Developing an R library for Big Data processing in the Field of Cryptocurrencies. An application for the cryptocurrency Bitcoin.

Efstathios Mazarakos

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Economics & Data Analysis

School of Business Administration
Department of Economics

Master of Science
“Applied Economics and Data Analysis”

September 2018
The present dissertation entitled

«Developing an R library for Big Data processing in the Field of Cryptocurrencies. An application for the cryptocurrency Bitcoin.»

was submitted by Efstathios Mazarakos, SID 1018413, in partial fulfillment of the requirements for the degree of Master of Science in «Applied Economics & Data Analysis» at the University of Patras and was approved by the Dissertation Committee Members.
I would like to dedicate my dissertation to my family.
Acknowledgments

I would like to express my sincere gratitude to my dissertation supervisor, Assistant professor Manolis Tzagarakis, whose advice and close guidance helped me complete this dissertation.

I extend my thanks to all members of the teaching faculty for their dedication and hard work for the duration of the master program "Applied Economics and Data Analysis"

Finally I would like to thank my family for supporting me during the time of my studies.
Summary

In cryptocurrencies, all transactions occurring in the network are public and can be retrieved and analyzed by anyone. Although sophisticated libraries for processing cryptocurrencies transactions are available, they present a plethora of problems mainly due to the big volume of the data. Transaction data of several GBs in a short duration of time are common in the field of cryptocurrencies and existing tools provided by the programming language R allow the preprocessing and processing of those data. However these available libraries are inefficient, and cannot cope with the size of the data. In this dissertation we aim to design and develop a library in the programming language R that allows efficient processing of cryptocurrency transaction data and especially those of Bitcoin.

Keywords: Cryptocurrencies, Transactions, Big Data, Library
Περίληψη

Στα κρυπτονομίσματα, όλες οι δοσοληψίες που συντελούνται είναι δημόσιες και μπορούν να προσκομίσουν και να αναλυθούν από οποιονδήποτε. Αν και υπάρχουν εξειδικευμένες βιβλιοθήκες για την επεξεργασία δοσοληψιών των κρυπτονομισμάτων, αυτές παρουσιάζουν аρκετά προβλήματα κυρίως λόγω του μεγάλου μεγέθους των δεδομένων. Δεδομένα δοσοληψιών της τάξεως των GB σε σύντομο χρονικό διάστημα δεν είναι σπάνιες στο πλαίσιο κρυπτονομισμάτων και οι διαθέσιμες βιβλιοθήκες που υπάρχουν στη γλώσσα προγραμματισμού R επιτρέπουν την προεπεξεργασία και επεξεργασία των δεδομένων αυτών. Ωστόσο, οι διαθέσιμες βιβλιοθήκες που υπάρχουν στη R, δύσκολα μπορούν να ανταπεξέλθουν στον μέγεθος των δεδομένων αυτών. Σε αυτή την διπλωματική εργασίας παρουσιάζουμε την ανάπτυξη και σχέδιαση μιας βιβλιοθήκης στη γλώσσα προγραμματισμού R που επιτρέπει την αποδοτική επεξεργασία δεδομένων δοσοληψιών κρυπτονομισμάτων και ειδικότερα του Blockchain.

Λέξεις κλειδί: Κρυπτονομίσματα, Δοσοληψίες, Μεγάλα Δεδομένα, Βιβλιοθήκη
# Contents

1 Introduction ..................................... 1  
   1.1 Bitcoin and Cryptocurrencies ............... 1  
   1.2 Thesis Purpose and Contribution .......... 3  

2 Theoretical Background ......................... 5  
   2.1 A brief overview of the System ............ 5  
   2.2 Transaction Scheme .......................... 6  
      2.2.1 Cryptographic Properties .............. 9  
      2.2.2 Different Transaction Types .......... 10  
      2.2.3 Validation Rules ..................... 10  
   2.3 Blockchain ................................ 12  
      2.3.1 Proof of Work and Mining ............ 12  
      2.3.2 Block Time and Rewards .............. 14  
   2.4 Block Scheme ............................... 14  
   2.5 Existing parsing tools ..................... 20  
   2.6 Blockchain and Big Data ................... 22  
      2.6.1 Big Data And Economics .............. 23  
      2.6.2 Big Data Tools and Techniques ....... 24  

3 Methodology & Data ............................ 28  
   3.1 Data collection ............................ 28  
   3.2 R packages ................................ 29  
   3.3 Blockchain Statistics ...................... 32  
   3.4 Timing Data ................................ 35  
      3.4.1 Code Development .................... 36
CONTENTS

3.4.1.1 Code Version 1.0 .......................... 37
3.4.1.2 Code Version 2.0 .......................... 39
3.4.1.3 Code Version 3.0 .......................... 42

4 Econometric Analysis .......................... 45
  4.1 Linear Models ................................. 45
     4.1.1 OLS Regression Model ....................... 45
     4.1.2 OLS Model Results ......................... 47
     4.1.3 Quantile Regression Model .................. 50
     4.1.4 Quantile Regression Results ............... 52

5 Conclusion ................................. 54

Appendix A Chapter 1 ......................... 57
Appendix B Chapter 2 ......................... 58
References ................................. 58
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The first components of a block, called the Blockheader</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Elements of the second part of a block</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>Tools for manipulating Big Data</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Total number of transactions at the years end</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>Summary Statistics for Code v1</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>Summary Statistics for Code v2</td>
<td>41</td>
</tr>
<tr>
<td>7</td>
<td>Summary Statistics for Code v3</td>
<td>43</td>
</tr>
<tr>
<td>8</td>
<td>Regression Results Table</td>
<td>48</td>
</tr>
<tr>
<td>9</td>
<td>Regression Results with all variables</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>Quantile and OLS regression results</td>
<td>53</td>
</tr>
<tr>
<td>B.1</td>
<td>The results of the profiling R package</td>
<td>59</td>
</tr>
<tr>
<td>B.2</td>
<td>The results of the profiling R package</td>
<td>60</td>
</tr>
</tbody>
</table>
# List of Figures

1. Representation of Bitcoin transaction Scheme ......................... 8
2. Bitcoins in circulation ................................................. 14
3. Representation of the Blockchain .................................... 20
4. The size growth of the Blockchain from 2009 to 2018 ................. 33
5. The per-block number of transactions ................................. 33
6. Fluctuations of the Unspent Transaction Outputs .................... 35
7. Parsing Time vs number of average transactions per block ......... 38
8. Cumulative parsing time vs Number of parsed blocks ............... 39
9. Parsing time vs Number of Inputs/Outputs .......................... 41
10. Scatterplot of parsing time and the total of inputs and outputs ... 43
11. Kernel density of the Residuals from the OLS regression .......... 52
A.1 Simple Code Flowchart ............................................. 57
Chapter 1

Introduction

1.1 Bitcoin and Cryptocurrencies

Bitcoin has hit headlines around the globe recently as the digital currency that could dethrone the traditional financial system and establish a new way of transacting with each other. With an all time high price of almost 20 thousand dollars and total market capitalization around a third of a trillion its existence as a financial product can no longer be ignored, while the technology behind it should provide with many research questions to be answered, in the field of both economics and computer science.

Bitcoin is created from a plethora of concepts and technologies aiming to provide the basis for a digital money ecosystem available to anyone with access to the Internet. Its users have the ability to store and transmit value over a network accessible from a wide range of devices such as smart phones, laptops etc to perform all kinds of transactions already done by traditional currencies, anonymously. Unlike traditional currencies however Bitcoin exists only in the "digital world", with no physical counterpart, while its major difference is the fact that there is no central authority/server maintaining records of transactions and accounts or governing its supply.

Regarding the commercial use of Bitcoin, is in its very premature stage with a tiny proportion of vendors accepting it as a form of payment. As of 2018, the
number of offline merchants accepting Bitcoin is calculated at roughly 12 thousand \(^1\) while adding also the online vendors, the total number reaches around 64 thousand indicating reluctance for both merchants to accept as well as holders of the currency to pay with due to issues of high price volatility, speculation about Bitcoin price increases as well as the sometimes insufferably high transaction fees.\(^2\)

The main problems a digital currency had to overcome in order to be considered secure enough to be utilized (although rarely), in commerce and everyday transfers of value, were that of authenticity of the funds, proof that the money can only be spend once (known as the 'double spend' problem) and the certainty that only the designated recipient has access to the funds. The Bitcoin protocol combined elements from past attempts at a digital currency\(^3\) along with the developments in cryptography to create a completely decentralized electronic cash system that does not rely on a central authority for transaction validation and settlement or currency issuance.

Satoshi Nakamoto in his paper *Bitcoin-A Peer-to-Peer Electronic Cash System* in 2008, first proposed this idea of a decentralized payment system, proposing interesting solutions to the electronic currencies problems, that following the financial crisis of 2009 started to gain traction. Since then and after Nakamoto's disengagement from the project on April 2011, developing and updating the code and network fell to the hands of volunteering programmers across the world. In the following years, and based on the open source software\(^4\) used for Bitcoin, alternative digital currencies were developed to improve upon various technical and practical aspects of bitcoin\(^5\); however, despite the fact that it may be technologically inferior to various alternatives, Bitcoin remains relevant due to its first mover

\(^1\)Information regarding offline merchants were obtained from the website coinmap.org, without certainty about the accuracy of the number.

\(^2\)Transaction fees, explained in more detail later, reached as much as 55 US dollars regardless of transfer amount.

\(^3\)The main ones being 'Hashcash' and 'b-money'

\(^4\)All information and code regarding the operation and technicalities of the Bitcoin network are available at https://github.com/bitcoin/bitcoin.

\(^5\)These currencies called *altcoins* are as of August 2018 around 1800, with the most popular being Ethereum, Ripple, Bitcoin Cash etc.
1.2 Thesis Purpose and Contribution

Bitcoin’s unique architecture allows for all data of every transaction ever performed on the network to be stored on a public ledger freely available for anyone to examine. The drawback however is that these data are not easy neither to decode (as they are maintained in non-human readable format) nor to handle. For this reason special tools are required to access the vast amount of information stored in this public ledger, and since the research topic is rather recent the literature is very scarce on efficient tools able to undertake the task. The software we aim to create in this thesis, would be able to accurately and quickly decode the input data (Bitcoin transactions) and present them in human readable form. Although existing decoders (from now on called parsers) are available in a lot of programming languages, an implementation in R is yet to be created. This thesis aims to fill this research gap by creating a library for R able to perform the transformation of the raw unstructured data to a human readable form.

Additionally we focus on testing the performance of this software implementation in R programming language, using a bottom-up approach starting from some basic utilities all the way to the full parsing procedure of the unstructured raw data from the Bitcoin ledger. The code development part is split into three levels, each improving from the last one by adding more capabilities, while also improving the inefficiencies identified. The main problems we encountered where mainly due to the large volume and unstructured nature of the data. We also had to take into account the structure of the information contained as it maintains a pre-established scheme.

In order to understand the demands of the software in terms of space, memory and speed we use existing R packages to collect performance and timing data for analysis. The timing data of these phases are analyzed to establish the influential variables increasing the parsing time, while also looking for nonlinearities indicat-
ing software problems. We find that the amount of parsing time is influenced both by the number of transactions and the number of inputs and outputs in those transactions. We also seem to be able to "capture" almost all the data variation as suggested by the almost perfect fit of the data, and high adjusted \( R^2 \).

The dissertation is split into 4 sections. In the first part of the thesis we present some background about the theory and operation of the Bitcoin ecosystem, and its contents. We intensely focus on the structural form and rules that maintain the sub-elements in working order, while also analyzing the details of each of those elements. As the point of the thesis is creating a software able to "decode" the Bitcoin’s raw data, mapping its scheme and understanding the rules it follows is essential. We then present some tools and techniques to handle the vast amount of the so called Big Data while also making the case about their usage in modern economic research. The second part of the thesis includes the R packages we used during the development of our R code, as well as the steps, problems and different approaches we used to create it. Furthermore summary statistics relevant to the development and examination of the software are presented. For our analysis of the performance of the created code in the next section, gathered timing data were used, giving us a clear picture about the influential factors, and the existing problems. Finally we conclude by presenting the three different code versions in the appendices.
Chapter 2

Theoretical Background

2.1 A brief overview of the System

Bitcoin is a complex scheme, and its implementation involves a combination of cryptography, distributed algorithms, and incentive driven behaviour. In essence, it is an electronic token backed by no underlying commodity or sovereign currency, and is not a liability on any balance sheet. As a result it has no intrinsic value, other than the ability to perform transactions within its network. The value of Bitcoin is derived by its use for making payments in the Bitcoin system, and from the purpose of accruing gains from its possible appreciation (Anton Badev, Matthew Chen, 2014).

The disruptive innovation operating behind the scenes in Bitcoin is what is called the 'Blockchain'. Essentially, the Blockchain is a public ledger keeping track of all past transactions in chronological order while continuously growing to include any new ones. As the system’s goal is decentralization and avoidance of a central authority/server, a copy of the Blockchain is freely distributed to every participant of the network. This creates the issue known in distributed computing as the 'Byzantine Generals’ Problem (Lamport L. et al, 1982) solved by Satoshi with another major innovation called the 'Proof of Work’ algorithm, which allows

---

6 The Byzantine Generals’ Problem refers to a situation in distributed computing where independent participants of a network try to agree over the state of the system, in this case the Blockchain, by exchanging information over an unreliable and potentially compromised network.
the network to reach consensus over the valid current history of transactions. This algorithm secures the network against possible attacks while also providing the participants with economic incentives to maintain and update this ledger. The network participants that "work" to record transactions and secure the blockchain are called "miners" and are rewarded with newly minted bitcoin increasing thus their total supply, and alleviating deflationary pressure.

The Blockchain consists of chunks of data called "blocks". The main contents of a block on the other hand are its transactions. They are stored in the Blockchain with a past referencing scheme that links old with new ones together through an input-output system going back to the first ever created block called the "genesis block". However before being stored, the network enforces that they have gone through a validation process and are considered structurally correct. It is very important to ensure that each element the Blockchain is constructed from, maintains a predefined and agreed upon structure as to avoid conflicting versions of the same software existing and been shared on the network. As the parsing software we are trying to develop is based on these predefined schemes for each sub element of the Blockchain, we considered it very useful to present this structure and explain along the way their "behind the scenes operations".

2.2 Transaction Scheme

We start our analysis of the Blockchain sub elements and their structure by the transactions scheme. Transactions are the most essential part of the Bitcoin ecosystem. We consider as transactions those data structures that encode the transfer of value between Bitcoin system participants (Antwnopoulos A., 2017). All transactions, after validation, are included in a public ledger called the "Blockchain", however the way they are preserved in this system is not easily understandable and consequently acquiring and processing these data, requires knowledge of the characteristics and components that define them.

Their main building blocks are their Inputs and Outputs. Inputs roughly trans-
late to the amount of value a given user currently holds, while outputs specify the
receiver and the transfer amount. However, since no central authority maintains
users’ balances, inputs work by referencing past transactions’ outputs where the
now payer was the receiver of funds, and thus has the ability to spend them. This
procedure goes back the very first block, creating this back linked transaction
chain. Contrary to traditional electronic payments where the id and balance of
the sender and recipient are readily available, a Bitcoin transaction must reference
the past to perform any new transfers of funds.

Below we present the exact structure of any given transaction:

• **A list of outputs** (TxOut), with information about the amount of BTC
  transferred in satoshis (smallest BTC denomination) and a cryptographic
  **puzzle** also known as "locking script" that sets the required conditions for
  the amount to be spent.

• **A list of inputs** (TxIn) that reference one or more previous **unspent** trans-
  action outputs, a **digital signature** that satisfies the required conditions of
  that output making it spendable and for each input the variable **Sequence
  Number**.

• The variable **Locktime** which indicates the earliest time or earliest block
  when that transaction may be added to the public ledger (Blockchain).
2.2 Transaction Scheme

**Figure 1:** Representation of Bitcoin transaction Scheme

*Source:* Blockchain.com

*Notes:* This figure depicts the sub elements of a transaction as well as the way that they interact.

In order to transfer funds between two parties, the sender creates a transaction where in the Inputs section a previous unspent output is referenced with funds greater to or equal to the desired amount to be transferred. He then "signs" the transaction by providing his digital signature, proving ownership of these funds. Multiple inputs may be aggregated to match the output amount, much like using five 1Euro coins to pay 5Euro worth of something.

In the Output section, the receiver is specified in the "locking script" along with the transaction amount. Again in case of multiple recipients, a list of outputs is created, with each one specifying the receiver and the transfer amount. After the transaction is properly created it is transmitted to the network, where a node (a Bitcoin network participant with access to the Internet) validates it based on certain rules and then adds it to the next block to be mined. The creator of a transaction is able to delay its submission to the next block, by providing the exact desired Block height or exact time for the transaction to be added, by using the variable of Locktime.

One important exemption to this input output system is a special kind of
transaction called a "coinbase". This is always the first transaction included in a
block, has no inputs and only one output and is the reward obtained by a miner
after updating the Blockchain. Its details are explained later.

2.2.1 Cryptographic Properties

The only prerequisite for someone to transact using Bitcoin is to create a pair
of digital keys. The Public and Private key pair, long existing topic in the cryp-
tography literature, acts as the user’s unique identity in the network and through
them he can receive and send funds. The public key is shared with anyone wanting
to send you funds and acts as the recipient’s name in a check ("Pay in the order
of"), while the private key (known only to its owner), proves ownership of an amount
and is required to authorize a transfer of funds to a third party. The cryptographic
puzzle and digital signatures incorporated in a transaction are created using a
scripting language unique to Bitcoin and take advantage of public key cryptogra-
phy. Although the technical details of this language are besides the scope of this
thesis, they require clarification as they are an essential part of the transaction
validation process.

In detail the cryptographic puzzle created in a TxOut, places a spending con-
dition, in other words 'locks' the transaction amount so that only the owner of
a specified (in the same script) Public Key may spent it. The 'digital signature'
script on the other hand, 'unlocks' the output amount when presented with proof
of ownership of that prespecified Public key, namely the corresponding Private
key available only to the recipient. Further useful cryptographic features of a valid
'digital signature' are that it gives a recipient reason to believe that the transac-
tion was created by a known sender (authentication), that the sender cannot deny
having sent the transaction (non-repudiation), and that the transaction details
were not altered in transit (integrity).

The aim of this complicated process is to ensure that only the intended recip-
2.2 Transaction Scheme

ient has access to the funds, the immutability of the transactions details and the inability to deny its creation.

2.2.2 Different Transaction Types

Bitcoin’s scripting language, although not very sophisticated, enables the users to create different types of transactions ranging in details such as the number of recipients required to unlock the funds. Their objective is to offer more flexibility in the transfer of funds between the users. These transactions are very important to consider when creating a Bitcoin parser as each transaction type is represented uniquely in the data. These are:

- Pay-to-Pubkey
- Pay-to-PubkeyHash
- Pay-to-Script-Hash
- Multisignature Script

The main differences in the data representation of these transactions are found in their "locking and unlocking scripts". For example a Pay-to-Public-key transaction requires the locking script to include the Public Key instead of its hashed value as in the Pay-to-Public Key hash. A multisignature transaction on the other hand, is more complex and requires multiple "digital signatures" to unlock the funds. The software we try to create should take into account these different types and be able to distinguish one from the other.

2.2.3 Validation Rules

In order for a transaction to be considered valid and be propagated in the Bitcoin network, certain rules regarding their structure need to be satisfied. The first node to receive a transaction is responsible to verify its validity by following certain
steps and reject any wrongly formulated or malicious attempts.

The first check is about whether the previous outputs referenced in the inputs section exist and have not been spent. This is done by searching and confirming that the said output is included in the Unspent Transaction Outputs Cache (UTXO) where all such transactions are stored.

Secondly, the node checks that the sum of the value of the inputs is equal to or greater than the sum of output values, to ensure that the sender does not spend more Bitcoin than he owns. It is generally the case that the input amount is slightly greater than the output, with the difference called the Transaction Fee. This fee is collected by the miners that include the valid transaction in the next block and is paid voluntarily by the issuer to ensure prioritization of their transaction over others. More information about the miners is given later in the text.

Finally, it checks that the "digital signatures" used to unlock each Input are valid and actually correspond to the referenced Public keys.

After this procedure is completed, the transaction is considered structurally correct by the node and joins a queue to be added in the next block and propagated across the network.

As previously mentioned, the input/output system requires that the referenced previous outputs to be unspent. Consequently, each validating node must be aware at any time whether a transaction's Inputs are actually spent or not. This is accomplished by maintaining a cache, a database, that holds all the unspent transaction outputs (UTXO). This cache is consulted for each input of an under validation transaction. If all inputs are matched correctly in this cache, the validation succeeds and the matched inputs are removed from the database. A thing to take into account is that this database in not stored anywhere in the network itself and as a result, needs to be created and constantly be kept updated externally by each node. This procedure is highly memory and processing power hungry and its implementation on a regular off-the-self computer could be very hard and time consuming. In detail as of 04/08/2018 the number of Unspent Transaction
2.3 Blockchain

Outputs is around 49 million, which amounts to 2.6 gigabytes of memory for the UTXO cache. However, its creation and maintenance is essential as it offers part of a solution to a long-existent problem of electronic currencies, that of "Double spending". With a valid transaction added to a block, the recipient can be certain that the funds transferred to him, are spent only once.

2.3 Blockchain

The disruptive technology behind the operation of Bitcoin's network is called the "Blockchain", and can be described as a database of all past transactions that is not maintained in a centralized server, rather is distributed peer-to-peer and secured by the network participants. Recorded transactions are organized in chunks named "blocks", and are after validation added to the "Blockchain" sequentially through a process called "mining". Each new block added contains a list of new transactions as well as a reference to the previous block in the chain, thus creating a back linked structure, where both new and past transactions are saved. This ever-growing database cannot be modified and is secured by methods of cryptography. The major cryptographic tool that is used extensively in the Blockchain, is the hash function.

2.3.1 Proof of Work and Mining

The way that participants ensure the security of the Bitcoin network is through the process called "mining". By mining we consider the provision of computational power to the network by individuals or groups called miners that "work" to ensure the validity and current state of the Blockchain and its contents. This sum of com-

---

7Double spending refers to the problem where the same single digital token (a Bitcoin) can be spent more than once. This is possible because a digital token consists of a digital file that can be duplicated or falsified. As with counterfeit money, such double-spending leads to inflation by creating a new amount of fraudulent currency that did not previously exist. This devalues the currency relative to other monetary units, and diminishes user trust as well as the circulation and retention of the currency.
putational power plays the role of the central authority in a traditional system (like a bank) thus eliminating the need for one (decentralization). Since the Blockchain is distributed and maintained by each and every node of the network there must always be consensus regarding its state. The protocol dictates that in order for an update to occur significant computational power must be expended. Then the nodes perform a kind of election to ensure that the valid blockchain is that with the most computational power. After providing with proof of their contribution miners are rewarded with a specified amount of BTC. The algorithm that the Bitcoin network uses as proof of work is the **partial hash inversion**. In non-cryptographic terms, miners working for the network have to solve an extremely hard mathematical problem where the most efficient solution is brute force. After the solution is found they relay it to the network and are rewarded with newly generated Bitcoin. This reward besides ensuring alignment of interests between miners and network security, also increases the supply of circulating currency alleviating deflationary pressure.

In more practical terms the mining system works like this: When a transaction is validated it is stored locally in the node that first received it, in a database called the **unconfirmed transaction memory pool**. After that a mining node creates a block by grouping unconfirmed transactions, and tries to solve a cryptographic problem as proposed by Bitcoin’s "Proof of Work" algorithm, competing with the other nodes to find the solution. The first to solve the hash inversion problem gets to add his block in the blockchain and obtains the 'block reward' through the 'coinbase' transaction along with any potential transaction fees. After the block is validated and added to the chain, all its transactions get a confirmation and the process starts anew. The time required to mine a block although in part random (brute force), also depends on the computational power of the node.

---

8Using the contents of the newly formulated block, miners must find a random value, later called "nonce", that when used as input along with that new block in the cryptographic function SHA256, the result must fall below a threshold as set by the "difficulty" of the puzzle.
2.3.2 Block Time and Rewards

The protocol that every node abides by, enforces the average time between mined blocks to be 10 minutes. This is achieved by adjusting the cryptographic puzzle’s difficulty every 2016 blocks or roughly 2 weeks, trailing the changes of computational power. The block reward or in other words the rate that the Bitcoin supply increases, is designed to diminish with time. This design choice is achieved by halving the block reward approximately every four years or 210000 blocks. It started at 50 bitcoin and has since then been halved twice to a current 12.5 reward. This exponentially diminishing increase in the supply of BTC reaches its maximum in 2140 where all bitcoin will have been mined and no new coins will be issued.

Figure 2: Bitcoins in circulation

Source: blockchain.info

Notes: This figure depicts the way Bitcoin’s supply diminishes and reaches a maximum by 2140.

2.4 Block Scheme

As the Blockchain is a collection of blocks, each 'stacked' on top of another, it is important to first describe the contents of a block, their meaning and how they are digitally represented in the raw data. The components of any given block can
be decomposed in two main parts. The first contains information about the block itself called the Blockheader, while the remainder consists of transaction information. The following tables give a short description of the sub elements included in the two parts.
### 2.4 Block Scheme

**Table 1. The first components of a block, called the Blockheader**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Size in Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagicID</td>
<td>4</td>
<td>Identifier for the beginning of a Block</td>
</tr>
<tr>
<td>Block Length</td>
<td>4</td>
<td>The size of the Block</td>
</tr>
<tr>
<td>Version Number</td>
<td>4</td>
<td>Always set to 1</td>
</tr>
<tr>
<td>Previous Block Hash</td>
<td>32</td>
<td>32 byte hash of the previous Block</td>
</tr>
<tr>
<td>Merkle Root</td>
<td>32</td>
<td>A hash made from the Block’s transactions</td>
</tr>
<tr>
<td>Timestamp</td>
<td>4</td>
<td>Time the Block was created</td>
</tr>
<tr>
<td>Target Difficulty</td>
<td>4</td>
<td>Target difficulty for this Block</td>
</tr>
<tr>
<td>Nonce</td>
<td>4</td>
<td>Random number used as part of the mining process</td>
</tr>
</tbody>
</table>

The first part of a block called a "Blockheader" containing information about the block.

The MagicID is a series of 4 bytes, always '0xD9B4BEF9', that represent the beginning a new block.

**Block length**, as the name suggests, represents the size of the current block in bytes. The current maximum size is decided to be 1 megabyte. This fact greatly influences the speed that transactions are processed since a 1mb block per 10 minutes can only support around 7 transactions per second. Network congestion in times of increased demand is not rare, with repercussions both in terms of increased fees and significant delays in the confirmation time. This creates fears regarding the scalability of the Bitcoin network as there is no way in its current form to support global commerce of potentially two hundred thousand transactions per second.

The **Version number** of a block is always set to 1 and has no further significance.

The next 32 bytes part of the blockheader is the **previous Block Hash**. As we have already discussed the Blockchain is a back linked database where each new block sequentially added, contains some information about its predecessor. These information are in the form of a Block hash, a unique way to identify a Block that can be obtained by hashing twice the blockheader (in other words the block’s first 80 bytes), through the cryptographic algorithm SHA256. An important thing to consider when creating the parsing code is that the current block’s hash is not
included in the block itself and must be externally created. The main utility of including the previous block hash in a new block is that it creates an immutable link all the way back to the first ever block (the "genesis block") that ensures the history and sequence of blocks. Furthermore, a validating node trying to link a newly found block firstly checks its previous block hash field to ensure they match and proceed to extend the chain.

**Merkle Root** is the next 32 byte part of the blockheader. Its main function is to shortly summarize all transactions contained in a given block, and making the process of accessing a certain transaction in a block efficient. As the Merkle root is an optimization feature used by the nodes in the Network, it is not significant for the parsing process and can be omitted.

The next 4 bytes represent the **Timestamp** of a block or the time it was created. Although it the only time variable included in the blockchain, the information it offers can be misleading since the time of a block does not always coincide with that of the transactions it contains. Many fee-less transactions are added in a block maybe even hours after their creation thus creating a mismatch between the time they were created and the timestamp of the block that included them. The format of the timestamp is an integer representing a Unix timestamp. This integer is the time elapsed in seconds from 01-01-1970, a date known as 'the Epoch'. This is important to consider when creating the parser as proper adjustments need to be made.

The final 8 bytes of the blockheader are the **target difficulty** and the **nonce** (4 each), both items related to the mining procedure. These values are insignificant and are not required for parsing or transaction interpretation.
Following the 80 bytes of the blockheader are information about the transactions contained in the block. As already stated these information are not to exceed the 1mb block size limit and are structured as below:

### Table 2. Elements of the second part of a block

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Size in Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Count</td>
<td>1,3,5 or 9</td>
<td>Number of Transactions Included</td>
</tr>
<tr>
<td>TransactionVersionNumber*</td>
<td>4</td>
<td>Expected to be 1</td>
</tr>
<tr>
<td>InputCount*</td>
<td>Variable</td>
<td>Number of Inputs for a Transaction</td>
</tr>
<tr>
<td>Transaction Hash**</td>
<td>32</td>
<td>Hash of the Referred previous TxOut</td>
</tr>
<tr>
<td>TransactionIndex**</td>
<td>4</td>
<td>Index to a specific previous TxOut</td>
</tr>
<tr>
<td>InputScriptLength**</td>
<td>Variable</td>
<td>Length of the Script in bytes</td>
</tr>
<tr>
<td>RawInputScript**</td>
<td>InputScriptLength</td>
<td>Raw byte code for the Input Script</td>
</tr>
<tr>
<td>SequenceNumber**</td>
<td>4</td>
<td>Always 0xFFFFFFFF</td>
</tr>
<tr>
<td>OutputCount*</td>
<td>Variable</td>
<td>Number of Outputs for a Transaction</td>
</tr>
<tr>
<td>OutputAmount**</td>
<td>8</td>
<td>The value transferred in 'Satoshis'</td>
</tr>
<tr>
<td>OutputScriptLength**</td>
<td>Variable</td>
<td>Length of the Script in bytes</td>
</tr>
<tr>
<td>RawOutputScript**</td>
<td>OutputScriptLength</td>
<td>Raw byte code for the Output Script</td>
</tr>
<tr>
<td>TransactionLockTime*</td>
<td>4</td>
<td>Always set to zero</td>
</tr>
</tbody>
</table>

Notes: The second part of a block, containing information about the transactions included.
* One star represents the elements that repeat for each transaction.
** Two stars represent the elements that repeat for each Input or Output in a Transaction.

The second part of the block begins with the actual count of the included transactions. Satoshi opted for a variable length integer to represent that number, meaning that the number of bytes used for the transaction count part, varies with the actual count between 1, 3, 5 or 9 bytes. This detail although small is crucial to take into account when creating the parser as it could result in errors in the
2.4 Block Scheme

byte interpretation.

The **version number** is always to be 1 or 2. It does not contain relevant information about the process besides being in every transaction.

The **number of inputs** as well as the **number of outputs** are included in each transaction and are also represented in the data with a variable integer as the transaction count above. These numbers are important for the parser to know beforehand as it determines how many bytes of data need to be read for parsing inputs or outputs.

The **previous transaction hash** along with the **transaction index** that follow are the most crucial elements for the back-linked nature of the blockchain. What these 32+4 bytes do, is to uniquely identify a previous unspent output, so that a new transaction can point to it and use it as an input. The important thing to consider here is that these hashes are not stored in the Blockchain and must be therefore computed and stored externally by the parser. Furthermore coinbase transactions need to be taken into account as they have special characteristics in the way they are structured.

Following the previous transaction hash are the raw data of the **Input Script**. Varied in length, these data are essentially useless for the parsing process, as no information included in these scripts can help parse the blockchain transactions. However, they need to be identified and then ignored. These are the data (along with the output script data) that the validation process uses to decide whether a transaction is correct or not. The input script usually contains the digital signature and public key of the sender that when "combined" through Bitcoin’s scripting language, unlock the referenced funds for spending.

Ending the input part of a transaction is the **sequence number** always set to 0xFFFFFFFF. After that the output part follows starting with the **output count** explained earlier.

Each output contains the **Amount** to be transferred in 8 bytes. These bytes represent the Bitcoin amount in Satoshis (100 millionth of a Bitcoin) the smallest
2.5 Existing parsing tools

denomination of the currency.

Next comes the output script length which is the size in bytes of the following output script. The output script contains the recipient’s Public key and is very important for the whole parsing process that it is found and stored. Usually the forms that the public key is represented in the output script are:

- As a 67-byte long output script containing a full 65 byte public key address.
- As a 25-byte long output script containing a 20 byte hashed public key address.

Although there are errors in some scripts that a parser should account for, these are the ways that the majority uses to represent a recipient’s Public key.

Finally the last piece of information included in each transaction is the Lock-time that is always set to 0.

Figure 3: Representation of the Blockchain

Source: Bitcoin.org

Notes: This figure depicts the way the Blockchain is structured and linked.
of parsers exist for other programming environments such as Python 2, Python 3, PHP and C++, while in R the only available Blockchain related packages are for querying external sources for Bitcoin related information such as price, volume etc. Some of the already implemented parses have significant problems with either the time required, or the fact that they rely on external sources to acquire the data and not the Blockchain itself. For example, an implementation on PHP programming language, found at https://github.com/bitcont/bitcoin, uses a blockchain.info API to retrieve information about addresses and transactions in a slow procedure that does not involve the public raw data of the Blockchain in any way.

In R, the only available package is called “Rbitcoin” by Jan Gorecki. The main objective of this package is to provide some Bitcoin related Utilities allowing the user to extract price information, volume of transactions etc from websites operating as exchanges. The open source software is found at the repository https://cran.r-project.org/web/packages/Rbitcoin/. No other packages are available in R related to Blockchain. The developed in this thesis code is the first try at a full Blockchain parser in R and as a result should cover this research gap.

In others programming languages like python however, there exist fully operating parses that work quite well and provide with all the information included in the Blockchain. One prime example is the software created by Github user 'Alecalve', in Python 3, with advances features like detecting output types, addresses in outputs, interpreting scripts etc. The github repository can be found at https://github.com/alecalve/python-bitcoin-blockchain-parser.

Another blockchain parser is available for C++ by Github user 'bitcoinjs' at https://github.com/bitcoinjs/fast-dat-parser/blob/master/LICENSE, which uses the full system’s capabilities to export the blockchain data as fast as possible.
2.6 Blockchain and Big Data

When dealing with Blockchain data, either as a node verifying transactions or as a researcher trying to "decode" and interpret them the major problem that hinders progress is their already vast but constantly increasing size and unstructured form, typical characteristics of Big Data. Although what constitutes as 'Big Data' is not set in stone in the bibliography, Laney (2001), Amir Gandomi, Murtaza Haider (2014), Chen, Chiang, & Storey, (2012); Kwon, Lee, & Shin, (2014) suggest that contemporary research focuses on the three V’s as the common analytical framework of Big Data. These represent Volume, Velocity and Variety. A definition by the TechAmerica Foundation’s Federal Big Data Commission (2012) considers Big Data as 'a term that describes large volumes of high velocity, complex and variable data that require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information.'

*Volume* captures the aspect of magnitude. Data sizes that are considered to fall on the 'Big' spectrum tend to reach terabytes (1024 gigabytes) or even petabytes (1024 terabytes). These sizes besides requiring great storage capacity also tend to be extremely tough to process efficiently with current tools and computational power. However time and ever-advancing technologies may deem these numbers obsolete in the near future.

*Velocity* captures the rate data are generated as well as the speed required for analysis. IBM estimations of daily data generation has been as high as 2.6 exabytes in 2016. High frequency data generation, as in the case of Bitcoin, is a Big Data characteristic that creates a need for real time moderation and analysis of inflows. Velocity also describes the way different databases using streaming data interact with each other in real time. Bitcoin nodes are familiar with this notion as they maintain several different databases that interact in real-time to validate transactions and blocks.

Finally *Variety* refers to the structured and unstructured nature of the data
either machine or human created. The endless channels businesses acquire data from nowadays, results in various data formats in need of special analysis and techniques. Variety is the characteristic best describing that fact, and is mainly about the different classes of available data and how to handle them.

The 3Vs has been in recent years extended to 5Vs (Bello-Orgaz et al., 2016), also including dimensions for Value (ability to process the data to extract valuable information or Big Data Analytics) as well as Veracity (a characteristic related with proper data governance and privacy concerns.)

Whatever the definition however, Big Data are sure to transform the current landscape of socio-economic policy and research (Einav and Levin, 2014; Varian, 2014) as well as the fields of economics, business management and decision-making.

### 2.6.1 Big Data And Economics

For an empirical researcher in the field of economics, one of the major issues hindering progress (or outright stopping it), is the availability of data. This situation seems to slowly but gradually reverse as the Big data revolution has changed the way economists access data resources both in terms of quantity as well as quality. A prime example of a rich resource for economists, is the public sector, who through digitalization and the usage of Internet services is able to maintain highly detailed data on individuals and corporations, while minimizing sample selection and attrition problems (D. Card, 2010). The correct treatment of these data, through a proper Big Data architecture could yield invaluable information in the fields of public finance, labor economics, health and education. The works of Piketty and Saez on income and wealth shares using administrative tax data, of R. Chetty on regional differences in economic mobility, S. G. Rivkin on the performance of public school teachers and J. M. Abowd on differences in wages and productivity across similar firms, highlight the importance of cooperation between economists and government-created Big Data.
2.6 Blockchain and Big Data

Another potential contributor to empirical research could be private companies. The unlimited ways modern companies acquire data not only provide an abundant resource of information but also open up the possibility for new areas of research where data were previously impossible to acquire. For example, personal communications, social networks, search and information gathering, and geolocation data can provide economists tools to empirically assess the role of social connections and geographic proximity in shaping preferences, information transmission, consumer purchasing behavior, productivity, and job search. Private sector companies can also help to track economic activity by providing aggregate statistics of their business and customers ahead of the government official announcements as their useful complements, or as a main source of the country’s economic situation where public reports are either scarce or manipulated.

However the implementation of Big Data in economic research is not without its challenges. Most important of all are the issues of privacy and confidentiality affecting the ease of access to the available data. Governments and companies are reluctant to grant full access to sensitive information, requiring confidentiality agreements in the best scenario or denying access all together to all but the most prominent of research teams. Privacy is also a major obstacle in the free flow of data, as information regarding an individuals’ health, location, electricity use, or online activity may be easily obtainable, raising concerns (O Tene, J Polonetsky, 2011) about its potential mismanage and exploitation.

2.6.2 Big Data Tools and Techniques

The approach that applied econometricians follow to answer a research question, given a dataset, is usually by first estimating a model and then focusing on a coefficient of interest representing a causal effect created by a policy, decision etc. Their main concern is to carefully create the standard errors of the parameters of inter-

---

9 Already the payroll service company ADP publishes monthly employment statistics in advance of the Bureau of Labor Statistics, MasterCard makes available retail sales numbers, and Zillow generates house price indices at the county level USA.
est, as to avoid problems such as heterogeneity, endogeneity etc and finally assess the robustness of their findings by estimating a variety of alternative specifications and running placebo regressions to see if the preferred model generates any false-positive findings. This approach comes in high contrast to some of the data methods used in statistics and computer science where the primary goal is predictive fit, especially out-of-sample fit, as well as the use of data driven model selection that identifies the most impactful features. These data mining tools include classification and regression trees, lasso and methods to estimate sparse models, boosting, model averaging, and cross-validation.

Differences in the goals of these approaches may explain the reasons that despite the availability of Big Data, empirical economists remain ‘faithful’ to old econometrics techniques and tools and will continue to do so. However as explained by both Varian and Liran collaboration of these two approaches could provide solutions to econometricians problems such as identification of important attributes of the data (A. Belloni et al., 2012), interaction effects (R. A. Lewis, J. M. Rao, 2014), counterfactual creation for measuring policy treatment effects and the ability to not only measure the average policy effect, but also an individuals sensitivity to policy changes. However not only economics and its empirical researchers are to gain by this collaboration.

Economic theory could be very useful as it can potentially provide with a simple framework of analysis for complex big datasets, where the abundance of relevant and irrelevant information hinder the researcher’s ability to find links between the key variables. As data sets become richer and more complex and it is difficult to simply look at the data and visually identify patterns, it becomes increasingly valuable to have reduced models to frame one’s thinking about what variables to create, their possible relationships or what useful hypotheses to test and experiments to run.

The already existing statistical tools and techniques used by econometricians, such as regression, are almost always sufficient to handle the data, but unique
issues arise when trying to work with huge datasets. Different and more powerful tools may be required in that case, in order to manipulate and analyze the data. The most common issues regarding the use of big data can be summarized as:

1. **Storage and Transport Issues**: The quantity of data available is largely dependent on the current storage technologies. The latest developments of data generation however largely outpaced new storage mediums, resulting in a mismatch between the two. Solution to this problem is offered by the distributed file systems, such as Google File system developed by Google, that use multi-thousands-computers to store huge data files across those computers. Transportation of these data is also a major concern as current communication networks can easily be overwhelmed (transferring an Exabyte of data with a network speed of 1Gb per second requires 2800 hours). A solution to this problem could be the processing of the data in 'the source' and transmitting only the research results(). Secondly the data can be sampled and only the critical for the analysis are transmitted.

2. **Management Issues**: Managing of endless amounts of data between multiple entities creates several issues such as those of access, metadata availability, utilization, updating, governance, and references. Given the volume, it is impractical to validate every data item for fulfilling certain conditions, and as long as no universally accepted way to store raw data, their reduced forms, and the code and parameter choices that produced them, a solution to managing them remains a gap in the research literature that needs to be filled (JASON, 2008).

3. **Processing Issues**: As the volume of data to be processed increases so does the time required. An exabyte of data needed to be processed in its entirety would require roughly 635 years end to end processing time by a 5ghz processor. Thus, effective analysis of exabytes of data would require extensive parallel processing and new analytics algorithms so as to provide useful information on time. These are also propriety to Google in the form of the software MapReduce, but open source tools (Hadoop) are also available.


2.6 Blockchain and Big Data

Table 3. Tools for manipulating Big Data

<table>
<thead>
<tr>
<th>Google name</th>
<th>Open Source Analog</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google File System</td>
<td>Hadoop File System</td>
<td>A system to support large file by distribution across hundreds or even thousands of computers.</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Hadoop</td>
<td>This is a system for accessing and manipulating data in large data structures such as Bigtables by utilizing Parallel processing capabilities of thousands of computers.</td>
</tr>
<tr>
<td>Sawzall</td>
<td>Pig</td>
<td>This is a language for creating MapReduce jobs.</td>
</tr>
<tr>
<td>Dremel, BigQuery</td>
<td>Hive, Drill, Impala</td>
<td>A form of SQL for performing fast queries.</td>
</tr>
</tbody>
</table>

Notes: Big Data tools propriety of Google in the left and their open source adaptations on the right.

Table 4 gives us a better overview of the available Big Data tools owned by Google while also providing the respective open source software that are created based upon them.
Chapter 3

Methodology & Data

3.1 Data collection

The first thing to consider before starting developing any code, is the availability and structure of the data. Luckily the Blockchain data needed as input for the parsing software are freely available online at bitcoin.org. The procedure to get the proper files is to download and install the open source software Bitcoin Core v0.16 or "Satoshi Client" as it is called, available for all operating systems. By running this client the computer becomes a full-node participating in the Bitcoin network, with access to all available information that a node has including the current consensus of the valid Blockchain. Furthermore as we continue to operate in the network all transaction data will continuously update in real time. Since the aim of this dissertation is mainly to create a software to decode the raw data, any real time data inflows are not useful for the development of the code and are not used. Nevertheless the size of the complete Blockchain raw data is around 180Gb and continues to grow each day by an estimated 137 mb on average as of August of 2018. This figure may seem relatively small to be considered as Big Data, however after the proper code has been developed and we have "decoded" the raw data, the size of the results could be multiple times more spacious and complex.

The files of interest are found in the form of .dat files each 131 mb in size. They contain the raw data of the blockchain transactions from the genesis of bitcoin in
3.2 R packages

Since parsing the Blockchain is a seriously demanding task we used already existing R tools to make the development faster and the code more efficient. The R packages we used as well as their purpose and contribution to the developing process are presented below.

DBI

The package DBI or ’Database Interface’ provides a connection between R and an SQL database. It operates in conjunction with the RSQLite package described below to enable operations such as the connection to an existing relational database, the creation and execution of statements on that database in SQL, the extraction of result/outputs that are generated from the statements as well as information (metadata) about the objects contained in the open database. The main objective of the package is to write the human readable contents of a decoded transaction into a SQL database, to avoid overloading the memory. As the number of transactions processed grew during development they occupied lots of RAM making the process slow and inefficient. More details will be presented in the next section.

RSQLite

The RSQLite package is a complementary to DBI. RSQLite is the easiest way to use a database from R because the package itself contains SQLite; no external software is needed.
3.2 R packages

openssl

The R package “openssl” is useful for encryption by applying the SHA256 cryptographic algorithm to any input data. As a primary component of the Blockchain, hash functions are greatly utilized in the code for the purposes of calculating a block’s or a transaction’s hash, since these information are not stored in the Blockchain and need external computation. The hashes are then used to establish the transaction database and locate each input’s corresponding output.

wkb

The next package used is called “wkb” and is used for converting a string’s hex representation, provided as input, into a raw vector. We use the function to decode into their raw form, transaction data that were serialized into a format named 'Blob'. A Binary Large OBject (BLOB) is a collection of binary data stored as a single entity in a database management system which in our case is SQL. The reason for this procedure is faster updating of the transaction database maintained in SQLite since they are compressed and easy to export. After that we unserialize the entire database using this R package.

Tic-toc

Tic toc is a timing package that provides the functions tic and toc. In general, calls to "tic and toc" start the timer when the 'tic' call is made and stop the timer when the 'toc' call is made, recording the elapsed time between the calls. After that a simple message with the elapsed time is printed. Every time a block is completely parsed the function returns the elapsed time of the procedure. As the point of the dissertation is to gather time data to develop an efficient code this package is heavily used providing with further data for analysis.
profvis

Profvis is a tool for helping us understand how R spends its time while running the parser code. It provides a interactive graphical interface for visualizing data from Rprof, R’s built-in tool for collecting profiling data,that correspond to the amount of time R spends in each step of the code. We use both Rprof and profvis to pinpoint the functions and exact lines that are responsible for any slowing down of the code or creating any problems. In addition the graphic interface of profvis provides with information about the memory usage ,memory allocation/deallocation as well as the milliseconds of each function.

logging

This library provides us with a tool to 'debug' our code. Debugging, in computer programming and engineering, is a multistep process that involves identifying a problem, isolating the source of the problem, and then either correcting the problem or determining a way to work around it. The final step of debugging is to test the correction or workaround and make sure it works. There are four level of logging defined in our application. **Debug** prints all information gathered by the logger, **Info** all but the information collected by debug and so on for **Warn** and **Error**.

data.table

Finally the data.table package is used to write the processed transactions into a CSV file. As the number of data to be written is large in size the fastest and most efficient package to handle the volume is data.table. It offers many options to better deal with the data and further help the analysis. We mainly use the "fwrite" function to export the data stored in the connected SQL database to a CSV file.
3.3 Blockchain Statistics

Python Package-Bitcoinlib

The unavailability of Blockchain related tools in R created problems with the code development that required external help from other programming languages for overcoming. One of the main objectives of the parser is to extract each user’s hashed Public key found at the output script. However this requires to perform a special kind of encoding solely used in the Bitcoin protocol called Base 58 encoding. As there is no such available library we opted to implement a Python call inside the R code. However this creates significant delays in the code execution as seen later in the text.\textsuperscript{10}

3.3 Blockchain Statistics

The main processing load affecting the speed and effectiveness of all already implemented parsing software is the ever growing number of transactions, especially in periods of high Bitcoin valuation, when the network is congested and transactional demands are high. It is important as a result to have an idea regarding the statistics behind the fluctuations in transactional movement as to expect relative slowdowns in the time the code operates. This will enable us to attribute any indication of slow parsing speed to an increased amount of transactions rather than a problem with the developed software. Below we present three graphs, visually highly correlated, that represent the size of the Blockchain in the span of time since its creation, the number of the Unspent Transaction Output Cache as well as the number of transactions included in a block. We gathered the data from a web API parsing the Blockchain in real time, found at www.blockchain.com.

\textsuperscript{10}Base 58 is a alteration of the Base64 encoding scheme. Base64 is a group of binary-to-text encoding schemes that represent binary data in a string format. The difference between them is that in the former, certain easy-to-confuse characters such as l, 1, 0, O etc are removed to avoid misinterpretations in the users receiving keys.
3.3 Blockchain Statistics

**Figure 4:** The size growth of the Blockchain from 2009 to 2018

![Blockchain Size Graph](image)

*Source:* Blockchain.com

*Notes:* This figure depicts the rapid growth of the Blockchain’s size. This size is tightly linked with the amount of transactions occurring on the network and seems to be steadily increasing by roughly 130 mb per day.

**Figure 5:** The per-block number of transactions

![Number of Transactions per Block Graph](image)

*Source:* Blockchain.com

*Notes:* This figure describes the number of transactions included in each block from 2009 until 2018. The recent spike in this number is probably caused by Bitcoin’s large media exposure and thus increased transactional demands from the network.
3.3 Blockchain Statistics

The changes in size of the Blockchain as seen in Figure 5 seem to be segregated into 3 different phases. The first phase from 2009 up until early 2012 is a phase of minimal growth, owing to the fact that no public awareness existed about the project prior to that time. As people came in and participated in the network, and transactions started to be created the rate of growth for the size of the Blockchain increased slowly but steadily, marking the second phase. Since 2016 the number of transactions per block occurring in the network dramatically increased creating a daily increase in blockchain size of about 137 mb. This increase remains and is expected to remain stable as the block size is fixed to be 1mb, hence the rate the blockchain size increases has a maximum cap.

Table 4. Total number of transactions at the years end.

<table>
<thead>
<tr>
<th>Year</th>
<th>Transactions</th>
<th>Percentage Increase %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>32687</td>
<td>-</td>
</tr>
<tr>
<td>2010</td>
<td>226688</td>
<td>594%</td>
</tr>
<tr>
<td>2011</td>
<td>2128883</td>
<td>839%</td>
</tr>
<tr>
<td>2012</td>
<td>10633942</td>
<td>400%</td>
</tr>
<tr>
<td>2013</td>
<td>30246540</td>
<td>184%</td>
</tr>
<tr>
<td>2014</td>
<td>55594094</td>
<td>84%</td>
</tr>
<tr>
<td>2015</td>
<td>101247924</td>
<td>82%</td>
</tr>
<tr>
<td>2016</td>
<td>184045240</td>
<td>82%</td>
</tr>
<tr>
<td>2017</td>
<td>288398401</td>
<td>57%</td>
</tr>
<tr>
<td>2018 (until 1/8)</td>
<td>338271469</td>
<td>17%</td>
</tr>
</tbody>
</table>

Source: Blockchain.com

Notes: This figure describes the number of total transactions at a year’s end as well as the percentage increase in transaction volume compared to the previous year.
3.4 Timing Data

**Figure 6:** Fluctuations of the Unspent Transaction Outputs

![Graph showing fluctuations of unspent transaction outputs over time.](image)

*Source:* Blockchain.com  
*Notes:* This figure describes the number of unspent transaction outputs currently stored by the nodes.

The number of UTXO presented in Figure 6 (unspent transaction outputs) is a matter of great importance when implementing our parsing software. Since every new transaction is created using an old unspent output as a reference, the parsing software must at all times have in store and accessible a copy of all those outputs to match with the inputs. This can be done either by keeping them in the RAM or by storing them in the hard drive. As this cache grows, the time it takes for searching and updating grows, and it is useful to have an idea when that happens and if its due to network increase movement or code inefficiencies.

### 3.4 Timing Data

During the development of the code, our primary concern was at first whether it actually runs as expected and finishes without errors. This however is not enough to consider the project complete. We also need to measure the efficiency with which it handles the increasing amount of information asked to handle. As already seen above, the amount of transactions increases exponentially with time and along with
it does the UTXO cache, as well as the amount of inputs and outputs contained in a given transaction. As a result we need to determine the efficiency of the code by collecting performance data, to locate and isolate the causes of possible time increases not explained by the input data.

Since the development of the code came in steps, in a bottom up approach\textsuperscript{11}, we collected this kind of data in all phases of production. The time data were created using the 'tic-toc' and Profvis R packages, as well as the innate to R profiling package called 'Rprof'. As already explained these libraries aim to measure the time between the beginning of the codes running duration up until it finishes or exits with error.

Using the 'Tic-toc' library we were able to extract interesting statistics regarding the time management of the code in the form of seconds spend on each block as well as the average time of blocks read in a given file. The data were stored in a csv file after the code ended running. The advantage of this package is that it provides high resolution level of data as all blocks are timed and it enables us to accurately locate the position (an exact block height) where delays starts to happen and the code starts to diverge from the expected and optimal performance. However the disadvantage is that it does not provide with the function or part of the code that causes the problem.

In order to understand the reasons of a potential slowdown or lagging performance we use the packages Rprof and Profvis. These packages give as a detailed analysis of each function used during the session as well as their memory usage and percentage of time relative to the whole. The data are stored in a txt file and later used in conjunction with the previous time data to perform our analysis.

\textbf{3.4.1 Code Development}

In the development of the code, as already mentioned, we followed a bottom-up approach and tried to ensure the project’s scalability. Since handling these big

\textsuperscript{11}A bottom-up approach is the piecing together of systems to give rise to more complex ones, thus making the original systems sub-systems of the emergent system.
amounts of encrypted data can be hard to accomplish we began testing different versions of code to find that more efficient and capable of performing the task. As a result the process of development was separated and analyzed in three major steps each providing an improvement to each predecessor. For each of these 3 steps we gathered separate timing data using the above mentioned packages. We next explain the capabilities and weaknesses of these different code versions as well as present the timing data gathered for each one. The R code can be found in the Appendix of the dissertation.

3.4.1.1 Code Version 1.0

Our first attempt at a parser was mainly focused on extracting the basic data elements of a Block, ignoring at first transaction details and their interconnection through inputs and outputs. The first step was to create a script that can read the blockchain binary raw data, and based on the scheme presented in the beginning of the thesis, correctly interpret each unique part. We also had to take into account the fact that besides the predefined Blockchain structure, there are also unique bugs in the history that cannot be removed and therefore must be considered as part of the Blockchain. One such example is the existence of an "empty" block with no information whatsoever that although not structurally correct, exists as part of the system. The results of the running code would be general information regarding the block such as its size, its SHA256 hash, the number of transactions included as well as their output amount. However, the disadvantages of this version would be inability to measure the input amount, and therefore calculate any fees, as well as find the addresses associated with each transaction, which is a very important part of the Blockchain and its back linked nature.

By timing this first version we got a first idea of how resource and time demanding our software is, whether there are unexpected errors and if parts of it can be improved before building upon it. Since the amount of calculations slightly
3.4 Timing Data

increases\textsuperscript{12} with the more transactions in a block we expect the required time for parsing to increase along with that number, however blocks with the same number of transactions should require the same amount of time. We present the scatter plot of time vs average number of transactions along with the cumulative parsing time against the number of blocks parsed. We collected around 210 thousand observations of time elapsed for each block, along with the contents of the block, such as the number of transactions and the cumulative number of inputs and outputs.

Figure 7: Parsing Time vs number of average transactions per block

Figures 7 and 8 gives us some information regarding the timing data relationships. First we can observe that the average number of transactions included in a block do have a positive relationship with the time required to parse as more calculations are needed with more 'full' blocks. On the second figure the non-linearity of parsing time vs number of blocks suggests that there is another influential variable increasing the amount of time other than blocks, further suggesting that the amount of block contents play a role.

Summary statistics presented in Table 5 gives us a first image regarding the time required for our software to decode the blockchain raw data. As seen above the average time for parsing a single block is around 0.016 seconds, however the

\textsuperscript{12}As seen in the script provided in the Appendix for each development phase.
3.4 Timing Data

**Figure 8:** Cumulative parsing time vs Number of parsed blocks

*Notes:* The vertical lines represent points where the linear relationship between time and blocks changes.

**Table 5. Summary Statistics for Code v1**

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>min</th>
<th>max</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>0.016</td>
<td>0.033</td>
<td>0</td>
<td>1</td>
<td>3424.53</td>
</tr>
<tr>
<td>InputsOutputs</td>
<td>195.932</td>
<td>475.086</td>
<td>2</td>
<td>13192</td>
<td>41304251</td>
</tr>
<tr>
<td>Transactions</td>
<td>44.985</td>
<td>107.982</td>
<td>1</td>
<td>1871</td>
<td>9475083</td>
</tr>
<tr>
<td>Observations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>210623</td>
</tr>
</tbody>
</table>

*Notes:* The table presents summary statistics for the collected time data of the 1st code version. Time is measured in seconds.

The amount of time increases as the block contents also increase. The total amount of time required for this first version to decode the available blockchain data (or around 210 thousand blocks) is around an hour or 3424 seconds. For the full blockchain the total time would be around 45 hours. However, many information such as addresses and transaction details are not supported in this code version.

### 3.4.1.2 Code Version 2.0

In the second phase of code development we started building a way to decrypt and store the details of the blockchain transactions. As we were building based upon the first version, this "code update" was just an addition to the previous work.
providing with essential modifications to complete our parser software. What we needed to do was create a storage of all parsed transactions that was easily and quickly accessible enabling the linking of transactions to their previous originating ones. As we have already explained the Blockchain does not store users’ account balance and information, rather it just stores all the performed transactions and any user can, following the links of transactions inputs and outputs, calculate his balance. We used R’s Reference Classes\textsuperscript{13} to create an environment/cache/registry where all extracted transactions are stored and later looked up on. The transaction registry is an R environment that is used as a hash table and transactions are stored as follows:

- **key**: hash of transaction
- **value**: the entire transaction

In addition the environment is used to store Statistics about the database/cache and some other variables such as: how many times did we lookup a Tx in the cache (lookup attempts), how many blocks we have read, total number of transactions encountered, number of coinbase transactions seen etc. One important thing to note here is that in order to speed up the searching time for a certain transaction this cache is maintained in the system’s RAM and is therefore always available, with the cost of potential slowdowns as we overload it. Further technical information about the environment and Reference Classes are beyond the scope of this thesis.

After creating the second version we started time testing it to again locate any potential problems, slowdowns while also checking the amount of RAM it demanded. Following the same logic as before we tried to measure the time required for the parser to compute as the number of blocks and their contents increased. We also kept records of RAM usage to understand whether any potential problem was

\textsuperscript{13}The software described here allows packages to define reference classes that behave in the style of “OOP” languages such as Java and C++. This model for OOP differs from the functional model implemented in standard R classes and methods, in which methods are defined for generic functions. Methods for reference classes are “encapsulated” in the class definition.
due to RAM mismanagement. We present below the scatter-plot of parsing time vs blocks: Figure 9 indicates a positive linear relationship between the amount of inputs and outputs of a block and the time required to decode it. However the cone shape created by the data, may indicate that 'block contents' is not the only variable that determines parsing time. Since finding a transaction in the registry is not a deterministic process and depends on luck, a block with a certain number of inputs and outputs may be decoded faster relative to other identical ones.

**Figure 9:** Parsing time vs Number of Inputs/Outputs

*Notes:*

The summary statistics of the second code phase in Table 7 provides useful insight about its efficiency and effectiveness. The average time for parsing

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Std.Dev</th>
<th>Min</th>
<th>Max</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.404</td>
<td>1.216</td>
<td>0</td>
<td>27.544</td>
<td>71056.883</td>
</tr>
<tr>
<td>Transactions</td>
<td>35.437</td>
<td>89.918</td>
<td>1</td>
<td>1852</td>
<td>6228072</td>
</tr>
<tr>
<td>Inputs/Outputs</td>
<td>154.81</td>
<td>398.225</td>
<td>2</td>
<td>13192</td>
<td>27245304</td>
</tr>
</tbody>
</table>

*Notes: This table presents summary statistics for the collected time data of the 2nd code version. Time is measured in seconds.*

The summary statistics of the second code phase in Table 7 provides useful insight about its efficiency and effectiveness. The average time for parsing
3.4 Timing Data

a block increased by almost 40 times in comparison to the first code version. The total time is also substantially increased to 71056 seconds almost twenty-fold. This may indicate a problem with the way our software handles storing and searching for parsed transaction in the cache maintained in the RAM. Results provided by the package Rprof, that measures each individual function’s running time, confirm this hypothesis and are included in the appendix. The function 'as.environment' representing the created cache, demands increasingly more time to operate resulting in slowdowns and substantial RAM engagement. As a result we concluded that this code version was not scalable, and the problem of storing and searching transaction details should be approached differently.

3.4.1.3 Code Version 3.0

After understanding that maintaining a cache in RAM for storing the transactions is not a viable solution we opted to create a database in SQL where transactions would be kept in the hard drive rather than the RAM. After downloading the appropriate R packages and a version of SQL called SQLite we modified our code to connect with these databases and be able to perform search and insert queries. Furthermore we added the previously missing capability of obtaining the addresses (or the Hashed public keys) of both the recipient and sender. One thing to note here is that since R had no packages supporting the required tools of this procedure we had to "borrow" one from python, called 'pythonlib'. In more detail we create a connection with a python script in R, which takes as input a transaction output and returns the associated addresses. This process although effective is really slow and creates a huge amount of delay especially as the number of inputs and outputs grow. A scatterplot of the parsing time vs the number of inputs and outputs in a block is presented in figure 10, as well as some summary statistics about the timing data collected, in table 8 below:

14 Acquiring an address from the raw data of the Blockchain is a procedure with certain steps one of which is Base58 encoding. As the task of this thesis is not programming an encoder we borrowed a already operational python script to perform this task.
3.4 Timing Data

**Figure 10:** Scatterplot of parsing time and the total of inputs and outputs

![Scatterplot of parsing time and the total of inputs and outputs](image)

*Notes:* A Scatterplot of parsing time and the total of inputs and outputs contained in a block. Data include 119 thousand blocks and their respective count of inputs and outputs.

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>St.Dev</th>
<th>Min</th>
<th>Max</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1.184</td>
<td>5.283</td>
<td>0.093</td>
<td>338.758</td>
<td>142031.517</td>
</tr>
<tr>
<td>InputsOutputs</td>
<td>10.692</td>
<td>43.028</td>
<td>2</td>
<td>2327</td>
<td>1276533</td>
</tr>
<tr>
<td>Transactions</td>
<td>3.628</td>
<td>15.657</td>
<td>1</td>
<td>850</td>
<td>435049</td>
</tr>
<tr>
<td>Observations</td>
<td>119976</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Notes:* The table presents summary statistics for the collected time data of the 3rd code version. Time is measured in seconds.

As Table 7 makes clear, the amount of time to parse an average block using the third version of the code is around 1,184 seconds making the software extremely slow to handle the enormous amount of information contained in the blockchain. The relationship between the number of inputs/outputs and parsing time as presented in Figure 10 seems to be strongly positive and does not seem to contain any non-linearities. Extremely problematic in this version of the code, is the python script called to decode every output in a transaction. According to Rprof, from the time it took to decode this 119 thousand blocks (142.031 seconds), around
3.4 Timing Data

92 percent was spent on 'system2' which is the command used to call the Python script. With the third version of the code we have completed building a fully capable Blockchain parser, but the amount of time required to complete the task is not optimal. However later we present some improvements that can be applied to increase its efficiency.
Chapter 4

Econometric Analysis

4.1 Linear Models

4.1.1 OLS Regression Model

For our analysis in this thesis we measure the efficiency of the created code as to locate points of slowdown and errors. The process of analyzing the collected performance data requires that we establish the relationship between the time required for the software to decode a given block as the number of blocks and the amount of transactions they contain increases. What we are looking for in these tests is to detect what other potential variables can affect parsing time, the form of that relationship as well as the effect magnitude in seconds of increased parsing demands. It is important to understand the rate at which the required time increases with each block, or as an economist would put it, we try to accurately measure the Marginal Cost (in terms of parsing seconds) of one extra decoded Block as the number of blocks increases, while also controlling for other influential factors. This would allow us to accurately predict the total parsing duration by extrapolating the coefficients and the magnitudes into the future, helping us understand the scalability and effectiveness of the software.

For the purposes of our analysis we will use linear regression models for each developmental phase of the code. Our aim using the simple linear model besides
trying to understand which factors influence our independent variable, and their impact, is to explain as highest percentage of the variance as possible. Contrary to regressions performed using economic data, where "capturing" a really high percentage of the variance (as represented by a large $R^2$), is either rare or an indicator of wrong specification, the performance data we collected do not share the same characteristics and should follow certain patterns, that if true, would enable us to very accurately model their behavior. In more detail, our first expectation is that the more efficient the code the less time required to parse a block, all other things being equal. This relates to performance and the way the code is structured. Secondly though more importantly, the effects of the independent variables are expected to remain constant and linear through the whole time of code compiling. Divergence from this pattern, like unexplained increases in parsing time, would be indicative of errors and software inefficiencies not captured by any of the possible variables. The results of such error would be unnecessary longer parsing times, that could be avoided if identified and improved upon. For this purposes consider the following linear regression model:

$$y_i = \alpha + \beta x_i + u_i$$ (4.1)

The dependent variable $y_i$ is the parsing time required for each block, measured in seconds ($time$) while for the independent ones $x_i$ we first try to measure the effect of the increasing number of blocks ($BlockHeight$). However as previously described the contents of a block are not identical and tend to increase as the Bitcoin network attracts more users, resulting in increased transactional demands. To address this issue we enhance our model with first the amount of transactions ($Transactions$) included in a block and further by the number of inputs and outputs ($InOut$). Finally $u_i$ represents the error term.

The linear model that we will estimate can be written as:

$$ParsingTime_i = \alpha + \beta Inout_i + \beta Transactions_i + \beta BlockHeight_i + u_i$$ (4.2)
4.1 Linear Models

4.1.2 OLS Model Results

Table 8 presents the results of the regression for each phase of code development as more explanatory variables are added. The results of Table 8 give as a clear picture about the influential factors and the magnitude of their effect. As already explained the main issue we want to address is the minimization of the explanatory variables coefficients, while also targeting to explain as much of the variance as possible.

By looking at Table 8 we observe that for each of the three phases of the code, `BlockHeight` as the only explanatory variable, although statistically significant at the 1% confidence level, only captures a small amount of the variance with adjusted-\(R^2\) of 0.21, 0.18 and 0.05 indicating what we already theorized, that the main factors influencing the dependent variable are the contents of the block rather than the number of blocks parsed.

By enhancing the model to include the number of transactions we significantly increased its performance and explanatory power. Captured variance as given by adjusted-\(R^2\) increased for all three regressions to 0.85, 0.81 and 0.89 respectively. However we still unable to explain all variance of the data points indicating that another variable may play an important role in determining parsing time. The coefficients of the variable "Transactions" for each of the 3 phases are all statistically significant and their magnitude is respectively 0.0003 for code V1, 0.0122 for code V2, and 0.3146 for code V3. The coefficients reflect the fact, previously seen in the the summary statistics section, that later iterations of the code require considerable more time to handle each sub-element of the Blockchain, creating concerns about the scalability of our code.

Finally we add the variable of the sum of inputs and outputs ("InOut") of a block, to test if the model’s explanatory power increases. According to Table 8 adding this variable improves the adjusted \(R^2\) of our first code regression to 0.9040, the sec-
### Table 8. Regression Results Table

<table>
<thead>
<tr>
<th></th>
<th>Parsing Time</th>
<th>(1st Phase)</th>
<th>(2nd Phase)</th>
<th>(3rd Phase)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1a)</td>
<td>(1b)</td>
<td>(1c)</td>
<td>(2a)</td>
</tr>
<tr>
<td>BlockHeight</td>
<td>2.57e-07***</td>
<td>1.48e-09***</td>
<td>-1.46e-08***</td>
<td>0.0000103***</td>
</tr>
<tr>
<td></td>
<td>(173.91)</td>
<td>(2.72)</td>
<td>(-18.65)</td>
<td>(199.76)</td>
</tr>
<tr>
<td>Transactions</td>
<td>-</td>
<td>0.0003***</td>
<td>0.0000266***</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>(939.86)</td>
<td>(2.50)</td>
<td>(771.89)</td>
<td>(34.42)</td>
</tr>
<tr>
<td>InOut</td>
<td>-</td>
<td>-</td>
<td>0.00006***</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>(24.47)</td>
<td></td>
<td>(10.62)</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>-0.0108 ***</td>
<td>0.0032***</td>
<td>0.0043***</td>
<td>-0.5030 ***</td>
</tr>
<tr>
<td></td>
<td>(-82.76)</td>
<td>(53.54)</td>
<td>(71.56)</td>
<td>(-95.91)</td>
</tr>
<tr>
<td>N</td>
<td>210623</td>
<td>210623</td>
<td>119976</td>
<td></td>
</tr>
<tr>
<td>AdjustedR²</td>
<td>0.2140</td>
<td>0.8487</td>
<td>0.9040</td>
<td>0.1850</td>
</tr>
</tbody>
</table>

* p < 0.05, ** p < 0.01, *** p < 0.001

$t$ statistics in parentheses
ond version’s $R^2$ remains almost unchanged to 0.8178 and finally the last one also improves substantially to 0.9353. The almost perfect fit of the data especially in the first and third model suggests that no other important influential factors exist, as already seen by the scatter-plots. Inability to closely model the data only in the second model, suggests that we did not miss an important variable rather the unexplained variance is due to idiosyncratic characteristics of version 2. The problem as revealed by the Rprof files included in the appendices, is the increasing time spent by the created cache (environment profile), potentially for searching the referenced previous transactions of the newly parsed ones.

Regarding the coefficients of the regression, as they are presented more condensed in Table 9, we can confirm our findings from the statistical analysis, suggesting an ever increasing marginal cost to parse a new block as we go from phase to phase. Although all variables are statistically significant, the most influential over the parsing time is the sum of inputs and outputs contained in the block. The variable of "Blockheight" captures the relationship of the parsing time with the number of blocks already parsed. Although statistically significant the effects are minuscule and remain relatively unchanged through the different versions. The variables "Transactions" and "InOut" capture the amount of seconds the parsing time increases for each additional transaction or Input/Output respectively in a block. The coefficients of the finalized code version are the largest at 0.135 and 0.0715 seconds, indicating very poor performance considering the Blockchain consists of around 340 million.
4.1 Linear Models

<table>
<thead>
<tr>
<th></th>
<th>(1st Phase)</th>
<th>(2nd Phase)</th>
<th>(3rd Phase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>0.0000266***</td>
<td>0.00971***</td>
<td>0.135***</td>
</tr>
<tr>
<td></td>
<td>(2.50)</td>
<td>(34.42)</td>
<td>(8.29)</td>
</tr>
<tr>
<td>InputsOutputs</td>
<td>0.0000626***</td>
<td>0.000601***</td>
<td>0.0715***</td>
</tr>
<tr>
<td></td>
<td>(25.48)</td>
<td>(10.26)</td>
<td>(11.37)</td>
</tr>
<tr>
<td>BlockHeight</td>
<td>-1.46e-08**</td>
<td>-0.000000112***</td>
<td>0.00000226***</td>
</tr>
<tr>
<td></td>
<td>(-18.49)</td>
<td>(-1.92)</td>
<td>(3.56)</td>
</tr>
<tr>
<td>Constant</td>
<td>0.00432***</td>
<td>-0.0229***</td>
<td>-0.200***</td>
</tr>
<tr>
<td></td>
<td>(71.56)</td>
<td>(-8.96)</td>
<td>(-9.84)</td>
</tr>
</tbody>
</table>

N adjusted $- R^2$ 210623 210623 119976

$t$ statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

The Table presents the results of each regression on the three different code versions.

4.1.3 Quantile Regression Model

Ordinary least squares regression techniques used above provide summary point estimates that calculate the average effect of the independent variables (inout3, trans3, blockheight) on the ‘average parsing time’. However, this focus on the average may hide important features of the underlying relationship. In addition, one of the underlying assumptions for OLS regression is that the error term—and by extension the dependent variable—are normally distributed. As presented in Figure 12, using the regression results for the finalized version of the code, the distribution of the dependent variable does not resemble that of the Gaussian distribution. While the optimal properties of standard regression estimators are not robust to departures from normality, quantile regression results are characteristically robust to outliers and heavy-tailed distributions. Another advantage is that, while conventional regressions focus on the mean, quantile regressions are able to describe the entire conditional distribution of the dependent variable. It is crucial as a result to examine the case of increased required time separately by using a quantile regression.
model, focusing on the upper part of the dependent variable distribution. Quantile regression techniques can help us obtain a more complete picture of the underlying relationship between parsing time and the independent variables. Developed in Koenker and Bassett (1978), this model yields parameter estimates at multiple points in the conditional distribution of the dependent variable. The model can be written as:

\[ y_i = \chi_i' \beta + u_i \text{ with } Q_\theta(y_i|x_i) = \chi_i' \beta \theta \]  

(4.3)

Where \( y_i \) is the dependent variable, \( x \) is a vector of regressors, \( \beta \) is the vector of parameters to be estimated, and \( u \) is a vector of residuals. \( Q_\theta(y_i|x_i) \) denotes the \( \theta \) th conditional quantile of \( y_i \) given \( x_i \). The model we are going to estimate is the same as [4.2] and uses the independent variables (inOut, transactions, block-height) focused only on the finalized version of the code. Regarding the quantiles we mainly focus on the upper percentiles, since possible behavior changes of the parsing software are more likely to appear when the amount of transactions included also is high. The results are presented in Table 10. The main thing we are looking for in these regressions is to investigate the behavior of the coefficients in the top 10 percentile of the distribution of the dependent variable.
4.1 Linear Models

Figure 11: Kernel density of the Residuals from the OLS regression

Notes: A Kernel density graph of the Residuals from the OLS regression for the final version of the code.

4.1.4 Quantile Regression Results

Using Stata we extracted the coefficients and estimations for the OLS and the different quantiles regressions, using robust standard errors all in Table 10. OLS estimates provide a baseline of mean effects, and we compare these to estimates for separate quantiles in the conditional distribution of the dependent variable. The quantile regression results show that the value of the estimated coefficients on the number of inputs and outputs varies greatly over the conditional distribution of the parsing time. The coefficients can be interpreted as the partial derivative of the conditional quantile of parsing time with respect to the number of inputs and outputs, the number of transactions and the blockheight. In other words, the derivative is interpreted as the marginal change in parsing time at the 90th, 92th, 94th, 96th and 98th conditional quantile due to marginal change in the number of inputs and outputs and the other independent variables.
### Table 10. Quantile and OLS regression results

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>0.135***</td>
<td>0.106***</td>
<td>0.097***</td>
<td>0.094***</td>
<td>0.090***</td>
<td>0.084***</td>
</tr>
<tr>
<td></td>
<td>(8.29)</td>
<td>(21.98)</td>
<td>(16.74)</td>
<td>(23.16)</td>
<td>(32.58)</td>
<td>(15.46)</td>
</tr>
<tr>
<td>InOut</td>
<td>0.0715***</td>
<td>0.146***</td>
<td>0.148***</td>
<td>0.153***</td>
<td>0.162***</td>
<td>0.180***</td>
</tr>
<tr>
<td></td>
<td>(11.37)</td>
<td>(12.78)</td>
<td>(18.45)</td>
<td>(25.50)</td>
<td>(16.49)</td>
<td>(20.78)</td>
</tr>
<tr>
<td>BlockHeight</td>
<td>2.26e-06***</td>
<td>1.36e-20</td>
<td>2.76e-08</td>
<td>1.63e-07***</td>
<td>2.56e-07***</td>
<td>3.32e-07</td>
</tr>
<tr>
<td></td>
<td>(3.56)</td>
<td>(0.01)</td>
<td>(0.96)</td>
<td>(8.40)</td>
<td>(6.77)</td>
<td>(1.91)</td>
</tr>
<tr>
<td>_cons</td>
<td>-0.200***</td>
<td>0.00112</td>
<td>0.00205</td>
<td>0.00751***</td>
<td>0.0113***</td>
<td>0.0244***</td>
</tr>
<tr>
<td></td>
<td>(-9.84)</td>
<td>(0.46)</td>
<td>(0.82)</td>
<td>(3.77)</td>
<td>(5.45)</td>
<td>(4.25)</td>
</tr>
<tr>
<td>N</td>
<td>119976</td>
<td>119976</td>
<td>119976</td>
<td>119976</td>
<td>119976</td>
<td>119976</td>
</tr>
<tr>
<td>pseudo $R^2$</td>
<td>0.9353</td>
<td>0.9239</td>
<td>0.9295</td>
<td>0.9353</td>
<td>0.9428</td>
<td>0.9526</td>
</tr>
</tbody>
</table>

* $t$ statistics in parentheses
** $p < 0.05$, *** $p < 0.01$, **** $p < 0.001$

Quantile regression results on the 92,94,96 and 98 quantile of the dependent variable compared to the OLS results.

In more detail Table 10 illustrates the fact that as we go to higher distribution quantiles the coefficient among the regressors change, and seem to increase, indicating higher marginal cost of parsing a block, when the blocks contain a lot of transactions and inputs/outputs. The OLS regression results differ greatly from that of the quantile one since an extra input/output according to the ordinary least squares model requires 0.07 extra seconds for the software to run, while in the upper quantiles, where the parsing time is in the top 10 percent of the distribution, almost double the time is required and ever increasing. This may indicate that as the amount of transactions in a block increases, the more time the parser spends decoding each of them. The variable of the number of transactions seems to play a smaller role since the quantile regressions coefficients seem to reduce as we use the upper part of the parsing time distribution. Finally the variable of BlockHeight seems to not play an important role in determining the software’s running time.
Chapter 5

Conclusion

The rising popularity of Cryptocurrencies and particularly that of Bitcoin, has created an increasing gap in the research literature regarding its nature and the way it operates. It can be shortly described as an decentralized electronic means of transacting, available to anyone with an Internet connection. One of the primary innovations behind the Bitcoin Protocol is what is called the 'Blockchain', an ever updating database containing all essential information about any past transactions occurring in the network. These data are publicly available to anyone wanting to perform research, however since the structure and format with which they are maintained in the Blockchain is encoded and not human readable, a phase of preprocessing must occur using specially created software.

The aim of this dissertation was to create such a software, that would decode the Blockchain data and return any useful information (called a parser), using the programming language R. Although there are a plethora of parsing software available in other programming languages, this is the only one operating using R. We were also interested in measuring the efficiency of our created software as to measure its ability to scale and handle the vast amount of information contained in the Blockchain.

We started by laying out the fundamental elements of the Blockchain as well as the way the system operates. The software we created was based on the schematic
of this system, taking into account all the sub elements and relationships it envelops as described thoroughly in the theoretical part of the thesis. Of utmost importance was the Transaction scheme, incorporating an Input Output system, that connects all performed transactions. The purpose of this interconnection of transactions along with other Blockchain innovations, such as the "proof of work system", ensure the operation, stability and security of the protocol, without external interventions, thus making it completely decentralized. Knowing the way all these different innovations and elements interact was essential before proceeding to creating the parser.

The code development phase was performed in three separate stages, following a bottom up approach. The input data used for testing were directly downloaded from the Blockchain using the "Bitcoin core" software. As we tested the code we also collected timing data, measuring the performance and required parsing time in addition to identifying the underlying causes of encountered problems. Our purpose was to monitor the efficiency and predict the way our software would behave as the amount of encoded data to be processed rose.

For the purposes of our performance analysis we used econometric tools and more specifically linear regression models to identify the variables that influence the parsing time as well as the magnitude of that effect. In addition we tried to ensure that the coefficients of our explanatory variables be constant as to allows us to safely predict the software’s efficiency and behavior as the amount of input data increases. The main tools we used was at first the Ordinary Least Squares model, using as the dependent variable the parsing time, while sequentially adding explanatory variables, such as the number of blocks parsed, the number of transactions as well as the number of inputs and outputs, to check how the model’s accuracy improved and which is the most influential factor. We found that the most important determinant of the parsing time is the number of inputs and outputs included in a block, while the other two captured only a small amount of the variance. This model explained 93 percent of the variance, a rather big number.
not easily encountered in analysis of economic data. However the coefficient of this variable calculated at 0.07 seconds indicates that our code is not sufficiently fast to handle the vast amount of information contained in the Blockchain and further software improvements are essential.

The second part of our empirical research focused on the behavior of the coefficients on different parts of the dependent’s variable distribution. Using quantile regression focusing on the upper quantiles of parsing time, we tried to model the behavior of the previously used explanatory variables. We presented the results compared with them in the OLS regression and found, that at the higher quantiles of the distribution the coefficient of the number of inputs and outputs almost doubled in comparison to the OLS one, while the other explanatory variables effects either were reduced or stayed the same. This indicates that a block with a higher number of inputs and outputs has increased marginal time cost of parsing an extra input or output, significantly reducing its efficiency.

All results stemming from the regression analysis suggests that the software we created although operational, is extremely slow and inefficient in handling the vast amount of data, in a reasonable time frame. Potential modifications in the code that could substantially improve it, and a further continuation of our research, could be creating the R analogous of the python ’Bitcoinlib’ library. One of the main subtasks of our code was to use Base-58 encoding in order to export the user’s address. This is accomplished by utilizing external help from the before mentioned python script. However as shown in the appendix, the time spent by the code on this python script is multiple times that of any other functions creating a huge slowdown. A future improvement to our code could be the creation of an R script able to perform this Base-58 encoding as to avoid implementing any python scripts in the code.
Appendix A

Chapter 1

In this first Appendix we present a flowchart with the most important code functions and interactions, we used to parse the Blockchain. Since the raw code is more than a thousand lines, it can not be included as a whole so we opted to present it like this:

Figure A.1: Simple Code Flowchart

Notes: A Simple Code Flowchart for the final version of the code.
Appendix B

Chapter 2

As explained throughout the thesis, one of the most important parts of code testing was the result created by the R package Rprof. We used the profiling capabilities of R as to assess the way our code spends its running time and identify problems. One example is the enormous demands in terms of time of the python script we incorporated in our code. The profiling files helped us identify the problems and suggest possible solutions. We present here some of the Rprof files, in tables 13 and 14, for the running of the second and third code version.

The V3 of our code as obvious by the Rprof files spends an disproportional amount of time calling the "system2" command, that connects with the python script. This is a potential area of improvement, since by creating the Base 58 encoder precisely mention the system 2 command may be totally ignored, resulting in a 10 times faster version of the code.
<table>
<thead>
<tr>
<th>Code V3 functions</th>
<th>self.time</th>
<th>Percent of total</th>
<th>Code v3 Functions</th>
<th>self.time</th>
<th>self.pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>system2</td>
<td>20808.84</td>
<td>88.09</td>
<td>system2</td>
<td>12971.04</td>
<td>92.76</td>
</tr>
<tr>
<td>substring</td>
<td>857.18</td>
<td>3.63</td>
<td>vapply</td>
<td>172.24</td>
<td>1.23</td>
</tr>
<tr>
<td>vapply</td>
<td>604.6</td>
<td>2.56</td>
<td>FUN</td>
<td>150.26</td>
<td>1.07</td>
</tr>
<tr>
<td>FUN</td>
<td>448.84</td>
<td>1.9</td>
<td>result_create</td>
<td>113.3</td>
<td>0.81</td>
</tr>
<tr>
<td>result_create</td>
<td>132.32</td>
<td>0.56</td>
<td>substring</td>
<td>111.68</td>
<td>0.8</td>
</tr>
<tr>
<td>gsub</td>
<td>99.72</td>
<td>0.42</td>
<td>paste</td>
<td>63.48</td>
<td>0.45</td>
</tr>
<tr>
<td>paste</td>
<td>73.46</td>
<td>0.31</td>
<td>gsub</td>
<td>39.52</td>
<td>0.28</td>
</tr>
<tr>
<td>initialize</td>
<td>37.44</td>
<td>0.16</td>
<td>initialize</td>
<td>25.08</td>
<td>0.18</td>
</tr>
<tr>
<td>is</td>
<td>29.84</td>
<td>0.13</td>
<td>is</td>
<td>18.84</td>
<td>0.13</td>
</tr>
<tr>
<td>validObject</td>
<td>27.58</td>
<td>0.12</td>
<td>validObject</td>
<td>18.08</td>
<td>0.13</td>
</tr>
<tr>
<td>as.environment</td>
<td>26.52</td>
<td>0.11</td>
<td>as.environment</td>
<td>15.82</td>
<td>0.11</td>
</tr>
<tr>
<td>.getClassesFromCache</td>
<td>23.12</td>
<td>0.1</td>
<td>.getClassesFromCache</td>
<td>15.46</td>
<td>0.11</td>
</tr>
<tr>
<td>strtoi</td>
<td>20.1</td>
<td>0.09</td>
<td>strtoi</td>
<td>12.98</td>
<td>0.09</td>
</tr>
<tr>
<td>getClassDef</td>
<td>17.8</td>
<td>0.08</td>
<td>.local</td>
<td>11.88</td>
<td>0.08</td>
</tr>
<tr>
<td>.local</td>
<td>17.76</td>
<td>0.08</td>
<td>@&lt;-</td>
<td>11.44</td>
<td>0.08</td>
</tr>
<tr>
<td>doTryCatch</td>
<td>13.46</td>
<td>0.06</td>
<td>doTryCatch</td>
<td>9.04</td>
<td>0.06</td>
</tr>
<tr>
<td>.identC</td>
<td>12.5</td>
<td>0.05</td>
<td>.identC</td>
<td>8.98</td>
<td>0.06</td>
</tr>
<tr>
<td>dbClearResult</td>
<td>11.24</td>
<td>0.05</td>
<td>$</td>
<td>8.16</td>
<td>0.06</td>
</tr>
<tr>
<td>tryCatch</td>
<td>10.76</td>
<td>0.05</td>
<td>dbClearResult</td>
<td>7.82</td>
<td>0.06</td>
</tr>
<tr>
<td>levellog</td>
<td>10.72</td>
<td>0.05</td>
<td>tryCatch</td>
<td>7.62</td>
<td>0.05</td>
</tr>
<tr>
<td>result_fetch</td>
<td>10.18</td>
<td>0.04</td>
<td>tryCatchOne</td>
<td>7.18</td>
<td>0.05</td>
</tr>
<tr>
<td>logger$log</td>
<td>9.84</td>
<td>0.04</td>
<td>strtoi</td>
<td>6.14</td>
<td>0.04</td>
</tr>
<tr>
<td>tryCatchOne</td>
<td>9.62</td>
<td>0.04</td>
<td>anyDuplicated</td>
<td>5.92</td>
<td>0.04</td>
</tr>
<tr>
<td>getILogger</td>
<td>9.04</td>
<td>0.04</td>
<td>&lt;Anonymous&gt;</td>
<td>5.3</td>
<td>0.04</td>
</tr>
<tr>
<td>anyDuplicated</td>
<td>8.6</td>
<td>0.04</td>
<td>logger$log</td>
<td>5.12</td>
<td>0.04</td>
</tr>
<tr>
<td>rev</td>
<td>7.36</td>
<td>0.03</td>
<td>levellog</td>
<td>5.06</td>
<td>0.04</td>
</tr>
<tr>
<td>exists</td>
<td>6.74</td>
<td>0.03</td>
<td>result_fetch</td>
<td>5</td>
<td>0.04</td>
</tr>
<tr>
<td>dbSendQuery</td>
<td>6.28</td>
<td>0.03</td>
<td>rev</td>
<td>4.68</td>
<td>0.03</td>
</tr>
<tr>
<td>length</td>
<td>6.22</td>
<td>0.03</td>
<td>getILogger</td>
<td>4.3</td>
<td>0.03</td>
</tr>
<tr>
<td>tryCatchList</td>
<td>5.32</td>
<td>0.02</td>
<td>dbSendQuery</td>
<td>4.28</td>
<td>0.03</td>
</tr>
<tr>
<td>slot&lt;-</td>
<td>5.24</td>
<td>0.02</td>
<td>length</td>
<td>3.38</td>
<td>0.02</td>
</tr>
<tr>
<td>dbGetQuery</td>
<td>5.1</td>
<td>0.02</td>
<td>TOTAL TIME</td>
<td>13355.51</td>
<td></td>
</tr>
</tbody>
</table>

TOTAL TIME 23332.08

Notes: The table presents the results of the profiling R package R prof regarding the amount of time each function of the code demands.
Table B.2. *The results of the profiling R package*

<table>
<thead>
<tr>
<th>Functions of code V2</th>
<th>self.time</th>
<th>self.pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>env.profile</td>
<td>13557.66</td>
<td>65.4</td>
</tr>
<tr>
<td>as.environment</td>
<td>2229.26</td>
<td>10.75</td>
</tr>
<tr>
<td>paste</td>
<td>799.4</td>
<td>3.86</td>
</tr>
<tr>
<td>levellog</td>
<td>455.88</td>
<td>2.2</td>
</tr>
<tr>
<td>$</td>
<td>444.04</td>
<td>2.14</td>
</tr>
<tr>
<td>exists</td>
<td>440.08</td>
<td>2.12</td>
</tr>
<tr>
<td>getLogger</td>
<td>261.22</td>
<td>1.26</td>
</tr>
<tr>
<td>logger$log</td>
<td>180.56</td>
<td>0.87</td>
</tr>
<tr>
<td>h2d</td>
<td>146.52</td>
<td>0.71</td>
</tr>
<tr>
<td>readBin</td>
<td>141.94</td>
<td>0.68</td>
</tr>
<tr>
<td>get</td>
<td>139.56</td>
<td>0.67</td>
</tr>
<tr>
<td>&lt;Anonymous&gt;</td>
<td>98.38</td>
<td>0.47</td>
</tr>
<tr>
<td>rev</td>
<td>95.32</td>
<td>0.46</td>
</tr>
<tr>
<td>as.list</td>
<td>94.22</td>
<td>0.45</td>
</tr>
<tr>
<td>handleTx</td>
<td>93.84</td>
<td>0.45</td>
</tr>
<tr>
<td>logdebug</td>
<td>83.86</td>
<td>0.4</td>
</tr>
<tr>
<td>.getClassesFromCache</td>
<td>76.86</td>
<td>0.37</td>
</tr>
<tr>
<td>readBlockChainFile</td>
<td>70.5</td>
<td>0.34</td>
</tr>
<tr>
<td>assign</td>
<td>63.86</td>
<td>0.31</td>
</tr>
<tr>
<td>execScript</td>
<td>62.82</td>
<td>0.3</td>
</tr>
<tr>
<td>readBytes</td>
<td>62.76</td>
<td>0.3</td>
</tr>
<tr>
<td>c</td>
<td>56.02</td>
<td>0.27</td>
</tr>
<tr>
<td>length</td>
<td>55.1</td>
<td>0.27</td>
</tr>
<tr>
<td>is</td>
<td>54.96</td>
<td>0.27</td>
</tr>
<tr>
<td>readTx</td>
<td>54.94</td>
<td>0.27</td>
</tr>
<tr>
<td>vapply</td>
<td>46.32</td>
<td>0.22</td>
</tr>
<tr>
<td>readTxInput</td>
<td>46.26</td>
<td>0.22</td>
</tr>
<tr>
<td>readTxOutput</td>
<td>38.94</td>
<td>0.19</td>
</tr>
<tr>
<td>[</td>
<td>33.6</td>
<td>0.16</td>
</tr>
<tr>
<td>readVariantInteger</td>
<td>32.86</td>
<td>0.16</td>
</tr>
<tr>
<td>structure</td>
<td>32.28</td>
<td>0.16</td>
</tr>
<tr>
<td>getClassDef</td>
<td>29.74</td>
<td>0.14</td>
</tr>
<tr>
<td>txRS$newLookup</td>
<td>29.42</td>
<td>0.14</td>
</tr>
<tr>
<td>$&lt;-</td>
<td>28.48</td>
<td>0.14</td>
</tr>
<tr>
<td>getNamespace</td>
<td>27.4</td>
<td>0.13</td>
</tr>
<tr>
<td>methods:::.setDummyField</td>
<td>25</td>
<td>0.12</td>
</tr>
<tr>
<td>envRefSetField</td>
<td>24.74</td>
<td>0.12</td>
</tr>
<tr>
<td>readBlock</td>
<td>24.36</td>
<td>0.12</td>
</tr>
<tr>
<td>rawhash</td>
<td>22.24</td>
<td>0.11</td>
</tr>
<tr>
<td>as.character</td>
<td>21.36</td>
<td>0.1</td>
</tr>
<tr>
<td>raw</td>
<td>21.3</td>
<td>0.1</td>
</tr>
<tr>
<td>rev.default</td>
<td>19.96</td>
<td>0.1</td>
</tr>
<tr>
<td>match.call</td>
<td>19.3</td>
<td>0.09</td>
</tr>
<tr>
<td>print.default</td>
<td>19.12</td>
<td>0.09</td>
</tr>
<tr>
<td>TOTAL TIME</td>
<td>23547.55</td>
<td></td>
</tr>
</tbody>
</table>

Notes: The table presents the results of the profiling R package R prof regarding the amount of time each function of the code demands, for the seconds version of the code.
References


Diebold, F. (2012). A personal perspective on the origin (s) and development of ‘big data’: The phenomenon, the term, and the discipline, second version.


