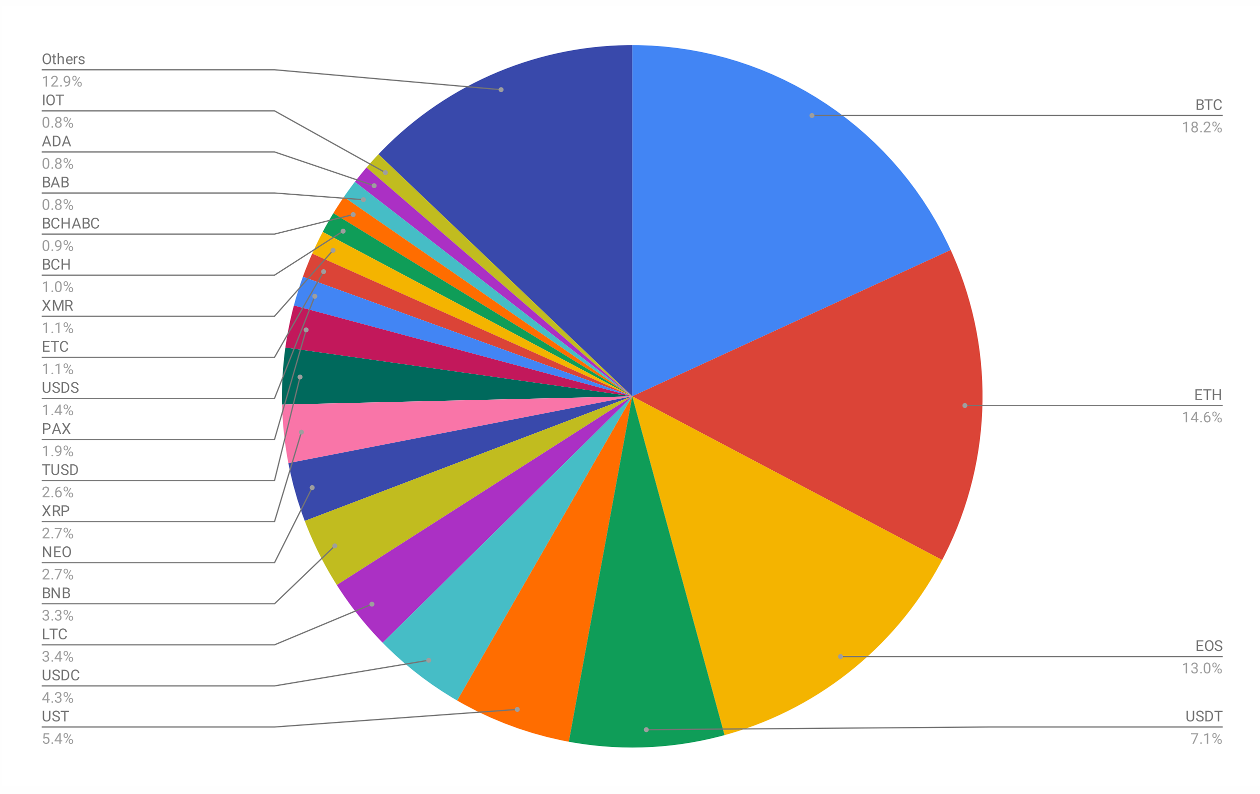

Figure 5.1.1: Buy+sell quantity part of whole at 1% slippage (*Apr 18 2019 09:20 UTC*)

The pie chart below shows the same as Figure 5.1.1, but using data from May 12 2019 10:00 UTC.

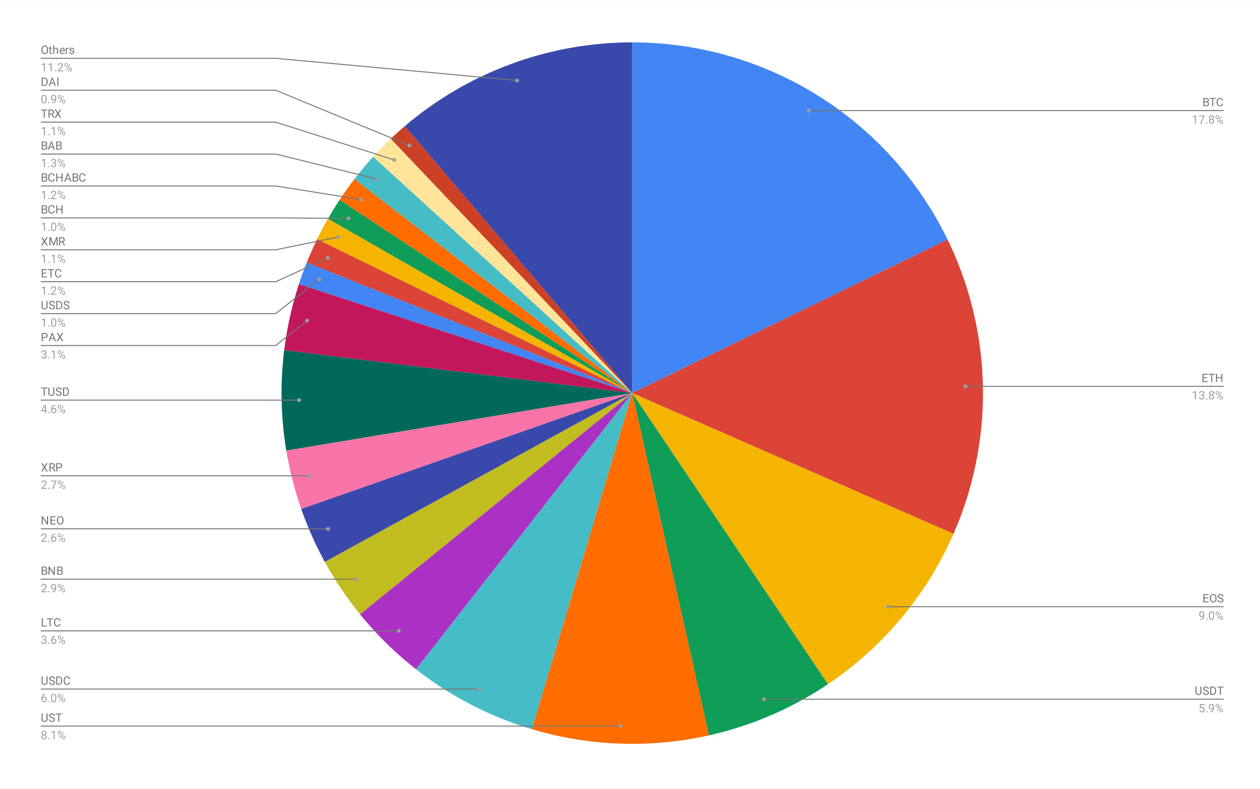

Figure 5.1.2: Buy+sell quantity part of whole at 1% slippage (*May 12 2019 10:00 UTC*)

The top 18 most liquid cryptocurrencies are the same for the Apr 18 and Mar 12 data, which is roughly 24 days apart. Only 19th and 20th place has changed from ADA & IOT to TRX & DAI, respectively.

Comparing the top 18 cryptocurrencies that were in the data for both Apr 18 and Mar 12, the following chart shows the percent change in buy+sell liquidity for the given cryptocurrency these two dates. In other words, this does not look at the change in distribution (as Figure 5.1.2 does), but instead looks the change in the buy+sell liqudity sum for the individual cryptocurrency.

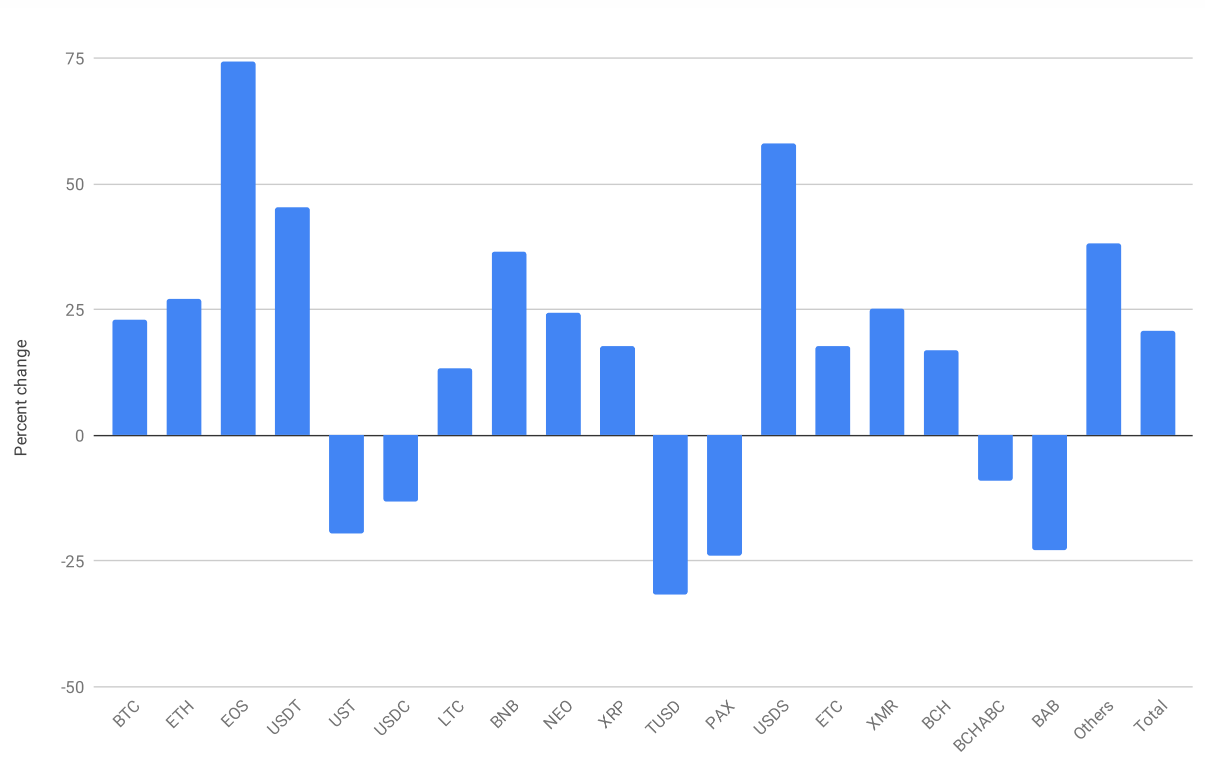

Figure 5.1.3: Change in liquidity between Apr 18 and Mar 12 2019

We see here that the liquidity of all 18 cryptocurrencies except six has increased. Both the liquidity of cryptocurrencies not included in this chart has increased (*Others*) as well as the liquidity of *all* analyzed cryptocurrencies (*Total*).

5.2. Comparison with "market cap" ranking

To compare the market cap measure with the measure presented in this paper, we look at the "percentage of total" figure for both these measures (the same as shown in table 5.1.1).

The table below displays the top 10 cryptocurrencies by market cap [15] and buy+sell liquidity, using data from May 12 2019 10:00 UTC.

Cryptocurrency	**Market cap**	**Sum of buy & sell liquidity at 1% slippage**
Bitcoin	59.3	17.6
Ethereum	9.36	13.6
XRP	6.11	2.71
Bitcoin Cash	2.92	0.997

Litecoin	2.52	3.53
Monero	0.6	1.04
Dash	0.5	0.275
NEM	0.24	0.164
IOTA	0.39	0.232
NEO	0.29	2.61
Others	17.74	57.1

Table 5.2.1: *market capitalization* percentage of total

The data above shows a large discrepancy between the figures for many cryptocurrencies. Notably, the relative value of *NEO*, using the liqudity measure, is 9 times higher than measuring its value using market cap. The relative value of both Bitcoin and Bitcoin Cash value is roughly 3 times greater measured using market cap compared to liquidity.

The pie chart below visualizes the *market cap* [15] part of Table 5.2.1 above.

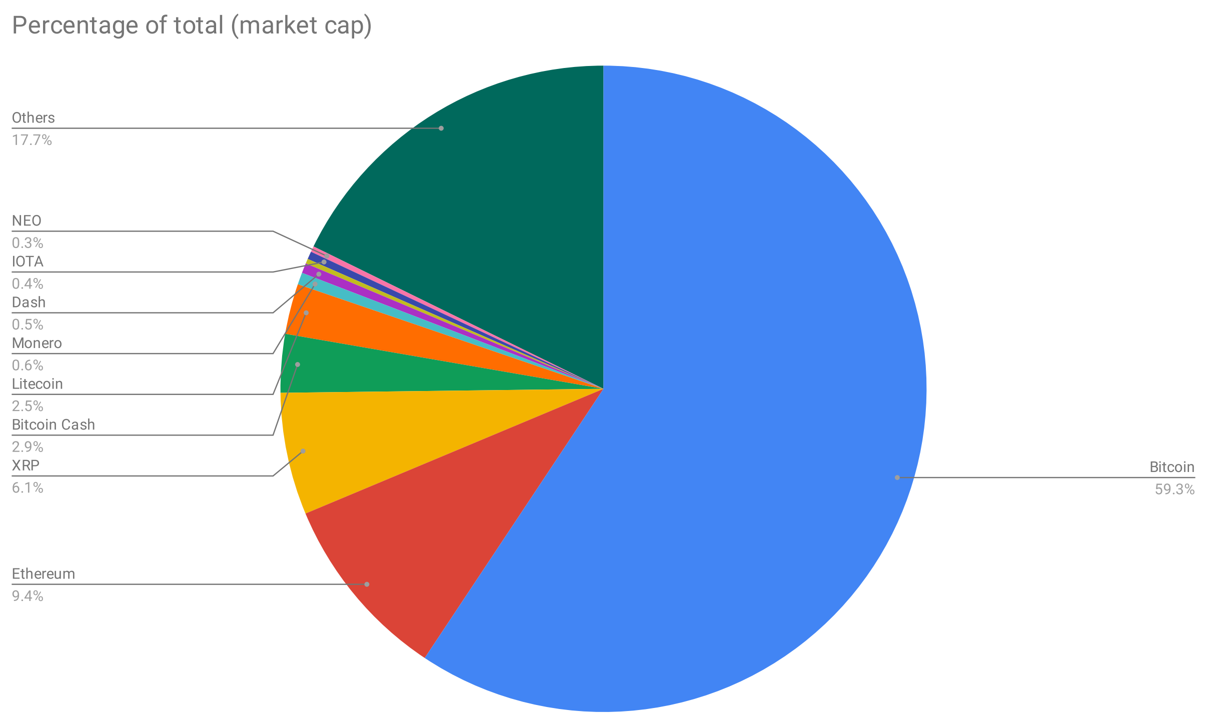

Figure 5.2.1: *market capitalization* percentage of total

The pie chart below visualizes the *liquidity* part of Table 5.2.1.

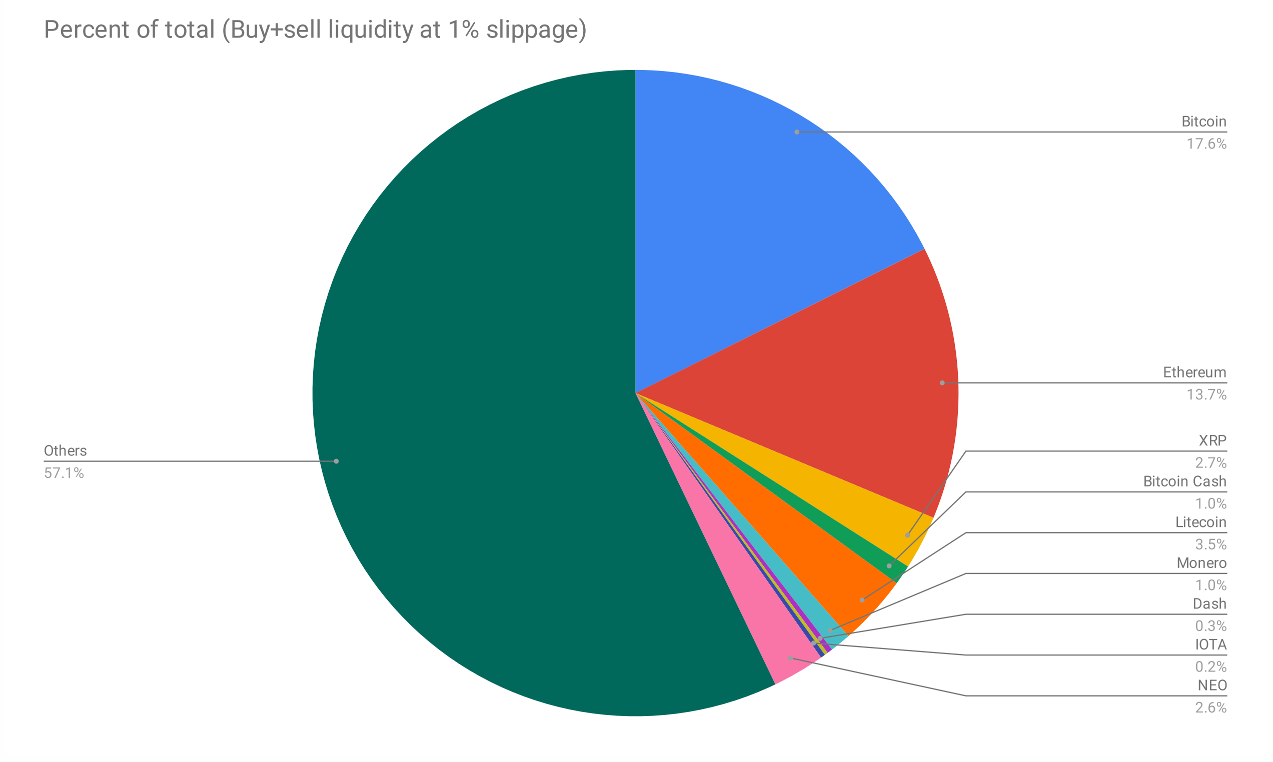

Figure 5.2.2: *buy+sell liquidity at 1% slippage* percentage of total

6. Conclusion

Results show that the relative value of *NEO*, using the liqudity measure, is 9 times higher than measuring its value using market cap. The relative value of both Bitcoin and Bitcoin Cash is roughly 3 times greater measured using market cap compared to liquidity. Also, the liquidity of the majority of the measured cryptocurrencies has increased (20.8% on average) between Apr 18 and May 12 2019.

The Haskell implementation of the algorithm presented in this paper is not fast enough if the number of cryptocurrencies increases by just a factor of 5. In its current form, it's barely fast enough, taking several hours to run on a 32-core VM analyzing roughly 400 cryptocurrencies.

7. References

[1] Sedgewick, R. (2011). Algorithms. 4th ed. Boston [etc.]: Addison-Wesley.

8. Notes

1. Ticker symbol *INTC* on the *Nasdaq* exchange↩

2. https://www.nasdaq.com/symbol/intc↩

3. https://www.nasdaq.com/symbol/intc/stock-report↩

4. https://www.bitfinex.com↩↩

5. https://www.bitstamp.net/↩↩

6. Data from May 14 2019 07:30 UTC↩

7. https://github.com/runeksvendsen/order-graph↩

8. https://github.com/runeksvendsen/bellman-ford↩

9. https://algs4.cs.princeton.edu/44sp/BellmanFordSP.java.html↩

10. Fetched on Apr 18 2019 09:20 UTC ↩

11. https://bittrex.com/↩

12. https://www.binance.com/↩

13. https://www.coinbase.com/↩

14. On a 2.2 GHz Intel Core i7↩↩

15. https://coinmarketcap.com/charts/#dominance-percentage (for May 12 2019 10:00 UTC)↩↩