Karlsruhe Institute of Technology

An Evaluation of Coin Selection Strategies

Master's Thesis
of

Mark Erhardt

at the Institute of Telematics,
Department of Informatics,
Karlsruhe Institute of Technology

Reviewer:	Prof. Dr. M. Zitterbart
Advisor:	Dipl.-Inform. M. Florian
Second advisor:	M.Sc.-Inform. S. Friebe

Preparation time: 2016-05-01 – 2016-10-31

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 31. Oktober 2016

Zusammenfassung

Bitcoin-Transaktionen erzeugen neue unspent transaction outputs (UTXOs), welche nur als ganzes verbraucht werden können. Um eine Transaktion zu erzeugen, müssen eine oder mehrere zur Verfügung stehende UTXOs als Input für diese Transaktion ausgewählt werden. Bleibt über den zu zahlenden Betrag Geld übrig, weist sich der Sender den Restbetrag in Form einer weiteren, neuen UTXOs selbst zu.

Bei der Auswahl dieser Transaktions-Inputs, der *"Coin Selection"*, gibt es verschiedene teilweise widersprüchliche Optimierungsziele. Um Blockplatz zu sparen und die notwendigen Transaktionsgebühren zu minimieren, sollte die Anzahl der Inputs möglichst klein sein. Um möglichst wenig private Informationen zu offenbaren, sollten wenige Adressen miteinander verknüpft werden und die Auswahl der Inputs sollte keine zusätzlichen Informationen über die dem Nutzer zur Verfügung stehenden UTXOs offenbaren. Andererseits sollten UTXOs nicht unnötig aufgespalten und Wechselgeld-Outputs nicht zu klein werden, da kleine UTXOs relativ zum eigenen Wert hohe Kosten bei der Ausgabe verursachen. Besonders problematisch ist, dass alle UTXOs des gesamten Netzwerks stets von jedem Bitcoin-Knoten parat gehalten werden müssen, da die Gültigkeit von Transaktionen anhand des unspent transaction output set (UTXO set) geprüft wird.

Nach einer Erläuterung der wesentlichen Grundlagen werden in dieser Arbeit die Anforderungen an und Ziele des oben beschriebenen Coin Selection-Problems detailliert vorgestellt. Es werden mehrere Algorithmen beschrieben, die derzeit von Bitcoin-Nutzern zur Lösung dieses Problems genutzt werden.

Im Folgenden wird das neu entwickelte Simulations-Framework eingeführt, welches verwendet wird um Coin Selection-Strategien zu vergleichen und ihren Effekt zu modellieren. Hierbei wird ein Modell für Bitcoin-Geldbörsen verwendet, das deren Funktion auf die Verwaltung von UTXOs und Transaktionsgenerierung reduziert. Der Simulator speist dieses Model mit eingehenden und ausgehenden Zahlungsaufträgen, so dass der UTXO-Vorrat der Bitcoin-Geldbörsen unter Verwendung von verschiedenen Coin Selection-Strategien verfolgt werden kann.

Es werden mehrere selbst entwickelte und implementierte Lösungsstrategien vorgestellt. Unter anderem werden hier mehrere Abwandlungen der Strategie des Referenz-Clients besprochen, aber mit der Branch'n'Bound-Strategie auch ein eigener Entwurf eingeführt.

Die Lösungsstrategien werden dann mithilfe des entwickelten Simulations-Framework evaluiert. Hierbei wird als Basis ein Zahlungsdatensatz eines Bitcoin-Online-Börsenservices herangezogen und zwei weitere abgeleitete Szenarien durchgespielt.

In der Evaluation werden die Wirkung auf das UTXO set, die Kosten für den Börsenbesitzer und andere Kriterien untersucht. Auf Basis der Ergebnisse werden Mängel und Vorzüge der verschiedenen Strategien ausgearbeitet. Für simple Geldbörsen ergibt sich zufällige Auswahl als hinreichend, für umfangreichere Projekte ist die neu entwickelte Branch'n'Bound-Strategie empfehlenswert.

Contents

1. Introduction

Bitcoin is a decentralized currency system governed by a peer-to-peer network. It was introduced by a group or person calling themselves Satoshi Nakamoto in their seminal paper [Naka08]. Value is transferred on the network by sending transactions. Approved transactions get collected in a decentrally maintained append-only log, the blockchain. Balances in the Bitcoin network are kept in the form of unspent transaction outputs (UTXOs). When transactions are created they consume UTXOs as inputs, and create new UTXOs as outputs.

The selection of the inputs for transactions in Bitcoin, or *coin selection* is an optimization problem with implications for user privacy, block space availability, transaction cost, confirmation times, and node memory burden. These entail a range of partly opposed goals and incentives for coin selection. For example, to reduce node memory burden, it is preferable to use more inputs while transaction cost is lower for fewer inputs.

The coin selection approaches currently prevalent on the network cause increased costs to node operators in the Bitcoin network. An improved solution is part of Bitcoin Core's roadmap for version 0.14 [Bitc16b].

1.1 Goals of this Work

This work provides a comprehensive analysis of the coin selection problem. To that end, the requirements and constraints are explored and described in detail. The design space, to which new coin selection strategies must adhere, is defined. Existing coin selection approaches in Bitcoin Core and other wallets are analyzed.

A simulation is created to compare the fitness of various existing and proposed coin selection approaches. The various approaches are simulated in multiple scenarios consisting of numerous incoming and outgoing payments.

The overall goal is to encompassingly describe the coin selection problem, existing solutions, and to provide a tool to inform the Bitcoin community of the benefits for users and network of different coin selection approach.

A secondary goal is to illustrate options for a long-term reduction of the unspent transaction output set (UTXO set)[1] size.

1.2 Acknowledgements

The author would like to thank his advisors Martin Florian and Sebastian Friebe for their valuable advice, input and feedback. In addition the author is grateful for the discussion input provided by Pieter Wuille (sipa), Jonas Schnelli, Gregory Maxwell (nullc), Gregory Sanders (instagibbs), Matt Corallo (TheBlueMatt), and Marco Falke over coffee, on IRC, and on Github, and David A. Harding for his outstanding contributions to Bitcoin.Stackexchange.com and the Bitcoin Developer Guide. Thank you to Ryan Havar for making available his payment dataset on GitHub, this work would have been a lot less exciting without it.

Lots of thanks to Tobias Zirr for helping to maintain the author's sanity in the past months by listening to him ramble about his thesis over burgers.

[1] Throughout this work, UTXOs refers to a further specified selection of UTXOs, unspent transaction output pool (UTXO pool) refers to one specific node's UTXOs, and UTXO set refers to the globally existing UTXOs.

2. Fundamentals

The Bitcoin[1] network is a decentral peer-to-peer network aimed at creating consensus in adverserial conditions. Hereby, the purpose of the Bitcoin protocol's rules is to faciltate a currency system with a limited money-supply. The main contribution of Satoshi Nakamoto's whitepater [Naka08] was to show how to create digital scarcity without relying on a trusted third party. Instead, the nodes rely on a decentrally maintained append-only log of all transactions verified on the Bitcoin network, the blockchain. The following sections will introduce the blockchain, Bitcoin transactions, and Bitcoin Core.

2.1 Nodes

The Bitcoin network is created from thousands of computers running the same software. Each node connects to at least eight peers in a decentralized network. By means of a gossip protocol, messages are passed through the network. These messages are called transactions and represent payment orders from one network participant to another. Each node checks every transaction for authenticity and validity.

However, if each node would just collect transactions as they are passed through the network, different nodes would receive notice of transactions at different times. Nodes would then maintain transactions in different order, and if an adversary were to broadcast a doublespend, i.e. two competing transactions spending the same funds of which only one can be confirmed, the network would split irreconcilably.

Hence, a subset of the nodes, the miners, engages in a Poisson process to pick an authoring node at random, which gets to produce the canonical update of the network's state, a *block*. Eligibility to win the mining process is bought with computational expenditure.

[1] Throughout this work, Bitcoin with a capital B refers to the protocol, concept, or network, while bitcoin or BTC refers to the currency unit.

2.2 Blockchain

The blockchain is the ledger of the Bitcoin network. It is a register of every transaction that was confirmed and therefore an accurate representation of all users' claims to bitcoins. The blockchain serves as the truth source and synchronization mechanism in the Bitcoin network.

So, to achieve consensus on the order of the transactions, the network designates a random node to extend the blockchain by one block approximately every ten minutes. The block comprises the node's version of the recent transactions and thusly forms an atomic set of updates to the state of the network. Each miner continuously creates block candidates and checks each candidate as to whether it fulfills the required properties to extend the blockchain. The underlying mechanism of mining is to apply the doubled SHA-256 cryptographic hash function to different candidate blocks until the resulting digest is a lower binary number than a global difficulty level.

As a valid block is discovered, the authoring miner broadcasts it to its peers. Other nodes check the new block's validity, then update their blockchain state accordingly. Immediately, the mining process starts anew to build upon the latest block.

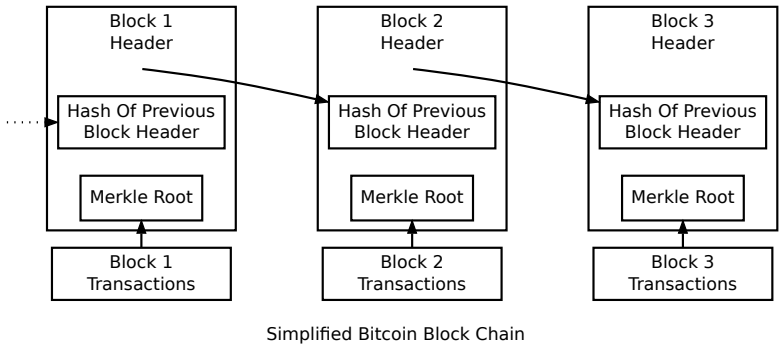

Figure 2.1: Blocks are chained together by including the digest of the preceding block [Bitc16d]

Hereby, each block is chained to its predecessor block by referencing the predecessor's hash digest in the header (see figure 2.1). As blocks build on each other inductively, more computational work is heaped on past blocks, continuously increasing the effort necessary to present a competing blockchain of equivalent computational work. The transactional history therefore becomes incrementally harder to change, and older blocks effectively immutable.

The resulting blockchain is a public property, available to everyone, providing full transparency of all transactions on the Bitcoin network while maintaining the privacy of the involved users by only listing them pseudonymously.

2.3 Transactions

Transactions in Bitcoin are the mechanism to transfer value to other users. Hereby, a sender creates an order to the network, which signs over funds to the control of a recipient. This *unconfirmed* transaction is relayed through the network. Each node

checks the validity and authenticity of the transaction, and then includes it in its memory pool of unconfirmed transactions. Once a miner finds a new block including the transaction, the transaction is confirmed. Upon receiving a block, every node updates their local registry of unspent transaction outputs (UTXOs) according to the recorded transactions, and removes the confirmed transactions from the memory pool. At that point, a payment can be considered to be settled.[2]

2.3.1 Composition of a Transaction

Transactions are identified by a hash over their data, the Transaction ID (TXID). Thanks to the *collision resistance* of the used cryptographic hash function, the TXID is effectively unique. Transactions in Bitcoin are composed of three pieces of information:

- a list of inputs into the transaction
- a list of outputs created by the transaction
- a signature

Each output has a unique index within the transaction and assigns an amount of bitcoins to whomever can satisfy the requirements of the included *output script*. The standard output script is Pay-to-PubKey-Hash (P2PKH) which requires proof of ownership of the private key corresponding to the recipient Bitcoin address to be spent.

Just as the outputs, an input has a unique index in the transaction. Each input references a specific output of a previous Bitcoin transaction. Such an output must not have been spent previously, and is therefore referred to as an unspent transaction output (UTXO) (see figure 2.2). The output is unambiguously identified by the TXID of the transaction that created it, and the output's index in that transaction.

The UTXO includes an output script which defines the requirements to allow spending. The sender must provide an input script that satisfies said ouput script's requirements. For example with P2PKH, the output script assigns the bitcoins to the owner of a specific address. The input script then proves the ownership of the private key associated with that address to allow spending.

Obviously, transactions cannot send more money than the consumed funds or they would be creating new money. Therefore, the value of the inputs must be equal or in excess of the value of the outputs.

$$\sum \text{inputs} - \sum \text{outputs} = \text{transaction fee} \tag{2.1}$$

The value of the inputs generally slightly exceeds the value of the outputs. The unassigned remainder, which results from the difference, may be claimed by the

[2]Strictly speaking, the payment could still be invalidated if a chain reorganization occurred and the chain tip that obsoleted the confirming block contained a doublespend of the transaction instead. In the case of valuable transactions, a user may opt to wait a few more blocks to reduce the likelihood of such an event.

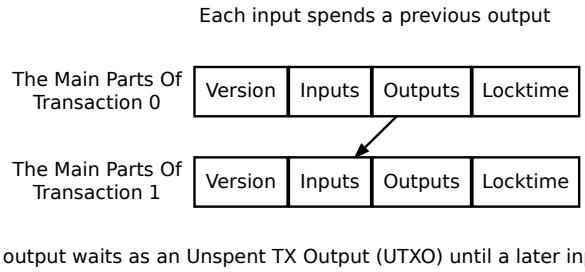

Figure 2.2: Transactions create and consume UTXOs [Bitc16e]

miner that authors the confirming block as the *transaction fee* (see Equation 2.1). While it was common for transactions without fees to get confirmed in the past, today users bid with transaction fees for inclusion in the limited blockspace.

To ensure the authenticity of the transaction, the transaction is completed with a signature over all of the above transaction data.

2.3.2 ”Balances” and UTXOs

When a user looks at their wallet, the software aggregates the sum of value of all their UTXOs and presents it to them as their ”balance”. In banking, when a transaction is issued the sent amount is deducted from the customer's account balance by their bank. It is a common question of people learning about Bitcoin how ”balances” are managed in Bitcoin.

Bitcoin doesn't know balances associated with an account or username as they appear in banking. On a technical level funds in Bitcoin exist solely in form of the aforementioned UTXOs. ”Balances” are merely an abstraction for the user's convenience. Hereby, spending a UTXO is more similar to cashing in a cashier's check than a bank wire: you have to authenticate your ownership, can only use it in full or not at all, and then get to assign the destination of the assigned value.

2.3.3 Change

As the number of UTXOs at a sender's disposal is finite, it is unlikely that a sender will have a combination of UTXOs that exactly matches the amount he wants to spend. Hence, most transactions have at least two outputs, one to send money to the recipient, and another to return the remainder after fees to the sender. The latter output usually goes to a new address owned by the sender, and is referred to as change output.

Example: Alice wants to send 1.5 BTC to Bob. She has two UTXOs at her disposal: one of 1 BTC and 0.8 BTC. The resulting transaction uses both UTXOs as inputs, and creates two outputs. One output assigns 1.5 BTC to Bob, a second output sends the change of 0.2999 BTC to Alice herself. The unassigned remainder of 0.0001 BTC will be claimed by the miner as transaction fee.

2.3.4 Dust

A transaction output is labeled as *dust* when its value is similar to the cost of spending it. Precisely, Bitcoin Core sets the dust limit to a value where spending an

output would exceed 1/3 of its value. This calculation is based on the minimum relay transaction fee, a node setting that causes transactions that don't at least include this lower bound of fee to be dropped from the memory pool, and not relayed to other nodes. With the default for the minimum relay transaction fee set to 1 000 satoshi per kilobyte, and the sizes of a P2PKH input being 148 bytes, and an output being 34 bytes, this computes to all outputs smaller or equal to 546 satoshis being considered dust by Bitcoin Core [Erha15]. The general formula to calculate the dust limit of a node when a different minimum relay transaction fee is set can be seen below in equation 2.2.

$$\text{dustLimit} = 3 \times (148 + 34)\text{bytes} \times \text{minRelayTxFee}\frac{\text{satoshi}}{\text{kilobyte}} \times 0.001\frac{\text{kilobytes}}{\text{byte}} \quad (2.2)$$

2.3.5 Validation and Execution of a Transaction

Each Bitcoin Core node maintains a list of the complete network's UTXOs locally, the UTXO set.

Whenever Bitcoin Core is lagging behind the blockchain tip, it will request all new blocks from peers and completely validate the included data. Hereby, it also checks and executes all transactions on its local copy of the network state.

For each transaction, it checks that the inputs are listed in the UTXO set, i.e. that they haven't been spent yet. When the transaction is valid, all UTXOs used as inputs are removed from the UTXO set and all new outputs created by the transaction are added to the UTXO set. As there is a definitive order to the transactions, every node deterministically arrives at the same UTXO set for each given height in the blockchain, especially all nodes share the same UTXO set when they are synchronized with the current blockchain tip.

2.3.6 Private key, public key, and address

Wallets are software that allows users to manage their bitcoins and track the network for transactions relevant to them. As there are no formal restrictions to entering the Bitcoin network, anyone may start using one or multiple Bitcoin wallets at their own discretion. When a new wallet is first activated at least one new private key will be generated. From this private key, a public key and an address are derived. While the private keys are used to prove ownership of UTXOs, addresses can be thought of as pseudonymous identifiers of users in the network. Addresses may be used to designate the recipient of payments. It is good practice to use a new address (and therefore new private key) every time a user wants to receive a payment. Technically, private keys are simply random numbers from the range of $[0, 2^{256} - 1]$. As this range of numbers is enormous, it is unlikely that anyone will ever re-create a private key that has been discovered before.

Bitcoin keys use the Elliptic Curve Digital Signature Algorithm (ECDSA), specifically the secp256k1 curve. ECDSA uses key pairs composed of a private and a public key. Private key and public key are in a mathematical relationship such that it is easy to derive the public key from the private key, but infeasible to derive the private key from the public key. The key pair represents a point on the elliptic curve.

Early in Bitcoin, payment recipients were identified directly with the public key in an output script called Pay-to-PubKey (P2PK). Today, the most common recipient representation is P2PKH, which use addresses.

Addresses are derived from the public key in a scheme using both SHA-256 and RIPEMD160. The address space has only 2^{160} unique addresses compared to the 2^{256} key pairs.

The addresses are usually presented encoded using Base58Check. Base58 is composed from the digits, lower case letters, and capital letters excluding the symbols 0 (zero), O (capital o), I (capital i), l (lower case L) due to their similarity. Base58Check additionally includes a four byte checksum at the end, making it highly unlikely to mistype an address without noticing.

For convenient import of the address with smartphones the addresses are often presented as quick response codes (see Figure 2.3).

An address in Base58Check: 1MDPuAy9WCbNQin71j9S3MKAAe9mGBRNVx.

Figure 2.3: The address 1MDPuAy9WCbNQin71j9S3MKAAe9mGBRNVx as a quick response code

2.4 Bitcoin Core

Bitcoin Core is derived from the original Bitcoin client, Bitcoin-Qt. It is maintained by the Bitcoin Core developers, a loosely organized group of volunteers. As there is no formal protocol specification yet, Bitcoin Core functions as the reference implementation of Bitcoin.

Bitcoin Core provides full node functionality: It synchronizes to the network's status quo from scratch by downloading and verifying the complete Bitcoin blockchain while enforcing all protocol rules. Bitcoin Core also has wallet capabilities, although many users have since migrated to thin-wallets, wallets that don't check the complete blockchain, such as MultiBit, Electrum, or mobile phone based wallets. Running a full node grants you the benefits of having a completely audited version of the Bitcoin network's state, which protects you from some vulnerabilities that other wallets (that don't fully check the blockchain) suffer from. In the standard configuration, Bitcoin Core will download, verify, and maintain the complete Bitcoin blockchain, but running it in a pruning mode with limited storage footprint is also possible. These pruned full nodes are just as secure, but cannot serve the complete blockchain.

Bitcoin Core used to include mining capability which has since been turned off by default, as CPU based mining has been long eclipsed by ASIC based mining.

3. Analysis

Coin selection describes the decision process of choosing UTXOs, or *"coins"*, as transaction inputs.

As described in the previous chapter, a wallet manages the private keys of its user, and thereby can track the associated UTXOs. This is the set of "coins" at the user's disposal, the UTXO pool.

The coin selection is part of the payment process. It starts out with a payment request put into the Bitcoin wallet. The payment request specifies an amount to be sent, the *spending target*[1]. As part of the generation of a transaction, the wallet must come up with a set of inputs to fund the transaction. The coin selection is hereby limited to draw from the wallet's UTXO pool and must select sufficient value to fund the transaction. It also considers secondary requirements such as privacy, fee costs, and its impact on other resources. Once a set of inputs is selected, the transaction is composed. Finally, the transaction will be broadcast to the network and wait for inclusion in a block for confirmation.

The prerequisites and constraints of the coin selection problem are explored in section 3.1. Section 3.2 discusses essays and reports about coin selection and UTXO set growth. In section 3.3 coin selection algorithms already implemented in prevalent cryptocurrency wallets are described, and the similarity of the coin selection problem and the subset sum problem are considered. The current coin selection implementation of Bitcoin Core is comprehensively described in section 3.4. Section 3.5 presents parameters and options to influence coin selection.

3.1 Constraints and Goals

3.1.1 Design Space for Coin Selection Strategies

First presented are the hard constraints that delimit the design space for coin selection:

[1] Throughout this work *target* or *spending target* is used exclusively to refer to the amount sent to the recipient.

- *The transaction must have sufficient funding.*
  So, in order to send X amount, the transaction inputs must be worth at least X.

- *The transaction's fee must suffice to achieve confirmation.*
  As blockspace is limited, miners get to select transactions for block inclusion. As transactions vary in data size, the miners prioritize transactions by *fee per byte* to maximize their revenue. Further, nodes no longer relay transactions without fees on the network in general.
  Therefore, transactions must be endowed at least with the minimum relay fee[2] to achieve confirmation. The necessary fee is not fixed, but can vary with the current demand. So, in order to send X amount, the transaction inputs must be worth at least $X + fee$.

- *All of the transaction's outputs must exceed the dust threshold.*
  "Dust" refers to transaction outputs that are less valuable than three times the mininum transaction fee and are therefore expensive to spend. Transactions creating dust outputs are not relayed by most nodes and not considered for confirmation by the majority of the miners.

- *Per protocol, a transaction may not exceed 1 MiB data size.*
  However, this restriction is of little practical importance for the analyzed transactions. Standard transactions usally come in at less than 1 kB.

3.1.2 User Goals

In addition to the hard constraints, there are user and global requirements for transactions.

Users are interested in minimizing transaction fees and confirmation times. Miners select transactions on the basis of their fee per data size (satoshi per byte). Therefore, it is cost effective for users to minimize transaction size. On the other hand, a higher fee will increase the likelihood of the transaction being included in the next block.

Many users aim to maximize their privacy. Addresses used together to fund a transaction are usually under control of the same entity. Therefore, users should use as few addresses together as possible. Users want their funds to be split between several UTXOs. If a wallet continuously merged all funds it contained, it would reveal the total balance and all economic activity to every trading partner of its user. Another user concern is resistance to data mining. Third party observers should be unable to tell which output is the change output and which output is the actual send.

3.1.3 Community Goals

On the other hand, from a global perspective it would be preferable if transactions used more inputs than the outputs they produce. Each UTXO used as an input shrinks the UTXO set, while each UTXO created as an output increases the UTXO

[2] This is a simplification as the *minimum relay fee* is an individual node setting.

set. As we can see in figure 3.1, the UTXO set has grown from about 200 MiB to 1 400 MiB in the past three years. Note that this is the storage footprint of the sequenced UTXO set on disk, unpacked into database format in memory it exceeds 4 GiB. The UTXO set is prerequisite to validating new transactions and blocks, so every full node on the network maintains a copy. Additionally, as new blocks are processed, the local copy of the UTXO set is updated to reflect which UTXOs were spent and added. The growth of the UTXO set is especially an issue for miners as they need to validate blocks as quickly as possible to commence work on the subsequent block. Retrieving uncached UTXOs from the sequenced disk copy is much slower than a lookup from the in-memory database. Every second of delay reduces their revenue. Hence, to be most competitive miners must minimize the look-up times for UTXOs and therefore keep the UTXO set in memory. For node operators slight delays in block verification are acceptable. however, the increased resource requirements are transcending cheap hardware such as RaspberryPis, which had been a popular choice to host nodes previously. An unchecked growth of the UTXO set could make home operation of a Bitcoin full node impractical[3] and threaten the decentralization of the network.

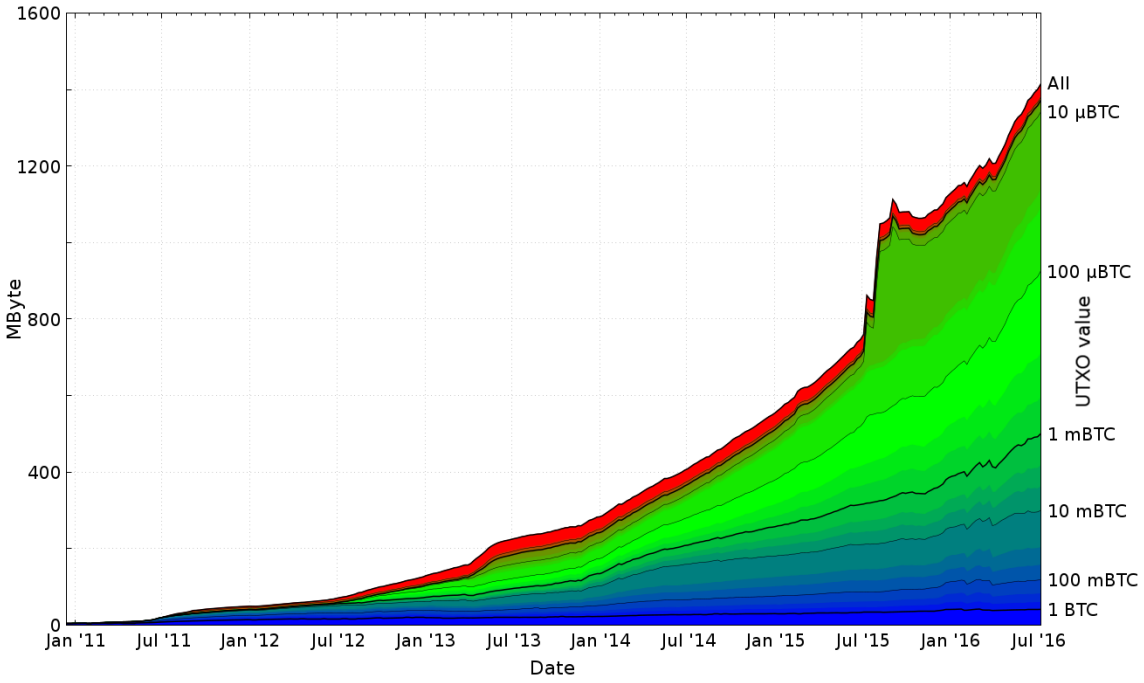

Figure 3.1: The storage footprint of the UTXO set over the past five years. Colors signify the value of the UTXOs [Wuil16]. The UTXO set has doubled in the last year, and grown sevenfold in last three years. Meanwhile, the average UTXO value has dropped from 0.97 BTC in 2014 to 0.38 BTC in 2016, as smaller UTXOs have grown faster than the larger.

3.1.4 Conflicting Goals

In the default transaction format P2PKH, the size of a transaction input is 148 bytes, while a transaction output is only 34 bytes. A transaction with a single input that

[3]Note that full nodes are not mining, but just maintain a fully verified copy of the blockchain. As full nodes enforce all rules of Bitcoin, they defend the network against mining nodes not complying to the rules.

sends to the recipient and creates a change output has $10+148+34+34 = 226$[4] bytes and a transaction of two inputs that foregoes creation of a change output would have $10 + 148 + 148 + 34 = 340$ bytes. Unfortunately, this causes wallet's optimizing for minimal cost per transaction to favor creation of more UTXO counter to the community goal of curbing the growth of the UTXO set.[5] Additionally, larger transactions increase the demand of blockspace, which is already a scarce resource. Therefore, transactions using more inputs don't only cost more, but additionally reduce the total transaction capacity. Still, the rampant growth of the UTXO set appears to be the most pressing issue.

3.1.5 Changing Conditions

Earlier, it was common for miners to select transactions by priority, a metric derived from the product of age and value of the spent UTXO and the transaction size (see 3.1).

$$\text{priority} = \frac{\sum_{inputs}(\text{input value}\,[satoshi] \times \text{input age}\,[blocks])}{\text{transaction size}\,[bytes]} \tag{3.1}$$

Example: A UTXO with 0.01 BTC value that was confirmed one day ago (144 confirmations) would weigh in for priority at 144,000,000. Another UTXO with 1 BTC value that was confirmed half an hour ago (3 confirmations) would weigh in with 300,000,000.

Transactions funded with larger and older UTXOs were favored, and there was a fixed minimal fee that was to be paid per rounded-up kilobyte.

As blockspace demand grew over the years, transaction fees slowly took on a bigger role in transaction selection. Up until February 2016, a portion of 50kB per block was reserved for high-priority transactions by default, allowing transactions to confirm just by merit of growing older. Priority-based transaction selection for mining was disabled by default in the release of Bitcoin Core v0.12.0 [Bitc16a]. Now, transactions are almost exclusively prioritized by their fee, and it is no longer feasible to send transactions without a fee based on merit of old aged UTXOs.

In the past years, demand of blockspace has caught up with the current data limit of 1 MiB per block. This has lead to fluctuating fee-levels per varying demand, and subsequently increased the importance of sane fee estimation mechanisms.

Fees have been rising over the past years, but it is hard to tell whether they will continue to rise. Even if they rise measured in purchasing power, they might decrease measured in bitcoins. If fees become more expensive (measured in Bitcoin) later, it would be advantageous to diminish the UTXO pool now while it is cheap. Vice versa, if fees decrease, it would be cheaper to defer spending of UTXOs until later.

However, as far as the author can tell neither of these changing conditions has had an impact on prevalent coin selection strategies being reworked.

[4] P2PKH transactions have a header of 10 bytes.

[5] Also, there isn't always a combination of UTXOs possible that omits change.

3.1.6 Individual Usage Patterns

Some service providers with high payment volume need to maintain a certain size of UTXO pool to have sufficient confirmed UTXOs at their disposal to fund outgoing payments. A coin selection policy that actively tries to minimize the UTXO footprint would be detrimental to their confirmation times.

Depending on your use-case, you might be receiving a multiple of payments than you send. For example, if you're a merchant, you might be receiving lots of small payments from your customers, but only occasionally make bigger payments to pay your suppliers. It would be a more convenient user experience then, if your wallet consistently used more inputs to create slightly bigger average transactions, than if it were to create small transactions most of the time, and enormous ones occasionally. On the other hand, you might top off your mobile wallet with a larger single payment to store some spending cash, but then spend it over the course of weeks until you receive another payment. In that case, you might experience delays when performing two payments shortly after each other, as the first transaction hasn't confirmed yet, and therefore its change output is equally still unconfirmed. Here, it might be interesting if the wallet were to maintain a minimum count of UTXOs, and when the first payment is performed with only the large UTXO in it, created multiple change outputs instead of only one.

Similarly, the average payment size may be very different. While one user's payments may average around five dollars value, another may be a traveling bitcoin enthusiast paying for flights and accomodation on a daily basis. In result, the two users would likely want to draw on very differently sized UTXOs to fund their transactions: when paying for a sandwich, it is harmful to reveal that your wallet contains at least a thousand dollars, yet to pay for a ticket by combining more than fifty UTXOs is also a privacy hazard. The size of the change outputs is under control of the sender and therefore may be the an effective tool to shape the composition of the UTXO pool. Perhaps, in the future, wallets can be equipped with a few settings to shape the employed coin selection policy to cater to individual needs. Until then, it seems unlikely that a single coin selection policy will fit all needs.

3.2 Related Work

To the author's knowledge no papers have been published on the topic of coin selection in Bitcoin. This section collects thoughts of established Bitcoin community members published on their personal websites or on the Bitcoin developer mailing list.

Maxwell has collected some thoughts about coin selection on a personal wiki site. He describes that improved coin selection could benefit user costs, transaction processing, privacy, and reduce blockchain bloat [Maxw12]. The text states that "Optimal coin selection is generally a mixed-integer non-linear program" and describes that "fee calculation is non-linear because the use of an input both contributes to the transaction's coin-days-destroyed as well as increasing the transaction size". This assertion may no longer be accurate as the miner's transaction selection has largely moved to a completely fee based prioritization. Maxwell provides a draft for an algorithm based on the above premise. Further, he proposes to use a Branch-and-Bound

algorithm to find an optimal solution. A Branch-and-Bound inspired coin selection approach was implemented for this work (see section 5.3).

Lopp has discussed the challenges of coin selection in [Lopp15]. Lopp describes four goals for an improved UTXO selection: 1) prevention of blockchain bloat, 2) support for high transaction volume, 3) privacy, and 4) cost minimization. His elaboration points out the contradicting nature of the goals. He establishes that using single UTXOs to fund transactions would minimize the fee costs, but generate a lot of small value UTXOs in the wallet. On the other hand, he points out how spending of small UTXOs, and avoidance to create them would help reduce the UTXO set. Lopp also mentions that entities with high transaction volume need to maintain a sufficiently large UTXO pool so that they can fund all transactions without relying on unconfirmed transactions. Further, he proposes that change output in the same size as the payment value would help obscure which output was assigned to the sender and to the recipient. A coin selection strategy that aims to create change output of at least the size of the spending target has been implemented for this work.

Stone published an essay on Bitcoin Core's transaction generation code in September [Ston16]. He was running experiments that required a large number of transactions to be generated and discovered that transaction generation slows down significantly when Bitcoin Core's UTXO pool contains more than 50 000 UTXOs. Stone implemented some improvements to the handling of data structures, but also analyzed the behavior of Bitcoin Core's coin selection algorithm. The essay notes that the first round of coin selection is started with the assumption of a fee of zero and will inevitably[6] fail to succeed, leading to a minimum of two rounds of selection. Stone further determines that Bitcoin Core's coin selection can end up with a valid selection but can still fail to recognize that it has succeeded (this point is discussed in detail in 3.4.2). He questions treating the coin selection as a knapsack problem and suggests to abolish the knapsack solver approach. Stone's proposed algorithm creates multiple solutions by combining UTXOs greedily and randomly. Finally, solutions are checked and the first valid solution is selected. Stone surmisses that his coin selection algorithm is about 200× more efficient computationally than Bitcoin Core's. Due to the only recent publishing of this proposal, Stone's algorithm has not been implemented for this work.

Todd states "that UTXO growth is a serious concern for Bitcoin's long-term decentralization" [Todd16]. Further, he highlights that the added latency of not keeping the UTXO set in memory will reduce the profit of a mining operations. Todd points out that maintaining a copy of the UTXO set is necessary to enforce the rules of Bitcoin, and thus a larger UTXO set increases the requirements for running a full node. In his report he elaborates that there are few incentives to spending UTXOs, let alone dust, to reduce the UTXO set. Additionally, there is some usage of the UTXO set as datastorage, for example to provide proof-of-existence services or publish PGP keys. Todd asserts that it is intolerable to let UTXOs expire, therefore a hard limit for the size of the UTXO set is not acceptable. He proposes to introduce *transaction output (TXO) commitments* as an additional data structure to archive

[6] Today, in general transactions without fee are neither relayed nor confirmed on the Bitcoin network.

older UTXOs and thus maintain a smaller effective UTXO set. TXO commitments appear to be mostly theoretical at this point.

Antonopoulos published an article on resource costs in Bitcoin [Anto16]. He points out that other than miners, non-mining nodes are not compensated for the cost of operation and thus a larger UTXO set is even more to their detriment. Antonopoulos suggests that fees provide a balancing mechanism to curb growth of the data traffic in Bitcoin. He proposes to evaluate transactions in the term of "Net-new UTXO", which is the difference of inputs and outputs of the transaction. "Net-new UTXO" were added as an evaluation metric in the simulation results.

A major reengineering of the transaction format, Segregated Witness [LoLW16] is about to be released to the main Bitcoin network. Segregated Witness will solve all known third-party transaction malleability. Currently, a transaction can be malleated by a third party and rebroadcast. The malleated version is a valid doublespend which however has a different TXID and therefore if confirmed invalidates follow-up transactions building upon the original transaction. Transaction malleability was most notably blamed to have contributed to the demise of MtGox, a major Bitcoin exchange. This attack-vector will no longer be possible with Segregated Witness transactions. Where currently services with high transaction volume are struggling to maintain a sufficient number of confirmed UTXO to fund transactions, in the future relying at least on the change outputs they generated themselves will be less risky. Segregated Witness also replaces the data size as a basis for transaction fee calculation with a "data weight". As the witness portion of the transaction can be pruned by nodes after verifying the transaction, a 75% discount is applied to it to calculate the weight. As this discount reduces the ratio of the transaction input size and the transaction output size from 148 byte and 34 byte to 68 byte (discounted) and 31 byte, spending of UTXOs in comparison to creating new UTXOs becomes significantly less costly. Segregated Witness complements improved coin selection.

The issue of having too few confirmed UTXOs available for spending could also be remedied by employing Opt-in Replace-by-Fee. Opt-in Replace-by-Fee is a recently added flag for transactions that marks them as non-final. This flag tells users not to rely on the transaction without confirmation[7]. Such non-final transactions can be useful to their sender, as the sender can broadcast an updated version for example with a higher fee, when a transaction didn't get selected for several blocks on the network. Usually, after a transaction was broadcast, the user would have to wait for the transaction to confirm, and the change output to become available for spending again. Instead, a transaction flagged as replaceable could be updated with additional outputs (and a smaller change output or additional inputs), to bundle more payments into an already broadcast transaction. While this would reduce the cost of the necessary transactions, the increased number of linked addresses should be considered a privacy concern. The usage of Opt-in Replace-by-Fee remains a rarity with in average less than 0.5% of the transactions per block opting-in [p2sh16].

[7] As explained before, Bitcoin transactions are only confirmed after included in a block. As the time between sending of a transaction and its confirmation can vary significantly, many payment processors accept transactions without confirmation if they consider the payment to have a very high chance of confirming.

3.3 Prevalent Coin Selection Policies

3.3.1 Highest Priority First

One of the most popular Bitcoin wallets for mobile phones, *Bitcoin Wallet for Android* relies on the *bitcoinj* library. Coin selection in bitcoinj is deterministic and relies on priority. As described in 3.1.5, priority is a metric derived from the age and value of UTXOs.

The main idea for this policy is that UTXOs are sorted by priority in descending order and selection pops from the front until sufficient value is reached. In case of matching priority higher value is preferred.

3.3.2 Oldest First

BreadWallet, a popular wallet for iOS and Android uses a pure *First In, First Out* (FIFO) policy for coin selection.

Breadwallet appends newly received UTXOs to the end of its UTXO pool. When funding transactions, Breadwallet selects the oldest UTXOs from the front of the UTXO pool until the target and a sufficient fee amount is reached. If the remainder after paying for target and fee exceeds the dust limit it is returned to the sender as a change output, otherwise the remainder is added to the fee [Brea16].

A similar policy is used by Electrum, a popular desktop wallet, by default.

3.3.3 Pruned Oldest First

Mycelium is a popular wallet for Android devices. It is developed in Java and makes use of the Bitlib library.

The coin selection in Mycelium[Myce16] follows a similar approach as BreadWallet and Electrum, but filters out the smallest UTXOs in a post-selection step:

1. Mycelium calculates a target on basis of the requested outputs.
2. It adds the oldest unselected UTXO.
3. The required fee is estimated on basis of an expected fee rate accounting for the currently selected number of inputs, the requested outputs, and one change output.
4. The fund selection stops when the selection's value exceeds target plus fee, otherwise it restarts from 2. by adding the oldest remaining UTXO.
5. In a second pass over the descendingly sorted selection, the minimal subset of UTXOs is found by pruning the smallest, unneeded UTXOs.
6. Finally, a change output is created if the remainder exceeds a fixed limit of 5 460 satoshi[8], otherwise the remainder is added to the fee.

[8]This fixed value is probably derived from the minimum dust limit of 546 satoshis.

3.3.4 Random Selection

Monero (XMR) is a competing cryptocurrency to Bitcoin. It forked from Bytecoin, which was a rewrite of Bitcoin, in April 2014 and is focused on privacy. Monero's coin selection is completely random, picking equiprobably from the UTXO pool until the transaction has sufficient funding [user16].

3.3.5 Target Sized Change

Additionally to the aforementioned default policy, Electrum also has a second coin selection policy focused on privacy. This "Private Mode" aims to minimize the value difference of target and change output. To that end, Electrum draws N sets of inputs randomly. From these sets, Electrum selects the one that minimizes $\delta(change, target)$.

3.3.6 Subset sum problem

This coin selection problem is related to the NP-complete *subset sum problem*. In the subset sum problem the goal is to find a set of N integers that sum up to a given target. Therefore, the problem is described with N decision variables which describe whether the integer at position n is included in the set, and P, the precision of the problem. If N is very small, exhaustive search becomes a practical solution. If the required precision P is small, a dynamic programming algorithm can solve the problem exactly. To our coin selection problem this translates as N being the size of the UTXO pool and the precision P follows from the coin selection approach. Depending on the state of Bitcoin Core and the coin selection approach, either N, P, or both may be small, and approximative subset sum solvers could provide an inspiration for coin selection strategies.

3.4 Bitcoin Core's Coin Selection

The author has first gotten interested in the topic of coin selection by trying to understand Bitcoin Core's coin selection algorithm to rectify degenerative behavior caused by a pathological UTXO pool. It has since come to light that few people understand Bitcoin Core's coin selection in detail, and that the thorough analysis conducted provides this chance to share insight. Perhaps this description will encourage the exploitation of the various improvement opportunities present in the algorithm.

3.4.1 Overview

Bitcoin Core's current coin selection consists of multiple steps and is focused on finding an *exact match* (see figure 3.2). An exact match here refers to finding an input set that is equal in value to the target. The benefit of an exact match is that it avoids the creation of a change output. If Bitcoin Core cannot find an exact match in several different attempts, it falls back to a knapsack solver which chooses an input set that minimizes the change output to a base of 0.01 BTC.

Although the algorithm is highly interwoven, its description will be partioned into three sections: Fee estimation, Exact match attempts, Knapsack selection.

Bitcoin Core offers a few additional features:

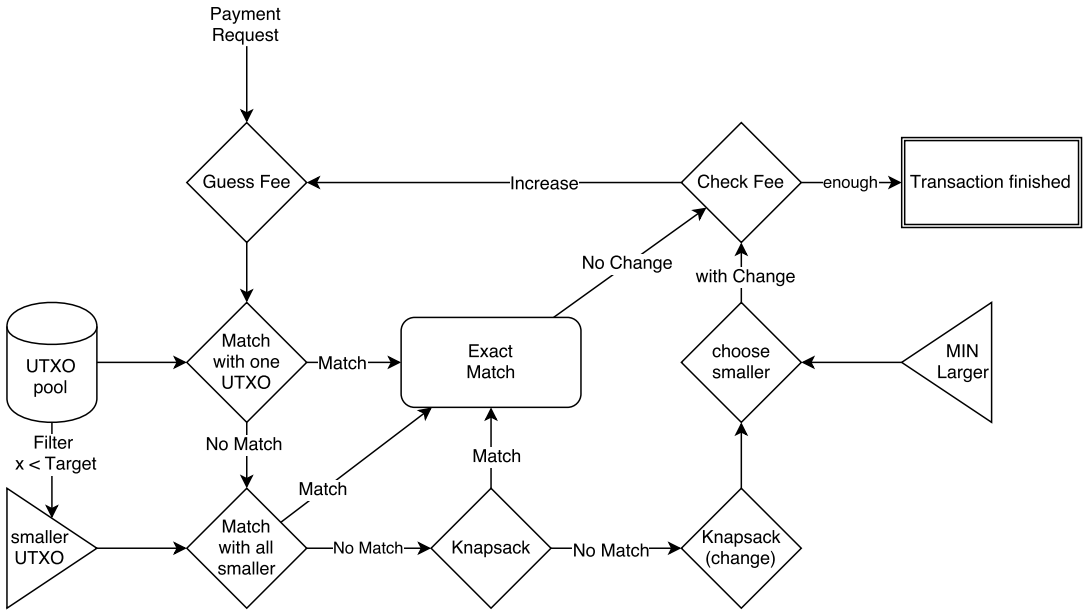

Figure 3.2: A partial flow chart of Bitcoin Core's multi-step coin selection process. – Bitcoin Core tries three different ways of creating an exact match, 1. with a single UTXO, 2. by combining all UTXOs smaller than the target[10], and 3. by performing a knapsack algorithm. Only then will it consider sets that would cause a change output. Finally, it will pick the smaller out of the knapsack result or the minimal larger UTXO. This will be repeated in loops for a fixed fee estimate that is increased in each cycle.

1. *CoinControl* – A feature which allows the user to manually designate inputs for transactions. Before the algorithmic coin selections starts, Bitcoin Core tests whether it can fund the transaction with the designated inputs. Otherwise, the coin selection will be performed with a target reduced by the value of the designated inputs, and the UTXO pool adjusted not to allow selection of the designated inputs.

2. *Multiple recipients* – Allows the user to set more than one recipient in transactions, but this work only regards the case of a single recipient.

3. *Manual fee* – Allows the user to manually set a fee per kilobyte.

4. *Fee deducted from sent amount* – The user may change the fee behavior, so that the fee is deducted from one or more recipient's outputs instead of the change output.

The further analysis and simulation will focus on the standard case of a transaction with a single spending output generated automatically. As none of the above mentioned features are used, they will not be further covered in this work.

3.4.2 Fee estimation

Before Bitcoin Core selects inputs, it will set a fixed fee estimate for each cycle. The spending target and the fee estimate are added to form an *adjusted target* for

[10]To be exact, they are filtered by (target + minimum change), but that's covered later in more detail.

the selection. Coin selection produces a candidate input set for the adjusted target. After the selection, Bitcoin Core calculates the minimum fee that would be required to get confirmation of the candidate transaction funded with the selected inputs. If this required minimum fee is lower or equal to the fee estimate, the algorithm has selected sufficient funds, and the transaction can go ahead. Otherwise, the fee estimate from this cycle will be used as the fee estimate for the next cycle. Starting the second cycle, the adjusted target will be derived from the previous adjusted target and the fee estimate.

There are a few issues here:

- *The first cycle is started with an estimate of a zero-fee.*
  Unfortunately, as mentioned in section 3.1, miners have not been confirming transactions with a zero-fee for some time, and most nodes don't even relay them to their peers. From today's perspective on the state of the network, this first cycle could be skipped, as a transaction without a fee would never be confirmed anyway.

- *Bitcoin Core could gauge the fee level just as easily before coin selection as after.*
  Then, leveraging the fact that inputs and outputs have fixed size, fees could be dynamically adjusted during the selection. It would be simple to reduce the coin selection to a single cycle that way.

- *Previous fee estimates are not predictive of the upcoming candidate input set.*
  If the first cycle finds a candidate input set with five inputs, the second cycle will end up paying for five inputs, even if the second candidate set only has two inputs. This causes fees to be frequently overestimated.

- *Fees are compounded over several cycles.*
  If a coin selection process needs three cycles or more to find a solution, fees are compounded but no longer counted towards the fee allowance. First, in variant (A) after selecting 3, 5, and 4 inputs in three computationally expensive cycles, Bitcoin Core's algorithm causes the user to pay for eight inputs when only four are used (see table 3.1). Second, in variant (B) after selecting 3, 5, and 6 inputs, the algorithm fails to recognize that it has found a valid candidate set because its current fee estimate only allows five inputs, and the candidate input set has six inputs. Even though it is paying for eight inputs already, the algorithm starts over into a fourth cycle, then allowing up to six inputs at the cost of fourteen. These issues have also been described by Stone [Ston16].[11]

Table 3.1 gives an example for a scenario in which the compounding fees occur to the user's detriment.

1. The coin selection procedure starts out with a fee estimate of zero and aims to discover a set that combines to the spending target. The resulting best candidate input set is composed of three inputs.

[11] The resulting monotonically nondecreasing fee estimate has the benefit that it guarantees the transaction generation's termination.

Cycle	Fee estimate	Adjusted Target	Candidates selected
1st	0 inputs	original target	3 inputs
2nd	3 inputs	original target + 3 inputs	5 inputs
3rd (A)	5 inputs	original target + *8* inputs	4 inputs
3rd (B)	5 inputs	original target + *8* inputs	6 inputs

Table 3.1: Example of two selection scenarios to exhibit the compounding fees. Costs of transaction header and transaction outputs are disregarded. Scenario (A) causes a transaction with four inputs to pay for eight inputs, variant (B) continues into a fourth cycle although a valid selection was found.

2. The fee estimate is updated for the second cycle to allow three inputs. The fee is added to the target to form an `adjustedTarget`. The resulting best candidate input set in the second cycle has five inputs.

3. The fee estimate now is set to five inputs, and this new estimate is added to the previous `adjustedTarget`. This causes the new `adjustedTarget` to pay for eight inputs, although the cycle will only accept solutions with up to five inputs as valid. If the third cycle finds a candidate set with four inputs (A), this is a valid solution, but the algorithm causes eight inputs to be paid where only four are used. On the other hand, if the third cycle finds a solution with six inputs (B), this is not recognized as a valid solution because it exceeds the limit of five inputs, although eight are already being paid for.

3.4.3 Exact match attempts

Bitcoin Core will try threefold to create an exact match, and check once whether an exact match is impossible.

1. *Exact match with a single UTXO*
   Bitcoin Core will pass through the UTXO pool and check if there is one that has the exact same value as the adjusted target (see algorithm 1). Again, the adjusted target here is derived from the spending target and the fee estimate of the cycle as described in 3.4.2. If a UTXO exists that matches to the satoshi, this UTXO will be returned as the candidate input set. No other selection steps will be performed unless a new cycle is started.

2. *Exact match with all smaller UTXOs*
   The UTXO pool is reduced to only the UTXOs that are smaller than (adjustedTarget + minimalChange[12]). This subset will be referred to as `smallerCoins` henceforth. If the sum of `smallerCoins` matches adjustedTarget, `smallerCoins` is returned as the candidate input set (see algorithm 2). No other selection steps will be performed unless a new cycle is started.

3. *Check if exact match is impossible*
   If the sum of `smallerCoins` is smaller than adjustedTarget, no exact match is possible. The smallest single UTXO that is greater than (adjustedTarget + minimalChange) is returned as the candidate input set (also see algorithm 2). No other selection steps will be performed unless a new cycle is started.

[12] minimalChange being 0.01 BTC in Bitcoin Core.

Algorithm 1: Match Single Check

Input: t, the adjusted target
Input: U, the UTXO Pool
Output: *bestSet*, with matching UTXOs or $\emptyset$

```
1  bestSet ← ∅
2  foreach u ∈ U do
3  |  if u.value = t then
4  |  |  bestSet.add(u)
5  |  |  break
6  |  end
7  end
8  return bestSet
```

Algorithm 2: Sum Of Smaller Checks

Input: t, the adjusted target
Input: U, the available UTXOs sorted by descending value
Output: *bestSet*, the selected UTXOs to fund the transaction or $\emptyset$

```
1  bestSet ← ∅
2  smallerCoins ← ∅
3  sum ← 0
4  foreach u ∈ U do
5  |  if u.value < t + 0.01 BTC then
6  |  |  sum ← sum + u.value
7  |  |  smallerCoins.add(u)
8  |  end
9  end
10 if sum < t then
11 |  minGreater ← ∞
12 |  foreach u ∈ U do
13 |  |  if (u.value > t) ∧ (u.value < minGreater) then
14 |  |  |  minGreater ← u.value
15 |  |  |  bestSet ← {u}                    // return minimal greater UTXO
16 |  |  end
17 |  end
18 else if sum = t then
19 |  bestSet ← smallerCoins
20 end
21 return bestSet
```

4. *Attempt to find an exact match by combining UTXOs from* `smallerCoins`
   Bitcoin Core runs the knapsack selection (see 3.4.4 trying to find an exact match. If an exact match is discovered, this combination is returned as the candidate input set. No other selection steps will be performed unless a new cycle is started. Otherwise, no exact match was found and the algorithm will fallback to creating a transaction with a change output.

3.4.4 The knapsack selection

The knapsack algorithm 3 performs up to one thousand selections on `smallerCoins` to find the input set with the smallest excess over the given adjustedTarget. In the first run of the knapsack algorithm for the exact match, `adjustedTarget` was the sum of spending target and fee estimate. In this second run, the knapsack algorithm's `adjustedTarget` is set to the sum of spending target, fee estimate, and minimal change.

For each of the 1 000 tries, the solver traverses the descendingly sorted `smallerCoins` in two passes. In the first pass, it selects each UTXO with a 50% chance. Once the addition of a UTXO causes the sum of the currently selected UTXOs to exceed the adjustedTarget, a candidate input set is found. If the current candidate input set is smaller than the previous smallest, it is stored as `bestSet`. The solver doesn't break the try then though, but rather it attempts to replace the last added UTXO with a smaller one: The last selected UTXO gets deselected, again dropping the sum of the currently selected below the adjustedTarget and the traversal continues. As before, newly added UTXOs that don't lift the sum over the adjustedTarget remain in the selection. This allows for following smaller UTXOs to combine with the remaining set to produce an even smaller sufficient set. If the first pass produced any valid solution, its `bestSet` is returned as the candidate input set.

If the first pass didn't find a solution, the second pass is performed. In the second pass, every UTXO that is not yet selected gets added to the set. Again, after each addition, when the target is surpassed, new best candidate sets get stored, and the last addition is removed from the set to try for a smaller solution (see algorithm 3).

3.4.5 Rounding up the coin selection

There are more corner cases covered in Bitcoin Core's algorithm, such as the cases when the payment request exceeds the wallet funds, the fee cannot be afforded for the requested amount, or the difference between requested amount and the wallet funds is smaller than`minimumChange`. Also, Bitcoin Core's coin selection will first restrict itself to UTXOs that have at least one confirmation if sent by yourself, or six confirmations if received from another wallet, then relax these requirements in two further passes if no suitable candidate input set could be selected. These points exceed the scope of this description and will be omitted for sake of brevity.

From the previous steps, we now have a candidate input set that is either an exact match or that minimizes the change over the mininum of 0.01 BTC. Finally, this candidate input set will now be compared to the smallest UTXO that surpasses the sum of `adjustedTarget` and `minimumChange` (see algorithm 4).

Looking at the complete coin selection algorithm[13], a few theories emerge:

[13] A few examples for the results of Bitcoin Core's coin selection are provided in section 3.4.6.

Algorithm 3: Knapsack selection

Input: t, the adjusted target
Input: U, smallerCoins sorted by descending value
Output: *bestSet*, the selected UTXOs to fund the transaction or $\emptyset$

```
1  bestSet ← ∅
2  bestSetValue ← ∞
3  selectionSum ← 0
4  targetReached ← false
5  for i ← 1..1000 do
6      for (pass ← 1..2) ∧ ¬targetReached do
7          foreach u ∈ U do
8              if
                ((pass = 2) ∧ unselected(u)) ∨ ((pass = 1) ∧ (binaryRandom() = true))
               then                                                  // 50% chance
9                  selectionSum ← selectionSum + u.value
10                 selectedUTXOs.add(u)
11                 if selectionSum = t then
12                     return selectedUTXOs
13                 end
14                 if selectionSum > t then
15                     targetReached ← true
16                     if selectionSum < bestSetValue then
17                         bestSet ← selectedUTXOs
18                         bestSetValue ← selectionSum
                           /* Deselect last addition and try for better
                              combinations                                */
19                         selectionSum ← selectionSum − u.value
20                         selectedUTXOs ← selectedUTXOs \ u
21                     end
22                 end
23             end
24         end
25     end
26 end
27 return bestSet
```

- It seems that the algorithm heavily biases towards the largest UTXO that is smaller than the target. This is suggested by 50% of the tries adding it to the candidate set in the first pass and keeping it in the set, since it alone cannot exceed the target and thus will not be removed after being selected. It is doubtful that the knapsack algorithm efficiently discovers potential exact matches, yet traversing the UTXO pool 1 000 times in this manner is computationally expensive.

- After the first element, the knapsack appears to iteratively bias towards the largest UTXO that fits in the remainder created by $target - selected$. This repeats for each "privileged" input that doesn't cause the set to go beyond the

Algorithm 4: Coin Selection

Input: t, the adjusted target (incl. fee estimate)
Input: U, the available UTXOs sorted by descending value
Input: mc, the minimumChange
Output: *bestSet*, the selected UTXOs to fund the transaction

$bestSet \leftarrow U$
if $MatchSingleCheck(t, U) \neq \emptyset$ **then**
  $bestSet \leftarrow MatchSingleCheck(t, U)$ `// see alg 1`
else if $SumOfSmallerChecks(t, U) \neq \emptyset$ **then**
  $bestSet \leftarrow SumOfSmallerChecks(t, U)$ `// see alg 2`
else
  $bestSet \leftarrow KnapsackSelection(t, smallerCoins)$ `// for match, see alg 3`
  if $bestSet.value > t$ **then**
    $bestSet \leftarrow KnapsackSelection(t + mc, smallerCoins)$ `// for change`
end
$minSingleUTXO \leftarrow min(u \in U; u > t + mc)$
if $minSingleUTXO.value < bestSet.value$ **then**
  $bestSet \leftarrow \{minSingleUTXO\}$
end
return *bestSet*

target and therefore gets to remain in the set. The algorithm thus may select ever smaller UTXO to just barely not fill the remaining gap to the target. In the presence of a large quantity of minuscule UTXOs, this can lead to a vast number of inputs. For example, an incident was reported where 586 inputs were selected to create a transaction [Moor12].

- The algorithm appears to quickly decimate small UTXOs, maintaining a small UTXO footprint, but on the other hand, also deprives itself of the broad choice of values necessary to find exact matches.
- The knapsack algorithm is called much more often than necessary. First, when the algorithm attempts to create a zero-fee transaction, it can never produce a valid result.
- Bitcoin Core's uncompromising definition of an exact match significantly increases the computational expense of the knapsack selection. Allowing a small allowance instead of the zero satoshi difference could make exact match much more likely to occur, which in turn stops the knapsacking early. By dropping this small excess to the fee, money could be saved by circumventing the creation of a change output.
- The minimum change of 0.01 BTC appears to be a magic number with unknown justification for its value. It appears likely that a dynamic level for the minimum change or another fixed value could perform better at least in some usage scenarios.

The following section provides some examples, in case the reader wants to get a better intuition for the results of Bitcoin Core's coin selection process in practice.

3.4.6 Examples for Bitcoin Core's Coin Selection

The examples disregard fees for simplicity's sake[14].

Alice has four UTXO: 0.1 BTC (A) , 0.3 BTC (B), 0.5 BTC (C), 0.85 BTC (D)

Example 1: Alice wants to send 0.3BTC.
Bitcoin Core discovers that (B) matches the target, and it only uses (B) as input.

Example 2: Alice wants to send 0.4BTC.
Bitcoin Core finds that the sum of all UTXO smaller than the target (i.e. (A) + (B) = 0.1 + 0.3 = 0.4) matches the target here. (A) and (B) are used as inputs.

Example 3: Alice wants to send 0.45BTC.
Bitcoin Core finds that (C) is the smallest UTXO greater than the target, and that the sum of all UTXOs smaller than the target (i.e. (A) + (B) = 0.1 + 0.3 = 0.4) does not surpass the target. (C) is used as the sole input, being the next smallest input greater than the target.

Example 4: Alice wants to send 0.35BTC.
Bitcoin Core finds that (C) is the smallest UTXO greater than the target, and that the sum of all UTXOs smaller than the target (i.e. (A) + (B) = 0.1 + 0.3 = 0.4) does not match the target. It performs the `knapsackSelection` to find an exact match, but doesn't succeed, then runs `knapsackSelection` again to find the smallest combination with change output. The smallest sufficient combination is then compared with the smallest single input greater than the target. Assuming that it does find the best combination here which would be (A) + (B), it finds that target < (A) + (B) < (C) and uses (A) and (B) as inputs.

Example 5: Alice wants to send 0.6BTC.
Bitcoin Core finds that (D) is the smallest UTXO greater than the target, and that the sum of all UTXO smaller than the target (i.e. (A) + (B) + (C) = 0.1 + 0.3 + 0.5 = 0.9) does not match the target. It starts running `knapsackSelection` as before, and will find that (A) + (C) = target. As it finds a combination that matches the target, it breaks and immediately goes with that combination. (A) and (C) are used as inputs.

Example 6: Alice wants to send 0.825BTC.
Bitcoin Core finds that (D) is the smallest UTXO greater than the target, and that the sum of all UTXO smaller than the target (i.e. (A) + (B) + (C) = 0.1 + 0.3 + 0.5 = 0.9) does not match the target. It starts trying knapsacking, and after two runs of 1 000 tries comes up with (A) + (B) + (C) as the best candidate set. As (D) is bigger than the target and smaller than (A) + (B) + (C), (D) is used as the sole input.

3.5 Ideas for Improvement of Coin Selection

Drawing on the above analysis there are a number of approaches that could yield improvements in the coin selection process. The following section will give an overview.

[14]The examples are based on the author's previous work [Erha14].

3.5.1 Improved randomness of drawing

As described in section 3.4, Bitcoin Core's current selection strategy can lead to degenerative behavior in the presence of large quantities of minuscule UTXOs, and it appears to not search all possible candidate sets with uniform probability. Selecting randomly from the UTXO pool seems an obvious strategy that avoids such degenerative behavior. Since the likelihood to select a greater number of inputs would be higher when many small UTXOs are available, and lower when funds are concentrated in fewer large UTXOs, it might also perform well to regulate the UTXO footprint. Additionally, random selection should leak little information, as it avoids obvious patterns and the UTXO pool from which the UTXOs are picked is not known to observers.

3.5.2 Change output matches spending target in size

Bitcoin Core's current selection strategy aims to create no change output or to minimize change output to a base value of 0.01 BTC. This allows data miners an easy educated guess as to which one of the (usually) two transaction outputs is the sent amount and which one is the change output. Similarly, the change output can often be guessed in the other presented policies: if one of the outputs is smaller than any of the inputs when there is more than one input, this output will almost certainly be the change output. Otherwise, the other inputs would not have been necessary for funding the payment. To obscure the destination of the outputs, coin selection could aim to create change outputs that are of the same size as the target (as Electrum Private Mode does, see section 3.3.5). This would improve user privacy and might create larger change outputs in average, additionally reducing the UTXO footprint. Under the assumption that users tend to create transactions of similar value, UTXOs in size of the spending target could be more useful to fund later transactions than minimized change outputs.

3.5.3 Change output matches average spending target in size

Developing further the previous assumption that users tend to create transactions of similar value, UTXOs in size of the *average* spending target could be more useful to fund later transactions than minimized change outputs. Also, if beneficial, this would adjust to the individual usage pattern. However, if it were known that this strategy is being employed, it might become very easy to tie different transactions of the same wallet together on basis of the change outputs' sizes.

3.5.4 Pruning of selected set

While pruning could help to minimize short term fees, it does cause small UTXOs to have a much longer retention time. This effect has already been shown by the addition (and subsequent later removal) of pruning to Bitcoin Core's algorithm in a previous attempt to improve coin selection. As pruning also is the only difference between the two oldest first coin selection policies, the previous results can be checked by looking at their respective simulation results. Therefore, it is dismissed as a potential improvement for Bitcoin Core's coin selection.

3.5.5 Donate minuscule remainders to fee instead of creating change outputs

When encountering a tiny remainder after coin selection, it may be economically beneficient to add it to the transaction fee instead of creating a change output. In the case that the discarded amount were less than the cost of creating a change output, the user actually would save costs by paying more fees. Therefore, the transaction becomes more attractive for miners to select due to the higher fee.

3.5.6 Require fixed minimum number of inputs for each transaction

It seems obvious that the UTXO set would decrease in size if transactions by default used a bigger number of inputs. However, strictly employing such a policy could quickly reduce the UTXO pool to a single UTXO, which would be detrimental: every recipient would become aware of the user's full balance, and all the user's remaining money would be tied in an unconfirmed change output after each payment. Such a policy would then require augmentation with a boundary below which it is turned off, or perhaps a mininum input set logarithmically scaling with the UTXO pool's size. Altogether, the approach seems unattractive from a user's point of view with regard to privacy and transaction cost, as well as hard to get right to make a "one size fits all" solution.

3.5.7 Require number of inputs greater or equal than number of outputs for each transaction

As shown, the totality of transactions in the Bitcoin network creates more UTXOs than it consumes. Hence, a policy that each transaction should use at least as many inputs as the number of outputs it creates could lead to at least a reduction of the UTXO set growth. However, this might not suffice if the user receives more payments than he sends, as well as diminish individual's wallet's UTXO pool too much, if the opposite is true.

3.5.8 Spend all UTXOs from the same address

As addresses have been omitted in this analysis for simplification, this approach is mentioned for completeness only. In cases where address reuse occurs, it would be preferable to select all UTXOs associated with that address whenever any one of them is selected. Otherwise, future transactions using remaining UTXOs associated with this address will clearly link the transactions. Spending all UTXOs at once minimizes the number of transactions associated with the address, even if the address remains in use and receives payments at a later date again.

3.5.9 Split large outputs for privacy

In cases where a very large input is selected it would increase the user's privacy to split the change output into a several change outputs. For example, one change output could match the size of the target, and another two could halve the remainder, or all of the change output outputs could all be different sizes. As the model in this work disregards addresses and so far only covers transactions with a single change output per transaction, this approach will be shelved until the model can be used to explore privacy.

3.5.10 Purposeful search for exact matches

Exact matches seem very beneficial all around: Not creating a change output saves fees, diminishes the UTXO pool, and cuts off linkage to other transactions from your wallet. The Pruned Oldest First policy may get a direct match by minimizing the selection sometimes, and it seems a design goal of Bitcoin Core. However, the latter's algorithm doesn't seem to effectively address this design goal. A two-staged algorithm that purposefully searches for an exact match in the first stage, and follows up with a fallback that maintains a broad selection of varying UTXO sizes in the UTXO pool could perhaps do very well.

3.6 Summary

This chapter introduced the design space for coin selection. While there are few hard constraints, the requirements are somewhat contradictory, and can further vary between individual use-cases. Although the transaction market has changed the paradigm, the discussed wallets' policies remain largely unchanged. Bitcoin Core's coin selection has been comprehensively described, and multiple improvement opportunities have been identified. Some approaches for improvements on Bitcoin Core's policy have been discussed.

4. Design and Implementation of the Simulation Framework

Coin selection had been discussed previously in the Bitcoin scene, but to the author's knowledge there has been no previous tool to analyze and compare different coin selection strategies.

The goal of this framework is to replace speculation about the merits of various coin selection strategies with plausible simulation results. As doing large amounts of coin selection on a real network causes unnecessary overhead, this framework restricts itself to simulating the progression of the UTXO pool through various scenarios. However, the simulation should adhere as closely to the real-world situation as possible. Therefore, the provided simulation framework closely models Bitcoin wallet behavior, including UTXO management, change outputs, transaction fees, and confirmation age of UTXO.

Yet, this framework cannot model the full complexity of the real-world network: The framework does not model blockspace availability, nor the resulting delay of transaction confirmation or changes in fee levels. While transactions can be assigned to a specific block height by the scenario, due to a lack of this information, this has been used to randomly skip a few blocks between transactions. The simulation does not try to model the diverse transaction composition with Pay-to-Script-Hash (P2SH) or transactions with multiple outputs. Instead, all transactions are assumed to be P2PKH with only one recipient output and an optional change output. The simulation also forgoes exploration of address reuse.

4.1 Simulation

4.1.1 The `Scenario` class

The `Scenario` class defines the initial UTXO pool and a sequence of `Payment`s to be performed. Each `Payment` has a unique id, a transacted value and a blockchain height at which it should be executed. Positive transacted values mark incoming payments, while negative transacted values are outgoing payments.

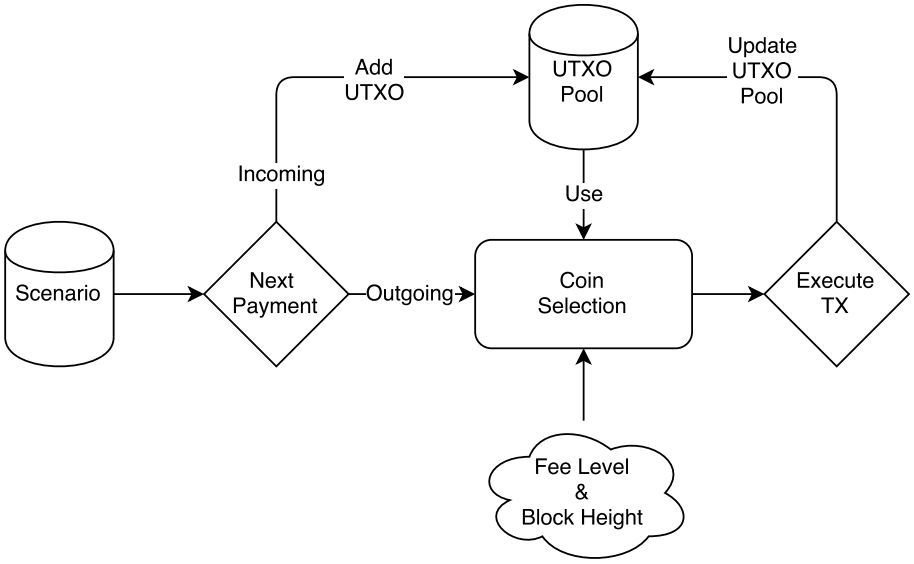

Figure 4.1: Overview of the Simulation: `Scenario` provides `Payments` which for an incoming payment add a `UTXO` to the UTXO pool, and for an outgoing payment trigger coin selection and an update of the UTXO pool.

4.1.2 The `Simulator` class

`Simulator` manages the simulated `Wallet`s, on basis of a `Scenario`. Each `Wallet` is set up with the initial UTXO pool, then each `Payment` from the `Scenario` is executed in parallel on each `Wallet`. Since scenarios may be based on random data, it is possible that outgoing payments request to spend more than is available. To mitigate this, outgoing payments are queued and only executed when all `Wallet`s have a sufficient balance.

4.1.3 The `Wallet` interface

Coin selection is simulated on implementations of the `Wallet` interface. The `Wallet` has a minimal UTXO management capability. At the start of the scenario, it can have an empty or given UTXO pool.

`Wallet` keeps track of the UTXO pool, and offers two functions for calls from `Simulator`: `Wallet.receive(value, nLockTime)` and `Wallet.spend(target, nLockTime)`. When an incoming payment is received, the `Wallet` merely adds a new `UTXO` to its UTXO pool. The `UTXO` is marked as created at the blockheight specified in `nLockTime`. All implementations inheriting from `Wallet` behave the same way for incoming payments.

When an outgoing payment is requested, `Wallet.spend(target: Long, nLockTime: Int)` is called. `target` here represents the amount of satoshis to be spend and `nLockTime` refers to the minimum height at which the transaction may be performed. Subsequently, the specific implementation of `Wallet.selectCoins(target: Long, feePerKB: Long, nLockTime: Int)` is performed. This includes checking adherence to fee requirements. When the coin selection is concluded, `Wallet.executeTransaction(target: Long, selection: Set[UTXO])` removes the selected set of `UTXO`s from the UTXO pool. If the coin selection results in a change

Algorithm 5: class Wallet

Input: name: String
Input: initialUtxoPool: Set[UTXO]

```
1  def receive(outputValue: Long, nLockTime: Int):
2  |   utxoPool ← utxoPool + new UTXO(outputValue)
3
4  def spend(target: Long, nLockTime: Int):
5  |   selection ← selectCoins(target, feePerKB, nLockTime)
6  |   executeTransaction(target, selection)
7
8  def selectCoins(target: Long, feePerKB: Long, nLockTime: Int):
9  |   ... Implementation-specific Coin Selection...
10 |   return selectedCoins
11
12 def executeTransaction(target: Long, selection: Set[UTXO]):
13 |   foreach utxo ← selection do
14 |   |   utxoPool.remove(utxo)
15 |   end
16 |   if getTotal(selection) > target + MIN_CHANGE then
17 |   |   change ← getTotal(selection) - target
18 |   |   receive(change, nLockTime)
19 |   end
20
```

output, the change output is added to the UTXO pool by calling `Wallet.receive(changeValue: Long, nLockTime: Int)` just as for an incoming payment.

Bitcoin Core's current coin selection, coin selection solutions of other prevalent wallets, and the proposed approaches were implemented in Scala.

4.1.4 `UTXO` model

Algorithm 6: UTXO class

```
1 class UTXO(uniqueID: Int, value: Long, height: Int)
```

The class `UTXO` simplifies UTXOs to only a unique identifier, the UTXO's value, and the blockchain height that the UTXO was created at, see Algorithm 6. The Scala *Int* being a 32-bit signed value, and bitcoins being divisible to the eighth decimal place would only allow expressing values up to 21.47483647 bitcoins with *Int*. Therefore, the value is instead stored as a *Long*, a 64-bit signed value.

Transaction outpoints, output script, and other more advanced features are disregarded for this simulation.

The identifier `UTXO.uniqueID` is provided by `Wallet` by means of a counter that is incremented for each new UTXO received. `UTXO.value` represents a number of satoshis.

4.1.5 Scenario Conclusion

As the scenario is concluded, each wallet model has a distinctly composed UTXO pool whose sum adds up to roughly the same amount. The final UTXO pool is written to the log for each `Wallet`. The simulation also creates a comma-separated-value table that tallies up interesting statistics. These include:

- count of remaining UTXOs in the UTXO pool after the scenario
- the mean count of UTXOs in the UTXO pool during the scenario
- count of UTXOs received throughout the scenario
- count of UTXOs spent from the UTXO pool
- count of change outputs created
- smallest/biggest/mean change output value
- standard deviation of change output value
- fees paid for all transactions
- cost to spend all UTXOs remaining in the UTXO pool
- minimum/maximum/mean # of UTXOs selected for input
- standard deviation of input set size

4.2 Summary

The simulation framework reduces the payment process to an isolated wallet. Wallets can receive payments, adding a UTXO to their UTXO pool, and send payments. Sending payments first performs coin selection, taking into account fees and block-height, then removes the selected UTXOs from the UTXO pool and adds a change output back when applicable. Scenarios provide a list of incoming and outgoing payments to be performed on the simulated wallets. The simulation creates an output file with statistical data relevant for the evaluation.

5. Design and Implementation of New Coin Selection Policies

This chapter will describe the implementation of ideas described in section 3.5, and describe the design and implementation of the Branch and Bound algorithm.

5.1 Bitcoin Core Variants

Two design decisions of Bitcoin Core's coin selection are investigated: The fixed value minimum change in two coin selection variants, and the uncompromising zero satoshi difference to achieve an exact match.

5.1.1 Double Target

As explained in section 3.4.5, Bitcoin Core uses a fixed minimum change of 0.01 BTC. Change outputs are one of the few tools the spending party has to influence the composition of their UTXO pool. It was theorized that creating change in the size of the target would be beneficial as it would make UTXOs available in case that a similar spending target occurs.

Double Target replaces the fixed mininum change with the value of the spending target of the current payment request. The updated coin selection algorithm (see algorithm 4 for Bitcoin Core's) can be seen in algorithm 7. There are four effects of this change:

1. The original function's input for the minimum change `mc` is omitted.
2. The `smallerCoins` set is now created by filtering the UTXO pool with $utxo.value < 2 \times t$ instead of the previous $utxo.value < t + 0.01\,BTC$ (see line 5 of algorithm 2).
3. The second round of `KnapsackSelection` (line 8) is called with the target of $2 \times t$ instead of the previous $t + mc$.
4. `minSingleUTXO` must be greater than $2 \times t$ now instead of $t + mc$ (line 9).

Algorithm 7: Double Target Coin Selection

Input: t, the adjusted target (incl. fee-estimate)
Input: U, the available UTXOs sorted by descending value
Output: *bestSet*, the selected UTXOs to fund the transaction

$bestSet \leftarrow U$
if $MatchSingleCheck(t, U) \neq \emptyset$ **then**
  $bestSet \leftarrow MatchSingleCheck(t, U)$ `// see alg 1`
else if $SumOfSmallerChecks(t, U) \neq \emptyset$ **then**
  $bestSet \leftarrow SumOfSmallerChecks(t, U)$ `// see alg 2`
else
  $bestSet \leftarrow KnapsackSelection(t, smallerCoins)$ `// for match, see alg 3`
  if $bestSet.value > t$ **then**
    $bestSet \leftarrow KnapsackSelection(2 \times t, smallerCoins)$ `// for change`
end
$minSingleUTXO \leftarrow min(u \in U; u > 2 \times t)$
if $minSingleUTXO.value < bestSet.value$ **then**
  $bestSet \leftarrow \{minSingleUTXO\}$
end
return *bestSet*

5.1.2 Average Target

Average Target also changes the minimum change. Instead of Bitcoin Core's fixed value, the mean spending target of the previous outgoing payments is used as minimum change. The `Wallet` is extended to keep track of the previous spending targets and their count, in order to calculate the average stably. Besides that, the same lines of code are affected as with Double Target.

5.1.3 Wider Match Donation

As discussed in section 3.4.5, Bitcoin Core requires a difference of zero satoshi to recognize a candidate input set as an exact match. Exact matches save the user money though, they omit creation of a change output which the user would then need to spend at a later date. Thus, instead of allowing no difference at all, *Wider Match Donation* allows a difference up to the dust limit for an exact match. This range is allowed for all instances in which Bitcoin Core attempts to produce exact matches (see section 3.4.3):

1. Exact match with a single UTXO
2. Exact match with all smaller UTXOs
3. During the `KnapsackSelection`

As no change output is produced when an exact match occurs, the difference between spending target and the sum of the selected inputs is added to the fee.

5.2 Single Random Draw

Deterministic selection algorithms can leak private information, such as the age of the oldest UTXO in the UTXO pool. On the other hand, Bitcoin Core's algorithm appears to bias towards spending specific UTXOs during one transaction generation, and can construct extremely large input sets under presence of many small UTXOs in its pool. Selecting randomly on the UTXO pool is an obvious naïve approach, that was not present among the examined prevalent coin selection policies. In the presence of few small UTXOs, the spending target can be expected to be reached with a small input set, vice versa will the chance increase to select some small UTXOs if there are more present. Therefore, *Single Random Draw* can be expected to have good privacy properties, as the UTXO pool is not known to observers, should self-balance its UTXO footprint to an extent, and should produce similarly stable input set sizes as e.g. the FIFO policy.

Algorithm 8 shows the logic necessary for the `selectCoins()` function of this policy.

Algorithm 8: Single Random Draw

Input: t, the spending target
Input: U, the available UTXOs
Input: mc, the minimum change
Output: *selectedCoins*, the selected UTXOs to fund the transaction

$selectedCoins \leftarrow \emptyset$
$shuffledPool \leftarrow randomize(U)$
while $sum(selectedCoins) < (t + estimateFee(selectedCoins) + mc)$ **do**
    $selectedCoins.add(shuffledPool.head)$
    $shuffledPool \leftarrow shuffledPool.tail$ `// drop head`
end
return *selectedCoins*

As can be seen, the algorithm shuffles the UTXO pool, and then selects UTXOs from the front of the queue until spending target, fee requirement, and minimum change are satisfied. The mininum change `mc` can be set to zero or more. When the resulting change output is smaller than the dust limit, it will be added to the fee instead of being paid to the sender.

In production code, this algorithm would need to be extended to mitigate pathological cases. For example a UTXO pool consisting of one UTXO with sufficient value to pay for the transaction among 10 000 UTXOs worth less than the cost of one transaction input.

5.3 Branch and Bound

After some initial experiments, the benefits of exact matches became more palpable. The following algorithm was designed to purposefully find exact matches, to be less computationally expensive than Bitcoin Core's coin selection policy, and to be easy to implement. The study of the subset sum problem was an inspiration as it describes a similar combinatorial problem.

The following were the two main ideas:

1. *Effective value of UTXOs*
   The implementation of Bitcoin Core's `KnapsackSelection` only considers fees between cycles. As the selection there does not adapt the fee estimation during selection, selection results are often invalid. However, the required fee level can be predicted before the selection just as easily as after selection. Since input scripts and output scripts have fixed data sizes, the cost per input and output are fixed known values within one payment request.

   It is thus a simple matter to calculate the *effective value*, i.e. the contribution of a UTXO towards the target, for the current payment request (see equation 5.1).

$$\text{effectiveValue} = \text{utxo.value} - \text{feePerByte} \times \text{bytesPerInput} \tag{5.1}$$

   As an example the effective value of one bitcent (= 0.01 BTC = 100 000 satoshi) at a fee level of 10 000 satoshi per kilobyte would be 98 520 satoshi (see equation 5.2).

$$98\,520\,\text{satoshi} = 100\,000\,\text{satoshi} - 10\frac{\text{satoshi}}{\text{byte}} \times 148\,\text{bytes} \tag{5.2}$$

   Bitcoin Core is attempting to pack a knapsack whose size is changing while it is being packed [Ston16]. *Effective values* allow reframing of the problem such that the target remains fixed. This is much more akin to the subset sum problem, but still slightly different as a small excess is acceptable.

2. *Efficient search for exact matches*
   Instead of restarting the search multiple times, and repeatedly exploring the same combinations, the combination space of the *effective values* can be searched exhaustively once with less effort. To that end, a binary tree is constructed where each level relates to the inclusion or omission of a single UTXO. The UTXOs are assigned in descending order by value (see figure 5.1).

This tree is searched in a depth-first-search by randomly expanding either the inclusion or omission path first. Paths whose sum exceeds the target are cut and not explored further. As the cost saved by not creating a change output is one less output in this transaction and one less UTXO to be spent in the future, a range of `costPerInput + costPerOutput` is allowed for exact matches. The search stops when the first exact match is found.

As the search tree grows exponentially with the number of elements in the UTXO pool, a limit has been added to the search for the number of combinations that get tried. If there is only one combination that produces an exact match, it should be found eventually, unless the algorithm reaches the limit first, but if there are multiple combinations hidden in the tree, the search will find them in different order at random.

If no match is possible, or the search limit is reached, the algorithm falls back to Single Random Draw (see section 5.2). Single Random Draw was chosen as it should provide a diverse set of change outputs which should be useful to find more exact matches.

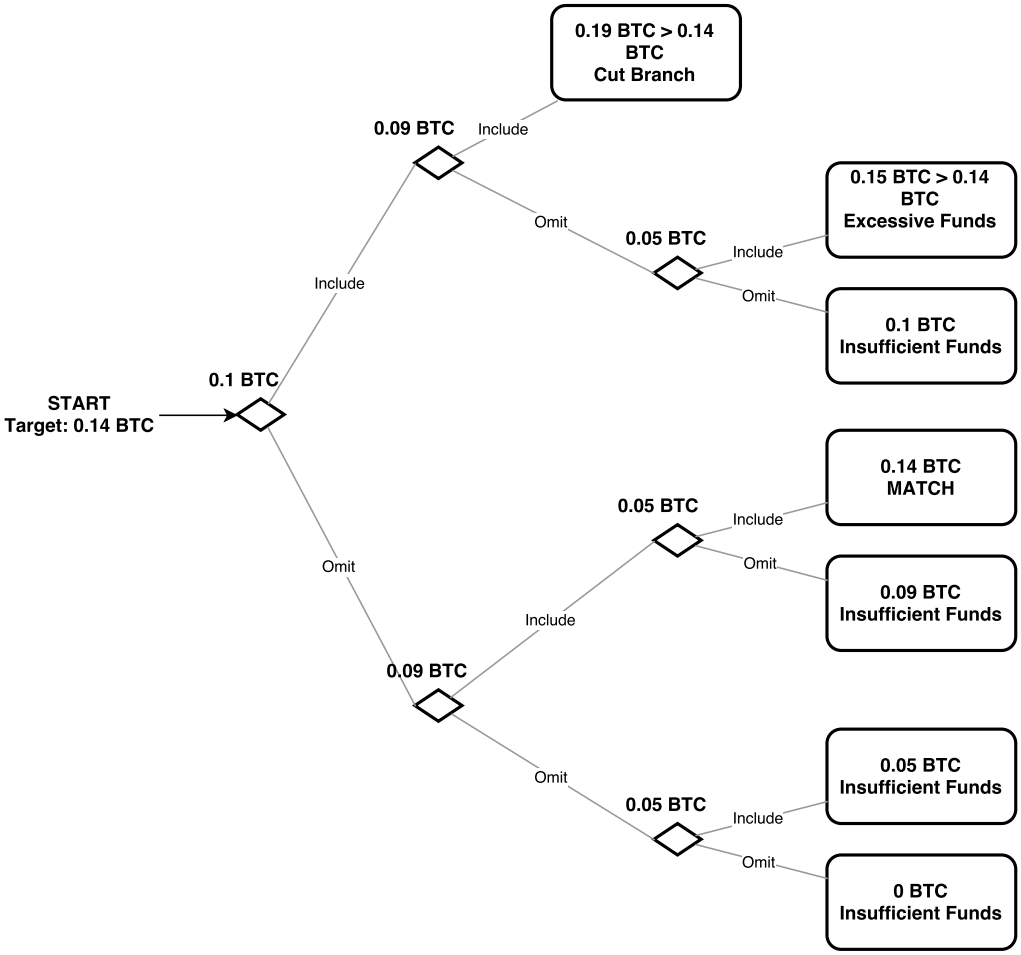

Figure 5.1: An example for the Branch and Bound search tree. The target is 0.14 BTC, and the UTXO pool consists of 0.1 BTC, 0.09 BTC, and 0.05 BTC. The top branch is cut when the combination of 0.1 BTC and 0.09 BTC exceeds the target, one leaf exceeds the target, one leaf produces a match, and the others don't reach sufficient funds.

The algorithm is presented in two steps: Algorithm 9 gives the overview, and algorithm 10 presents the recursive search.

The function is called with the spending target, the UTXO pool and a minimum change. For each payment request, the search limit `bnbTries` is set to 1 000 000 (see line 2). Every call of the recursive search decrements this by one, preventing that a very large UTXO pool that does not contain a exact match is searched exhaustively as the cost of doing so grows exponentially with the number UTXOs in the UTXO pool. The tree search is started at the root, depth 0, with an empty selection and thus zero value selected (see line 3). If no exact match is found, `branchAndBound(...)` returns the empty set, $\emptyset$. The minimum change is only needed for the fallback, the Single Random Draw that is performed if no exact match was found (see lines 4 – 10).

As mentioned above, algorithm 10 starts out by decrementing the remaining tries (line 4). Then, the current selection is tested:

1. *Exceeded target (lines 5, 6)*
   The effective value of the current selection is bigger than the target plus the

Algorithm 9: Branch and Bound: Select Coins

Input: t, the spending target
Input: U, the available UTXOs
Input: mc, the minimum change
Output: *selectedCoins*, the selected UTXOs to fund the transaction

```
1  selectedCoins ← ∅
2  bnbTries ← 1 000 000                          // class member limits search
   // start recursive dfs:  depth 0, no UTXOs selected, effValue 0, t
3  selectedCoins ← branchAndBound(0, ∅, 0, t)
   // If no match, Single Random Draw
4  if selectedCoins = ∅ then
5  |  shuffledPool ← randomize(U)
6  |  while sum(selectedCoins) < (t + estimateFee(selectedCoins) + mc) do
7  |  |  selectedCoins.add(shuffledPool.head)
8  |  |  shuffledPool ← shuffledPool.tail                          // drop head
9  |  end
10 end
11 return selectedCoins
```

allowed range for a match. As adding any further UTXOs would just further increase the sum, the branch can be cut. The empty set is returned to the parent call in the recursive structure.

2. *Match found (lines 7, 8)*
   The effective value is greater than the required funds for the transaction, but not too large as the exceed clause didn't trigger. Therefore, the current selection represents a match. The current selection is passed to the parent, and subsequently through all parent calls to the top, the search ends.

3. *Search limit reached (lines 9, 10)*
   The search exhausted the limit and is discontinued. The empty set is returned to the parent.

4. *Leaf reached, but no match (lines 11, 12)*
   The search has progressed to the end of a branch. As the above checks haven't triggered, the branch does not contain a match. The empty set is returned to the parent.

If no special cases are triggered, the search recursively expands the tree at random by including or omitting the UTXO at this depth (see algorithm 10 lines 14 – 25). In the case that the inclusion branch is being expanded, the selection is extended with the UTXO at the current depth, and the effective value of the selected set is increased accordingly. On the other hand, in the omission branch, the selection and its effective value remain the same. Then, `branchAndBound(...)` is called by incrementing the depth, with the thusly changed current selection. As the search is performed depth-first, the other branch will be explored eventually unless the first branch has found an exact match, or the search has reached the limit.

Algorithm 10: Branch and Bound: Search

Input: d, depth level in search
Output: *currentSelection*, UTXOs selected on this branch
Input: *effValue*, effective value selected
Input: t, target
Output: *selectedCoins*, the selected UTXOs to fund the transaction

$targetForMatch \leftarrow t + costOfHeader + costPerOutput$
$matchRange \leftarrow costPerInput + costPerOutput$
$utxoSorted \leftarrow sortDescending(U)$
$bnbTries \leftarrow bnbTries - 1$ `// count towards limit`
`// Check for solution and cut criteria`
if $effValue > targetForMatch + matchRange$ **then**
  return $\emptyset$ `// excessive funds: cut branch`
else if $effValue >= targetForMatch$ **then**
  return *currentSelection* `// match!`
else if $bnbTries <= 0$ **then**
  return $\emptyset$ `// search limit reached`
else if $depth >= utxoSorted.size$ **then**
  return $\emptyset$ `// leaf reached, no match`
else
  `// Randomly explore next branch`
  if *binaryRandom() = true* **then**
    `// Explore inclusion branch first, else omission branch`
    $withThis \leftarrow branchAndBound(d + 1, currentSelection + utxoSorted[d], effValue + utxoSorted[d].effValue, t)$
    if $withThis \neq \emptyset$ **then**
      return *withThis* `// match found`
    else
      $withoutThis \leftarrow branchAndBound(d + 1, currentSelection, effValue, t)$
      if $withoutThis \neq \emptyset$ **then**
        return *withoutThis* `// match found`
    end
  else
    `// As above but explore omission branch first`
    [...]
  end
end

6. Evaluation

6.1 Evaluation Criteria

As mentioned before, the simulation outputs a comma-separated-value table with statistics. A choice of the metrics collected from the simulations are presented in the following sections, and found again in the result tables.

6.1.1 UTXO footprint

As introduced in section 3.1.3, the UTXO set growth is an issue of concern. One goal of this work is to compare the impact of different coin selection policies on it. As wallet usage in real-world applications doesn't usually have a defined end point, every point in the simulation is of equal relevance. Hence, the simulation records the size of each wallet's UTXO pool after each incoming and outgoing payment. The *UTXO footprint* is then calculated as the arithmetic mean of these recorded UTXO pool sizes.

The metric *∅net-new UTXO* shows us the impact an average transaction produced by a policy has on the UTXO set.

Additionally, coin selection policies differ in the composition of the UTXO pools they maintain. To gain insight into the characteristics of the various UTXO pool compositions, a snapshot of the UTXO pool at the end of the scenario is stored.[1]

6.1.2 User Costs

The users' interest is to minimize their personal cost. The following quantities are collected:

1. *Mean Fee*
   The average fee per transaction is indicative of the short-term cost of a coin selection policy. It is labeled "∅fee" in the result tables, and measured in satoshis.

[1] It stands to reason that other times would be just as valid.

2. *Total Fees*
   This metric represents the aggregated fees of all transactions performed by the coin selection policy. It measures the total explicit cost for the user over the course of the scenario. As some coin selection policies opt to add change below the dust threshold to the fee, costs could differ between policies even for the same number of transactions with the same transaction sizes. It is measured in satoshis in the result tables.

3. *Empty Pool Cost*
   The above two quantities don't include the cost of spending the UTXOs that have been accumulated in the UTXO pool. This metric represents the required fees for spending all remaining UTXOs to empty the UTXO pool. It is measured in satoshis in the result tables.

4. *Total Costs*
   This metric compounds the total fees and the cost to empty the UTXO pool. It is indicative of the long-term cost to operate a coin selection policy. It is measured in satoshis in the result tables.

6.1.3 Input Set Sizes

Outlier input set sizes surprise users and cause discussion (see [Moor12]). It seems clear that slightly bigger fees in average would be an acceptable price for fewer and smaller outlier input sets. The simulation keeps track of the *biggest input set* as an indicator for the magnitude of these outliers, while *mean input set size* gives us a feeling for the common set size and an idea of the level of address linkage created by a coin selection policy. The *standard deviation of the input set size* tracks the stability of the selection. While in the Bitcoin network larger transactions might find it more difficult to get confirmed, the input set size in the simulation is not limited.

6.1.4 Change outputs

In transit gives the ratio of the change output to the UTXO pool's remaining total. The change output created by a transaction must wait for confirmation before it becomes reliably spendable again to the user[2]. Since the transaction's recipient can surmise that the second output on the transaction goes back to the sender, this metric also indicates a loss of privacy for the sender.

Count of matches is interesting in this context as transactions whose input set matches the sending amount don't create a change output. This metric can be called on to evaluate hypotheses such as whether Core's focus on exact matches pays off. Further, exact matches appear to be all around beneficial: circumventing a change output reduces address linkage, lowers transaction fees, and helps reduce the UTXO pool size.

Small change outputs have a high relative cost when spend. It may be more economical to create larger change outputs, or to forego creating them and add them

[2]Although, strictly speaking, the user could anticipate its return and already issue another transaction that spends it even before the first transaction is confirmed.

to the transaction fee instead. To that end, *smallest change output value*, *largest change output value*, *mean change output value*, and *standard deviation of change output value* are recorded.

6.1.5 Adaptability to Different Use Cases

It seems unlikely that one coin selection policy will fit all use cases. However, it is valuable if a policy works well in different usage scenarios. To that end, the simulation is run with several scenarios (see section 6.2) that differ in their ratios of incoming to outgoing transactions.

6.2 Scenarios

As described above, each scenario consists of an initial UTXO pool and a list of payments to be performed.

6.2.1 MoneyPot.com scenario

The most relevant scenario is a real-world data set of 36 256 payments provided by the web wallet service MoneyPot.com [Hava15a]. The data consists solely of the values of the payments and their order. The involved addresses or the times of the transactions were not made available.

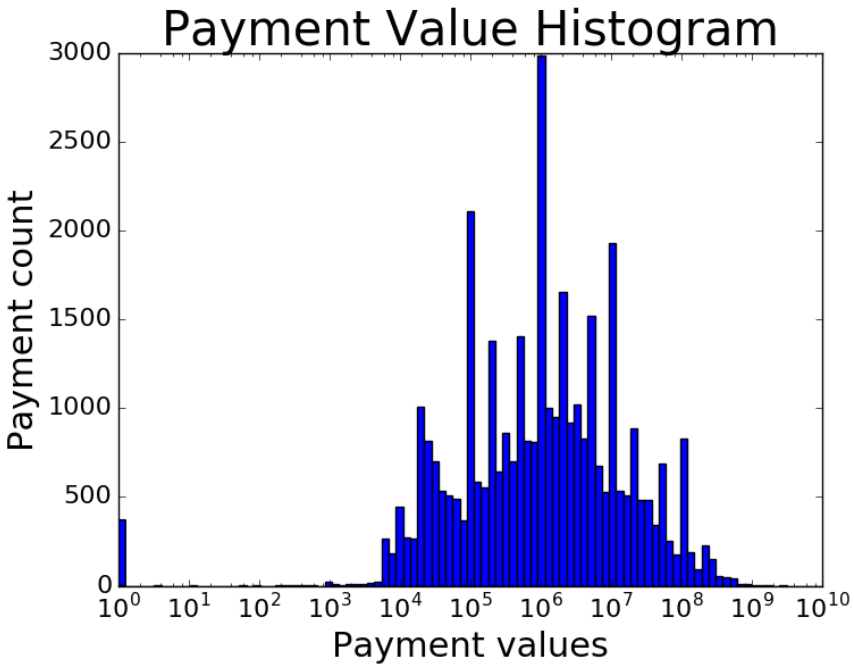

Figure 6.1: Histogram of payments in the MoneyPot.com scenario. Shown are the count of UTXOs of different value levels. The UTXOs' values are in satoshis and presented in logarithmic scale.

A distribution of the values of the payments in the scenario can be seen in Figure 6.1. The scenario consists of 24 388 incoming payments and 11 860 outgoing payments. While most incoming payments are at least 1 000 satoshi, there are 391 incoming

payments smaller than that, some of them below the dust limit. All outgoing payments are at least 0.1 mBTC (10 000 satoshi). The distribution of the payments has the semblance of a Gauss distribution centered around 0.01 BTC, overlayed with peaks at various round figures. The scenario is initialized with an empty UTXO pool and ends up with a final balance of more than 55 BTC.

6.2.2 Derived Scenarios from MoneyPot.com Data

The simulation results are extremely dependent on the scenario data. Since the MoneyPot.com dataset has about 2.06 incoming payments per outgoing payment, the scenario has a natural tendency to build up a larger UTXO footprint in wallets during the simulation. To see how the wallets would perform in a more balanced scenario, an additional scenario was derived from the MoneyPot.com data set by combining pairs of two subsequent incoming payments to a single incoming payment placed in the position of the second.

Example:
Starting out with the six payments $[1, -0.5, 0.75, 0.3, -0.4, 1.8]$*, where positive integers represent incoming payments,* 1 *and* 0.75 *would be combined to* 1.75 *and placed in the position of* 0.75*. Then the combination of* 0.3 *and* 1.8 *would be put at the previous position of* 1.8*. The result are the four payments* $[-0.5, 1.75, -0.4, 2.1]$.

The derived scenario results in 12 194 incoming payments and 11 860 outgoing payments, or about 1.03 incoming payments per outgoing payment. Only 152 incoming payments are now below 1 000 satoshi.

This approach is repeated to create a second derived data set from the first derived data set which now has fewer incoming than outgoing payments. As before, two subsequent incoming payments are combined to one, resulting in a set that has 6 097 incoming payments, and 11 860 outgoing payments. Looking at the original and the two derived scenarios, the results inform whether a coin selection policy adapts well to different use-cases.

6.3 Results

6.3.1 Policies in the Following Result Tables

The first four rows (see 6.1) show the policies inspired by existing wallets: FIFO (BreadWallet, Electrum), Pruned FIFO (Mycelium), Highest Priority (bitcoinj), and Bitcoin Core as described in section 3.3.

The next three rows show the coin selection policies with variations of Bitcoin Core:

- *Average Target* uses a minimum change that adjusts for each transaction to the average spending target of all previous transactions.
- *Wider Match Donation* allows for a broader range for exact matches than Bitcoin Core by recognizing differences up to the dust limit still as such. This difference is added to the fee in trade for circumventing the creation of a change output.

- *Double Target* uses a minimum change of the same size as the target instead of the fixed value Bitcoin Core does.

Other than the explicitly mentioned differences, these policies use exactly the same coin selection algorithm as Bitcoin Core does.

The third section cointains the two approaches that are independent of previous Bitcoin wallet implementations. *Single Random Draw (No MC)* shuffles the UTXO pool and selects from the front until it has reached the target. A change output is only created if it is greater than the dust limit, otherwise the remainder is added to the fee. *Single Random Draw (Cent)* follows the same policy for selection, but immediately aims for a change output of at least 0.01 BTC. The Branch and Bound coin selection policy is shown in two variants. *Branch'n'Bound (Cent)* uses a minimum change of 0.01 BTC, while *Branch'n'Bound (Millie)* uses a minimum change of 0.001 BTC. The Branch and Bound wallet purposefully searches for exact matches in a randomized depth-first-search and is followed by a single random selection in case no exact match is found in the first phase.

6.3.2 The Original MoneyPot.com Scenario

First discussed are the results from the original MoneyPot.com scenario (also see section 6.2.1). All of this section refers to table 6.1.

Policy	final value	∅#UTXO	final #UTXO	#received	#spent	#matches	#changes created
FIFO	5,528,692,230.60	182.82	246.10	36,232.50	35,986.40	15.50	11,844.50
Pruned FIFO	5,528,872,968.60	763.22	1,020.50	35,467.80	34,447.30	780.20	11,079.80
Priority	5,538,069,044.30	2,547.91	6,576.20	36,246.40	29,670.20	1.60	11,858.40
Bitcoin Core	5,509,924,210.00	169.27	43.40	36,180.90	36,137.50	67.10	11,792.90
AvTarget	5,514,137,750.80	128.49	101.20	36,179.60	36,078.40	68.40	11,791.60
WMDonation	5,505,726,611.70	496.69	568.70	35,581.50	35,012.80	666.50	11,193.50
Double Target	5,512,435,712.00	244.25	354.60	36,177.40	35,822.80	70.60	11,789.40
SRD (No MC)	5,528,698,806.20	187.15	255.10	36,221.90	35,966.80	26.10	11,833.90
SRD (Cent)	5,528,683,578.00	171.17	234.80	36,248.00	36,013.20	0.00	11,860.00
BnB (Cent)	5,532,264,911.60	105.89	79.70	32,634.00	32,554.30	3,614.00	8,246.00
BnB (Millie)	5,532,419,977.70	109.35	85.70	32,482.20	32,396.50	3,765.80	8,094.20

Policy	min change	max change	∅change	stDev of change	in transit	total fees	∅fee
FIFO	4,600.00	999,934,240.00	40,138,427.11	73,593,390.75	6.90%	62,541,683.40	5,273.33
Pruned FIFO	5,462.00	978,017,980.00	17,073,302.24	50,178,149.80	2.44%	62,360,945.40	5,258.09
Priority	5,700.00	999,898,130.00	78,582,144.03	111,142,824.94	8.80%	53,164,869.70	4,482.70
Bitcoin Core	1,000,000.00	470,448,612.00	3,303,305.37	16,002,328.07	2.38%	81,309,704.00	6,855.79
AvTarget	5,213.40	427,768,004.00	20,830,560.73	14,823,069.42	4.82%	77,096,163.20	6,500.52
WMDonation	1,000,000.00	431,444,150.00	2,949,862.98	14,428,964.02	2.51%	85,507,302.30	7,209.72
Double Target	15,880.00	1,451,614,259.60	19,659,464.34	58,898,494.36	2.43%	78,798,202.00	6,644.03
SRD (No MC)	4,599.00	997,199,161.70	38,115,471.25	74,026,698.98	5.44%	62,535,107.80	5,272.77
SRD (Cent)	1,000,456.10	993,181,226.00	42,966,112.93	77,784,158.70	6.22%	62,550,336.00	5,274.06
BnB (Cent)	1,002,159.30	999,942,808.10	49,578,507.24	99,048,240.15	5.69%	58,969,002.40	4,972.09
BnB (Millie)	101,070.20	998,262,312.40	46,519,186.78	96,452,298.37	5.92%	58,813,936.30	4,959.02

Policy	empty pool cost	total cost	max #inputs	∅Net-new UTXO	∅#inputs	stdev #inputs
FIFO	364,228.00	62,905,911.40	97.90	-1.0356	3.0343	5.3676
Pruned FIFO	1,510,340.00	63,871,285.40	131.40	-0.9703	2.9045	4.9482
Priority	9,732,776.00	62,897,645.70	1,389.50	-0.5018	2.5017	18.0969
Bitcoin Core	64,232.00	81,373,936.00	264.70	-1.0527	3.0470	4.1749
AvTarget	149,776.00	77,245,939.20	121.70	-1.0478	3.0420	2.8287
WMDonation	841,676.00	86,348,978.30	1,066.00	-1.0084	2.9522	10.7969
Double Target	524,808.00	79,323,010.00	319.70	-1.0264	3.0205	5.2659
SRD (No MC)	377,548.00	62,912,655.80	93.30	-1.0348	3.0326	4.7803
SRD (Cent)	347,504.00	62,897,840.00	85.10	-1.0365	3.0365	4.3621
BnB (Cent)	117,956.00	59,086,958.40	85.60	-1.0496	2.7449	3.1072
BnB (Millie)	126,836.00	58,940,772.30	89.50	-1.0491	2.7316	3.1625

Table 6.1: Simulation results for the MoneyPot.com scenario averaged over ten runs

6.3.2.1 UTXO footprint

The UTXO footprint is shown in column ∅`#UTXO`.

Bitcoin Core does well at maintaining a small UTXO footprint. Out of the existing wallets' policies (rows 1 – 4), it maintains clearly the smallest UTXO pool with 169.27 in average. FIFO does well considering the simplicity of the approach, following with 182.82. The Pruned FIFO approach collects a sizeable UTXO pool. Taking into regard the composition seen in the UTXO pool histogram (see figure 6.2), the obvious explanation seems to be that UTXOs of small value only exit the UTXO pool with some delay. While these UTXOs get to be the oldest UTXO remaining in the UTXO pool after awhile, and therefore sit at the front of the selection queue, they presumably get selected several times, yet also pruned from the candidate input set as they are smaller than the candidate inputs' change. As Pruned FIFO minimizes its change output size, it seems likely that the peak between 1 000 and 10 000 is created by its own change outputs which then later tend to not get spend. However, clearly ineffective at maintaining a small UTXO pool is the Highest Priority policy. Its UTXO footprint of 2 547.91 UTXOs in average is fifteen times as large as Bitcoin Core's. As the calculation of priority is much more influenced by the value of the UTXO than the age, it appears that Highest Priority mostly selects one or two of its larger UTXOs and grinds these down. This is also present in its UTXO pool histogram (see 6.2), which clearly shows hardly any large UTXOs remaining, but rather a peak between 10 000 and 100 000 satoshis. In comparison, Bitcoin Core's histogram shows few UTXOs below 10 000 000 satoshi (0.1 BTC), and FIFO shows a broad band of different values.

Average Target (128.49) undercuts the UTXO footprint significantly compared to Bitcoin Core, presumably because its minimum change policy adjusting to the average target leads to a much bigger average change size. The other two Core variants have larger UTXO footprints than Bitcoin Core, with Double Target doing 44% worse, and Wider Match Donation almost tripling Bitcoin Core's UTXO footprint.

However, Single Random Draw (Cent) puts Bitcoin Core's performance into perspective: Although it is one of the explicit goals of Bitcoin Core to maintain a small UTXO footprint, it just barely outperforms Single Random Draw (Cent)'s 171.17. Branch'n'Bound (Cent) manages to underbid all other policies with the best result of 105.89. In comparison to Branch'n'Bound (Millie), Branch'n'Bound (Cent)'s bigger minimum change causes a slightly larger average change output (∅`change`) and slightly increases its average input set size (∅`#inputs`). These two effects apparently are more influential on the UTXO footprint than Branch'n'Bound (Millie)'s higher exact match count (`#matches`).

6.3.2.2 User Costs

This section refers to the columns `total fees`, ∅`fee`, `empty pool cost`, and `total cost`.

Highest Priority clearly optimizes best for short-term costs. With an average fee (∅`fees`) of 4 482.70 satoshis per transaction, it is almost a tenth cheaper than the runner-up, Branch'n'Bound (Millie). Clearly most expensive are Bitcoin Core and its derivatives. The most effective Average Target saves about five percent compared to Bitcoin Core, Double Target saves about three percent. However, Bitcoin Core costs

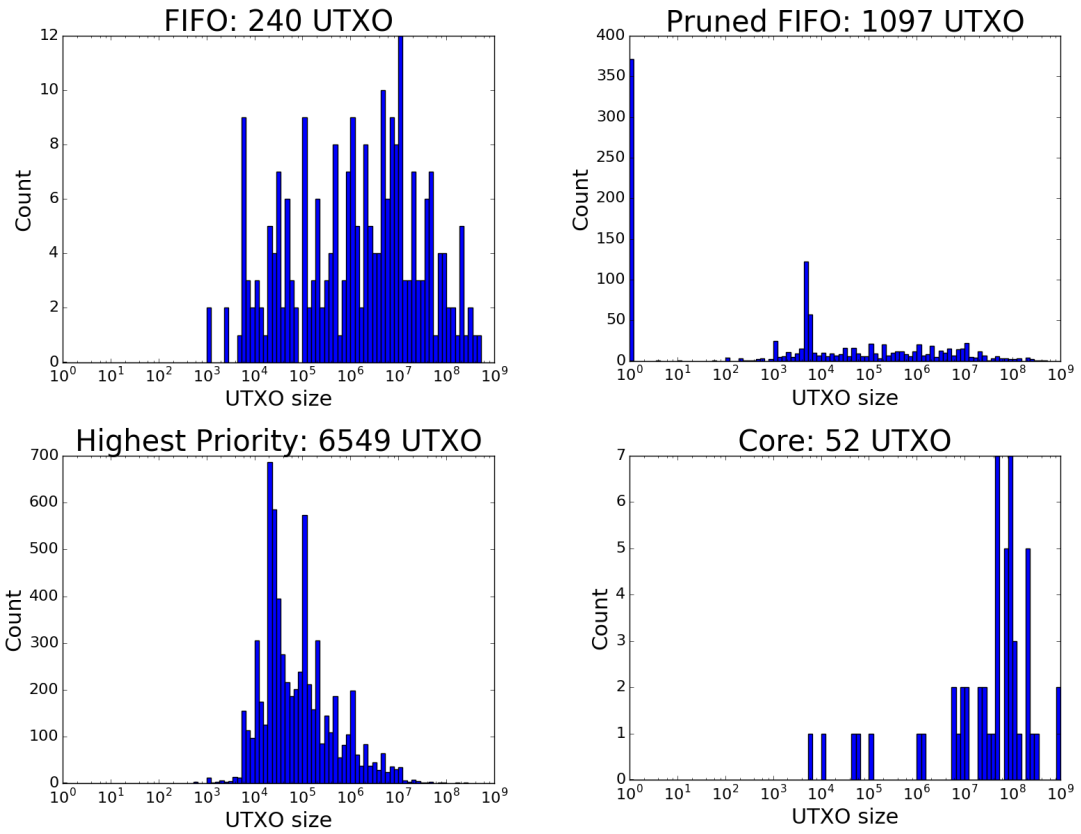

Figure 6.2: The composition of the UTXO footprints after the last transaction. Shown are the number of UTXO over their value measured in satoshi. The value is presented logarithmically.

about 53% more than Highest Priority per transaction. Wider Match Donation is the most costly overall with 7 209.72 satoshis per transaction. As the number of UTXOs received and spent (see `#received` and `#spent`) of the Core variants doesn't diverge significantly from the more economically performing FIFO policy, these higher must be attributed to Bitcoin Core's compounding fees (see section 3.4.2).

However, in the long-term perspective, Highest Priority loses ranks by having accumulated more than 6 500 UTXOs at this point still unpaid for. In total cost (`total cost`), Branch'n'Bound (Millie) turns out to be the most cost-effective, likely due to creating 31.75% exact matches. Not surprisingly, most of the remaining policies, exempting Core variants, are very close in the total cost. Solely, Pruned FIFO is slightly more expensive as it has also gathered a large UTXO pool.

6.3.2.3 Input Set Sizes

The most stable input set size (`stdev #inputs`) is created by the Average Target policy. Average Target uses Bitcoin Core's aggressive spending of small UTXOs, and creates larger change outputs than Bitcoin Core in this scenario. This appears to reduce the number of instances in which larger input sets are required to fund transactions, as well as reduce the size of the outlier input sets (`max #inputs`) from Core's 264.70 to 121.70.

Surprisingly, Wider Match Donations appears to produce rather unstable input set sizes. It seems likely that the sway given at finding exact matches causes it to find

single UTXO input sets at an increased rate, which then in turn causes Wider Match Donations to combine the smaller UTXOs in fewer, larger transactions than other Core variants.

As expected, Highest Priority causes unsteadily small and large input sets, culminating in a biggest input set (`max #inputs`) of 1 389.50 (in average over ten runs). This is easily explained by the Priority queue prefering old large transactions, resulting in change output with diminished value and reset age. Once the preeminent UTXOs have been trimmed, the coin selection policy combines numerous aged but small UTXOs.

Pruned FIFO is only slightly less stable than Single Random Draw (No MC), especially it manages a surprisingly small largest input set that is only half as large as Bitcoin Core's.

The smallest largest input sets are created by Single Random Draw (Cent) and Branch'n'Bound (Cent), which are 85.10 and 85.60 respectively. Both the Branch'n'Bound variants come close to Average Target's stability of the input sets.

6.3.2.4 Change outputs

The Core variants here provide the smallest in transit ratio (`in transit`). Bitcoin Core only sends an average of 2.38% of the wallet's total balance as change output. Pruned FIFO is almost on par with the Core variants. This is easy to explain, as all of these policies actively minimize the value of the change outputs. Highest Priority sends the largest portion of its funds on the round-trip with an in transit ration of 8.80%. The remaining policies lie between 5.4% – 6.9%.

Even though Bitcoin Core features several mechanisms to actively seek exact matches, it only finds 67.1 exact matches (`#matches`) in 11 860 outgoing payments (0.57%). The assumption that producing change outputs of the target's size would increase the chance of exact matches on basis of recurring payment levels can be rejected as Double Target merely finds 70.6 that is 3.5 more, exact matches than Bitcoin Core. Wider Match Donations improves on Bitcoin Core by producing 666.5, about ten times as many, exact matches which are about 5.62% of all outgoing payments. However, no Core variant can even outbid Pruned FIFO's 780.20 (6.58%), let alone Branch'n'Bound (Cent)'s 3 614 (30.47%) exact matches.

6.3.3 The First Derived MoneyPot.com Scenario: Balanced Payment Count

The second scenario simulated is the one where incoming and outgoing payments are almost in balance (see section 6.2.2). As the policies will behave qualitatively similarly, this section will focus on general differences and prominent observations. All of this section refers to the results shown in table 6.2.

6.3.3.1 UTXO footprint

The UTXO footprint is shown in column ∅`#UTXO`.

Now only contending with half as many incoming payments compared to the original dataset, all UTXO footprints have fallen significantly. The biggest improvement is

Policy	final value	∅#UTXO	final #UTXO	#received	#spent	#matches	#changes created
FIFO	5,546,571,853.00	100.78	134.00	24,041.00	23,907.00	13.00	11,847.00
Pruned FIFO	5,546,573,066.90	342.74	446.90	23,704.30	23,257.40	349.70	11,510.30
Priority	5,549,765,106.90	954.64	2,286.30	24,052.30	21,766.00	1.70	11,858.30
Bitcoin Core	5,543,405,686.00	38.33	25.50	24,052.50	24,027.00	1.50	11,858.50
AvTarget	5,542,475,686.60	58.52	23.10	24,046.30	24,023.20	7.70	11,852.30
WMDonation	5,542,307,167.40	41.94	25.20	23,865.80	23,840.60	188.20	11,671.80
Double Target	5,538,434,396.00	67.36	59.80	24,051.10	23,991.30	2.90	11,857.10
SRD (No MC)	5,546,599,573.30	105.93	153.20	24,043.00	23,889.80	11.00	11,849.00
SRD (Cent)	5,546,595,278.00	100.49	143.30	24,054.00	23,910.70	0.00	11,860.00
BnB (Cent)	5,548,171,367.90	52.21	62.70	22,456.70	22,394.00	1,597.30	10,262.70
BnB (Millie)	5,548,229,602.10	52.55	64.70	22,406.90	22,342.20	1,647.10	10,212.90

Policy	min change	max change	∅change	stDev of change	in transit	total fees	∅fee
FIFO	4,932.00	997,593,773.00	51,207,660.18	89,088,701.09	8.21%	44,662,061.00	3,765.77
Pruned FIFO	5,464.50	948,761,860.00	21,663,760.66	57,527,027.00	3.36%	44,660,847.10	3,765.67
Priority	5,313.30	1,002,586,670.00	86,947,680.32	117,476,037.42	9.87%	41,468,807.10	3,496.53
Bitcoin Core	1,000,000.00	508,004,033.10	5,338,001.01	18,897,024.16	2.66%	47,828,228.00	4,032.73
AvTarget	5,065.80	446,456,967.30	22,431,585.98	16,849,997.94	5.08%	48,758,227.40	4,111.15
WMDonation	1,000,003.60	456,160,296.00	5,406,154.49	19,217,596.25	2.77%	48,926,746.60	4,125.36
Double Target	20,505.40	1,500,010,329.80	21,509,103.00	60,499,867.75	3.34%	52,799,518.00	4,451.90
SRD (No MC)	5,019.40	998,852,243.40	46,869,396.07	82,972,675.07	6.65%	44,634,340.70	3,763.44
SRD (Cent)	1,001,329.30	997,814,772.20	50,499,698.15	84,586,244.33	7.26%	44,638,636.00	3,763.80
BnB (Cent)	1,001,627.90	1,002,329,675.20	73,186,172.15	114,914,054.00	7.50%	43,062,546.10	3,630.91
BnB (Millie)	101,578.50	995,046,970.00	71,528,206.49	114,515,783.57	7.34%	43,004,311.90	3,626.00

Policy	empty pool cost	total cost	max #inputs	∅net-new UTXO	∅#inputs	stdev #inputs
FIFO	198,320.00	44,860,381.00	63.00	-0.02	2.02	2.94
Pruned FIFO	661,412.00	45,322,259.10	75.40	0.01	1.96	2.84
Priority	3,383,724.00	44,852,531.10	521.60	0.16	1.84	7.35
Bitcoin Core	37,740.00	47,865,968.00	45.50	-0.03	2.03	1.63
AvTarget	34,188.00	48,792,415.40	71.50	-0.03	2.03	1.70
WMDonation	37,296.00	48,964,042.60	58.90	-0.03	2.01	1.66
Double Target	88,504.00	52,888,022.00	70.20	-0.02	2.02	1.96
SRD (No MC)	226,736.00	44,861,076.70	52.90	-0.02	2.01	2.69
SRD (Cent)	212,084.00	44,850,720.00	52.00	-0.02	2.02	2.54
BnB (Cent)	92,796.00	43,155,342.10	39.60	-0.02	1.89	2.01
BnB (Millie)	95,756.00	43,100,067.90	40.60	-0.02	1.88	2.05

Table 6.2: Simulation results for the first derived scenario "balanced inputs and outputs" averaged over ten runs

shown by Wider Match Donation which reduces its UTXO footprints by 91.56% from 496.69 to 41.94. Bitcoin Core also improves drastically with a reduction of 77.35% from 169.27 to 38.33, to the now smallest UTXO footprint in absolute numbers. The two Single Random Draw variants and FIFO only improve by 41.30% – 44.87%.

Even though Pruned FIFO and Priority had large absolute reductions, they still maintain an average of 342.74 and 954.64 UTXOs respectively in their UTXO pool over the course of the scenario. These policies are the only two that remain at higher UTXO footprints than the previous lowest levels.

6.3.3.2 User Costs

In the original scenario the ∅`fee` was spread between 4 482.70 (Highest Priority) and 7 209.72 (Wider Match Donation), a range of more than 60% increase over the lowest. In the balanced scenario, this range has gotten much smaller. It is now reduced to 3 496.53 (Highest Priority) and 4 451.90 (Double Target) which is only an increase of about 27%.

It is notable that now Bitcoin Core itself has the lowest average fee and lowest total cost among the Core variants, whereas before both Average Target and Double Target were more efficient. The fee estimation of Bitcoin Core and its variants appears to work much better in the balanced scenario, presumably because their

much smaller UTXO pools now contain hardly any small UTXOs, and therefore the compounding fee issue (see section 3.4.2) is mitigated.

In terms of total cost, the best policy is again Branch and Bound (Millie) with only 431.00 mBTC. Disregarding the variants, Bitcoin Core is the most expensive with 478.66 mBTC.

6.3.3.3 Input Set Sizes

Input set sizes in general become much more stable. The range in `stdev #inputs` is reduced to the range of 1.63 – 7.35 where it was 2.82 – 18.10 in the original scenario.

The outlier input sets drop below the smallest count of the original scenario for all policies except Highest Priority which still has a `max #inputs` of 521.60.

6.3.3.4 Change output

Average change output value (and therefore also the in transit ratio) rise slightly for all policies.

Meanwhile, the number of exact matches falls significantly for the policies that produced some before. Bitcoin Core now only finds 1.50 exact matches in 11 860 outgoing payments (down from 67.10). Average Target and Double Target both fall below 10 exact matches. Wider Match Donation drops from 666.50 to 188.20, whereas Pruned FIFO is less affected with 780.20 to 349.70. Branch and Bound (Millie) retains the highest count with 1 647.10 exact matches (13.89%), compared to also the highest count in the original scenario of 3 765.80.

6.3.3.5 Summary

In the derived balanced scenario, the three Core variants become almost entirely obsolete: where before each had some advantages over Bitcoin Core, now Core performs better in almost all criteria. The only exception is exact matches where the variants find more. However, the increased matches do not facilitate reduced cost or smaller UTXO footprint in this case. The sole benefit is the reduced address linkage, which seems a feeble argument even for the 188.20 exact matches for Wider Match Donation in light of this only affecting 1.59% of its outgoing payments.

6.3.4 The Second Derived MoneyPot.com Scenario: One Incoming per Two Outgoing Payments

In this second derived scenario, there are fewer incoming than outgoing payments (see section 6.2.2). As the general trends of the policies have already been covered above, this section will focus on discrepancies to previous observations. All of this section refers to the results shown in table 6.3.

6.3.4.1 UTXO footprint

∅`#UTXO` are generally low in this scenario as a small UTXO pool is easily achieved at 0.51 incoming payments per outgoing payment. Still, Pruned FIFO and Highest Priority have UTXO footprints of 167.76 and 342.23. In the original scenario with four times the number of incoming transactions, Bitcoin Core had a UTXO footprint of 169.27.

Policy	final value	∅#UTXO	final #UTXO	#received	#spent	#matches	#changes created
FIFO	5,555,515,404.00	59.73	76.00	17,955.00	17,879.00	2.00	11,858.00
Pruned FIFO	5,555,468,801.80	167.76	229.70	17,748.90	17,519.20	208.10	11,651.90
Priority	5,556,491,382.70	342.23	733.00	17,956.60	17,223.60	0.40	11,859.60
Bitcoin Core	5,554,832,080.00	28.02	17.70	17,957.00	17,939.30	0.00	11,860.00
AvTarget	5,554,676,500.00	21.90	18.00	17,956.40	17,938.40	0.60	11,859.40
WMDonation	5,554,768,666.40	24.11	17.70	17,918.30	17,900.60	38.70	11,821.30
Double Target	5,554,224,882.00	30.29	25.30	17,956.50	17,931.20	0.50	11,859.50
SRD (No MC)	5,555,529,559.20	61.47	85.30	17,954.20	17,868.90	2.80	11,857.20
SRD (Cent)	5,555,534,330.00	59.95	86.20	17,957.00	17,870.80	0.00	11,860.00
BnB (Cent)	5,556,184,554.00	37.21	54.40	17,298.60	17,244.20	658.40	11,201.60
BnB (Millie)	5,556,226,150.60	37.87	54.40	17,265.80	17,211.40	691.20	11,168.80

Policy	min change	max change	∅change	stDev of change	in transit	total fees	∅ fee
FIFO	6,806.00	1,029,277,740.00	67,664,151.84	100,007,776.39	9.79%	35,718,510.00	3,011.68
Pruned FIFO	5,503.00	975,265,242.70	31,277,087.26	67,668,614.47	4.10%	35,765,112.20	3,015.61
Priority	11,962.90	1,026,855,450.00	100,971,999.46	122,663,339.65	11.13%	34,742,531.30	2,929.39
Bitcoin Core	1,000,015.20	663,312,279.70	9,821,088.45	26,176,028.77	3.41%	36,401,834.00	3,069.29
AvTarget	6,809.50	450,885,900.00	25,965,888.56	22,159,313.43	5.80%	36,557,414.00	3,082.41
WMDonation	1,000,040.50	626,448,716.90	9,950,737.87	26,619,087.92	3.45%	36,465,247.60	3,074.64
Double Target	25,922.80	1,500,013,869.20	25,404,829.07	61,478,463.81	3.63%	37,009,032.00	3,120.49
SRD (No MC)	6,544.00	1,008,085,151.70	63,741,512.13	97,639,841.33	8.05%	35,704,354.80	3,010.49
SRD (Cent)	1,001,302.20	1,012,001,429.20	65,630,573.98	96,296,281.66	8.51%	35,699,584.00	3,010.08
BnB (Cent)	1,002,997.40	1,025,073,003.00	90,628,546.16	120,658,408.35	9.21%	35,049,360.00	2,955.26
BnB (Millie)	103,499.10	1,018,509,791.20	88,643,507.89	118,140,522.22	8.75%	35,007,763.40	2,951.75

Policy	empty pool cost	total cost	max #inputs	∅Net-new UTXO	∅#inputs	stdev #inputs
FIFO	112,480.00	35,830,990.00	33.00	0.49	1.51	1.63
Pruned FIFO	339,956.00	36,105,068.20	39.00	0.51	1.48	1.56
Priority	1,084,840.00	35,827,371.30	206.30	0.55	1.45	3.25
Bitcoin Core	26,196.00	36,428,030.00	26.90	0.49	1.51	1.06
AvTarget	26,640.00	36,584,054.00	19.20	0.49	1.51	0.95
WMDonation	26,196.00	36,491,443.60	26.10	0.49	1.51	1.06
Double Target	37,444.00	37,046,476.00	27.60	0.49	1.51	1.16
SRD (No MC)	126,244.00	35,830,598.80	30.70	0.49	1.51	1.54
SRD (Cent)	127,576.00	35,827,160.00	30.90	0.49	1.51	1.49
BnB (Cent)	80,512.00	35,129,872.00	23.40	0.49	1.45	1.37
BnB (Millie)	80,512.00	35,088,275.40	21.70	0.49	1.45	1.38

Table 6.3: Simulation results for the second derived scenario averaged over ten runs

6.3.4.2 User Costs

The variance in `total cost` has come down to a range of less than 6%. The Core variants still are the most expensive, although Bitcoin Core itself least so.

Pruned FIFO's total cost is only below the Core variants, and higher than Single Random Draw or FIFO. This may be explained by the pruning of the candidate input set. The pruning minimizes the size of the change output which is added to the fee below the fixed limit of 5 460 satoshis. This range is broader than for example in Branch'n'Bound which delimits exact matches to a range of 1 820 satoshis in our scenario.

As in the other two scenarios, Branch'n'Bound (Millie) has the lowest total cost.

6.3.4.3 Input Set Sizes

Even in this benign scenario, Highest Priority creates outlier input sets with `max #inputs` of 206.30.

6.3.4.4 Change Outputs

`In transit` increases further from the other two scenarios. Highest Priority now adds an average of 11.13% of the wallet's total balance as a change output to its transactions. Bitcoin Core has the lowest figure at 3.41%.

Pruned FIFO still produces a `#matches` of 208.10. As explained above this appears to be detrimental here, though. Branch'n'Bound (Millie) finds 691.20 exact matches (5.83% of 11 860 outgoing payments) but still has a lower ∅`fee`.

6.4 Discussion

6.4.1 FIFO

The simple approach of spending the oldest UTXOs first produces remarkably stable results. It adapts well to the different use cases and isn't prone to outlier input sets, even though the input set size has a higher variance than other policies. As the policy doesn't shape its change size, the in transit ratio is in the lower upper range.

6.4.2 Pruned FIFO

The Pruned FIFO policy clearly leads to a bloated UTXO pool as its UTXO footprint is large even for the benign scenario. Additionally, its pruning appears to sometimes cause increased fees as the limit to drop change outputs to the fee is higher than in other policies. The outlier input sets were less pronounced than expected by the author. Rather, the deviation and size of the input sets remain below the numbers of FIFO. As Pruned FIFO minimizes its change output, it has a low in transit ratio.

6.4.3 Highest Priority

Priority based selection has only one redeeming quality, its optimization of short term costs. However, in light of the fees being likely to rise in the longterm, putting off spending of UTXOs actively harms the user interests. Besides this, it causes an enormous UTXO pool in all examined scenarios, has the largest in transit ratio and only a fraction of the lead in short-term costs carries over to the total cost. Highest Priority has by far the largest outlier input sets which likely causes users to scratch their head at times even at much fewer transactions than performed in these scenarios.

6.4.4 Bitcoin Core

In the first scenario, Bitcoin Core only barely stays below the UTXO footprint of Single Random Draw (Cent), but it does much better in the balanced and outgoing payment heavy scenarios. The main point of criticism is the increased cost resulting from its fee estimation design. Especially with a large UTXO pool, user costs lie far higher than for other prevalent wallet policies. Compared to Highest Priority, Bitcoin Core pays 29.4% more fees in the original scenario. In the other scenarios this gap shrinks to 6.7% and 1.7% respectively. As Bitcoin Core is probably popular among online services such as the source of the original scenario, this may be cause significant real-life costs to those users.

In general, Bitcoin Core's selection produces stable input set sizes as the `stdev #inputs` is among the lower, but especially in the original scenario big outlier input sets can occur.

Bitcoin Core's exact match centric design does not pay off. Even in the original scenario where it maintains a pool of 169.27 UTXOs in average, it only produces an

average of 67.10 matches (1.43%). In the other two scenarios this drops to 1.50 and 0.00.

The computational cost of Bitcoin Core's policy is a multiple greater than any of the non-related policies. While the original scenario can be run with e.g. the Pruned FIFO scenario in less than a minute, Bitcoin Core clocks in at more than an hour.

6.4.5 Core Variants

The Core variants suffer most of the same drawbacks as Bitcoin Core itself. They improve on it in some minor ways, but none of them improves performance across the board. Average Target has lower UTXO footprints in the original and second derived scenario, but a higher in the balanced scenario. Compared to Bitcoin Core, it stabilizes selection in the original scenario, but not in the first derived scenario. Wider Match Donation produces more exact matches, but has a higher total cost in all three scenarios. Double Target doesn't seem to do anything better than Bitcoin Core, so except for its privacy benefit change outputs of the same size of the spending target don't appear beneficial.

6.4.6 Single Random Draw

While it first was intended as a naive baseline, Single Random Draw actually performs well in most aspects. As expected, its values are close to FIFO most of the time which is similar as it also doesn't perform any additional sorting on the UTXO pool before coin selection. Single Random Draw (Cent) produces a similar UTXO footprint as Bitcoin Core in the original scenario, and remains significantly below Pruned FIFO and Highest Priority for the other two scenarios.

The variant with a minimum change of 0.01 BTC has across the board a slightly lower UTXO footprint and lower `total cost`. It outperforms all core variants in terms of costs, but doesn't contend for the cheapest. By the nature of the selection, #matches are basically non-existent.

6.4.7 Branch'n'Bound

Branch'n'Bound performs very well in almost all aspects. While Branch'n'Bound (Cent) has a slightly smaller UTXO footprint than Branch'n'Bound (Millie), both are much smaller than any other policies' for the original scenario. Altogether, they have the smallest range in their UTXO footprint, and they remain the smallest except for Bitcoin Core and its variants in the two derived scenarios. It might actually be advantageous not to shrink the UTXO pool too far, both to increase the chance of exact matches occurring and to increase the size of the set on which the links between addresses is spread.

Branch'n'Bound (Millie) has the lowest `total cost` across all three scenarios, underbidding even Highest Priority by 6.7% in the original, 4.1% in the balanced, and 2.1% in the second derived scenario.

Both Branch'n'Bound variants come close to Highest Priority for the smallest average input set size, yet without building up an enormous UTXO pool.

Only for the `in transit` ratio, Branch'n'Bound is between Single Random Draw and FIFO, and therefore significantly higher than the Core variants.

Its computational effort is significantly greater than most other coin selection approaches, when no exact match is found, yet the expenditure remains capped other than for Bitcoin Core. The expense seems well worth in light of the benefits arising from it.

6.4.8 Privacy

As a full exploration of the privacy benefits of the examined coin selection policies lacks support in the model without addresses, just a few thoughts at this point: Bitcoin Core practically announces itself with its transactions by being the only wallet that makes use of the `nLockTime` flag on every transaction. This in combination with the fixed minimum change of 0.01 BTC reveals the identity of the spending output in almost every case. Deterministically selecting algorithms as Highest Priority and FIFO exhibit patterns that offer insights to scrutiny, at least the type of wallet, and perhaps information about the oldest or largest UTXOs in the wallet's UTXO pool. Ironically, address linkage might be worst in the approach with the smallest average input set, Highest Priority, as it tends to have the largest outliers for the input set size. Users that are willing to invest into their privacy would probably do best to use Bitcoin Core's CoinControl feature and to manually keep track of the different groups of their addresses that they want to keep separate.

Algorithmically, the Branch'n'Bound policy would be expected to exhibit good privacy properties: In the case of an exact match, there is no change output to link to other addresses of the wallet in the first place. As the fallback in the case of no match is equivalent to Single Random Draw, it seems fair to surmise that the fallback will stably produce small sets (Single Random Draw (Cent) has a standard deviation of 4.36 for the input set size), and as the algorithm is randomly selecting on a set unknown to the observer, there should be little information leakage.

6.5 Summary

This chapter presented evaluation criteria to examine the impact on the UTXO set, the user costs, and other selection behavior of policies. Three scenarios were introduced and simulation results were presented for each. Nine different coin selection approaches were examined and discussed.

It was shown that Priority-based coin selection and Pruned FIFO create a large UTXO footprint in different usage scenarios. These policies appear to negatively impact on the UTXO set without balancing user benefits. The author feels that their use should be discouraged in the future.

Bitcoin Core's match centric design doesn't pay off. In none of the three scenarios a significant amount of exact matches are produced. Among the established approaches, Bitcoin Core appears to be the most costly for its users, and maintains the smallest UTXO footprint. When it is operating with a large UTXO pool, computational expenses are significant, and it can produce large outlier input sets. Bitcoin Core performs significantly better on the derived scenarios where there are not significantly more incoming payments than outgoing payments.

The proposed ideas to improve Bitcoin Core's coin selection did not have the expected impact. The different variants all improve Bitcoin Core's performance only in some aspects, but none is obviously better.

FIFO and Single Random Draw do surprisingly well for coin selection. Single Random Draw (Cent) is a bit cheaper and should exhibit better privacy than FIFO, so it is recommended as a simple coin selection policy for newly created wallets. In fact, it appears that it would mean an improvement for many established wallets used in the Bitcoin community.

Branch'n'Bound clearly improves on the established coin selection policies across the board: It produces a very low UTXO footprint in the original scenario, and comes close Bitcoin Core in the other two. It reduces the cost of the previously cheapest policy of Highest Priority and produces small largest input sets. Branch'n'Bound produces more than fifty times as many transactions without change output as Bitcoin Core. The only shortcoming is a slightly higher average change output size when no exact match is found which might make it less useful for high volume services. This algorithm is recommended as a basis for a future standard coin selection approach.

7. Conclusion and Future Work

This work provides a comprehensive analysis of the coin selection problem. While the design space is not restrictive, there are many requirements some of which are contradictory. The main conundrum is to achieve low user costs, low network resource usage, and good privacy all at once.

This thesis describes several coin selection policies currently in use on the network. Among them, the coin selection algorithm of the reference implementation is elaborated in detail. A number of issues with the discussed coin selection algorithms are identified. Further, some thoughts of the Bitcoin community about coin selection are collected, and a range of approaches to improve coin selection in regard to aforementioned goals is explored. The change output size is identified as a tool for the wallet to influence its own UTXO pool composition, and exact matches are shown to be beneficial in regard to all of the three main requirements.

A framework is provided to simulate the long-term development of a wallet's UTXO pool composition under different coin selection strategies. The framework restricts itself to a model of an isolated wallet that only receives and sends payments. The payment process in the model closely adheres to the actual process in regular wallet software. The model manages its own UTXOs, and performs coin selection including fee estimation and observing blockheight. The simulator sends incoming and outgoing payment orders to run the wallet model through scenarios. The simulation creates an output file with statistical data relevant for the evalution.

The simulation framework has been made available to the public, to allow in house testing of different coin selection policies for the purposes of organizations unwilling to share their data set with the public.

Multiple coin selection policies have been reimplemented, and several of the proposed improvement ideas have been implemented. With the Branch'n'Bound algorithm, a new coin selection policy has been designed and implemented. Branch'n'Bound expediently searches for exact matches, and then uses a random selection as fallback when none are discovered.

All nine described coin selection policies have been tested in regards to a scenario that might be indicative for a merchant and service provider use case. Further, the

algorithms have been simulated on two derived scenarios which shift the ratio of incoming and outgoing payments. Where the original scenario, a real-world data set provided by an online wallet service, has a ratio of about two incoming payments per outgoing payment, the derived scenarios are almost balanced, and inverted to one incoming per two outgoing payments.

It was discovered that two policies inspired by prevalent wallet software on the Bitcoin network create large UTXO footprints even in benign scenarios with fewer incoming payments than outgoing payments. Bitcoin Core and the examined derived policies were shown to overestimate fees by up to 30% in the original scenario, i.e. in an incoming payment heavy use case. The exact match design of Bitcoin Core's coin selection doesn't pay off, as matches occur only on less than 1.5% of the outgoing payments. Of the established coin selection policies, Bitcoin Core's produced the smallest UTXO footprint. The derived variants of Bitcoin Core show only localized minor improvements, but none is a clear improvement in comparison to Bitcoin Core.

Selecting randomly from the UTXO pool as a coin selection strategy has shown to produce reliable but not excellent results. With some added fallbacks to mitigate pathological cases, random selection could represent an improvement for some existing wallets, or serve as a simple coin selection solution for new wallets aimed at casual users.

The introduced Branch'n'Bound algorithm performs very well in almost all aspects. It has the lowest range of UTXO footprints across the three scenarios, even underbidding Bitcoin Core's UTXO footprint in the original scenario by more than 37%. Depending on the scenario, the algorithm discovers exact matches for up to 30% of the outgoing payments. In user cost, it outperforms the previously most economic, priority-based selection by 2.1% – 6.7%. While it is more computationally intensive than most policies, it is still much faster than Bitcoin Core in the simulation. The only drawback is it's average change output size, which is significantly higher than Bitcoin Core's, yet not the highest.

In the future, the framework should be extended with addresses to further explore the privacy impact of coin selection policies. In that regard, also support for multiple change outputs could be added. A model could be integrated to extend the simulation framework with changing fees. As the evaluation is chiefly built around one real-world data set and the simulation results are highly dependent on the scenario, experiments with additional data sets should be arranged to check the representativeness of this work's results. The Branch'n'Bound algorithm could perhaps be improved by adding an additional cut-off for branches that cannot produce a valid result anymore. For this "lookahead" it should be simple to calculate the maximum value that could still be added after a given level of the tree, making the DFS much more efficient by not exploring fruitless branches to the leaf.

Glossary

Base58Check Extends Base58 with a four byte checksum appended to the end in order to make mistakes when manually importing very unlikely to go unnoticed. 8, 59

address A hash representation of a user's public key. Addresses have 33 or 34 symbols when presented in Base58Check, and start with a 1 for P2PKH and a 3 for P2SH. 5, 59

Base58 An encoding for large integers on basis of digits, lower case letters and capital letters. The symbols 0, O, I, and l are excluded to avoid mix ups. 8, 59

Bitcoin Core The reference implementation of Bitcoin, currently, the *defacto* protocol definition. Written in C++. Formerly known as Bitcoin-Qt. Maintained by the Bitcoin Core Developers: http://github.com/bitcoin/bitcoin. vii–ix, 1, 3, 6–9, 12, 14, 17–20, 22, 24–26, 28, 31, 33–36, 44–55, 58, 59

change output A transaction output that returns to the sender what's left after paying for spending target and fee. viii, ix, 6, 10, 12–18, 22, 24–36, 42, 44–48, 50–55, 57–59

doublespend The act of authoring two transactions which both reference the same UTXO as input. Since UTXOs can only be spend once, the transactions are in competition and only one of them can be confirmed. 3, 5, 15, 59

dust A UTXO whose spending would cost more than a third of its own value in fees. 6, 7, 10, 14, 16, 34, 35, 44, 45, 59

ECDSA Elliptic Curve Digital Signature Algorithm. 7, 59

exact match A set of inputs that will avoid the creation of a change output. vii, viii, 17, 18, 20, 22–25, 28, 33–38, 42, 44–48, 50–55, 57–59

FIFO A queuing scheme where the oldest entries are processed first. 16, 35, 47, 59

full node A "full node" refers to Bitcoin clients that validate the complete transaction history of Bitcoin and enforce all rules in Bitcoin. Most full nodes store a complete copy of the Bitcoin blockchain and provide thin clients with lookup services. 8, 11, 14, 59

height The blockchain height refers to the index of a block in the longest valid chain derived from the Genesis Block. The Genesis Block hereby is at height zero, while the fifth block after the Genesis Block is at height five. 7, 30, 31, 59

input A script that references a previous UTXO to be used to fund a transaction. 5, 59

memory pool A node's storage of unconfirmed transactions. Memory pools between different nodes may differ in content. A limit for the memory pool's size can be set with a mininum fee ration (fee per byte) to admit transactions, or a maximum size of memory set aside for it. When the memory pool is full it will evict the transactions with the smallest fee ratio to admit better endowed transactions. 5, 59

miner Depending on the context, either a person, an entity or a device that contributes computing power to secure the Bitcoin network. 3–5, 59

mining The competitive process to determine the author of the next block. Results in synchronization of the network's state and securing of the network's transactional history. 4, 59

Opt-in Replace-by-Fee A transaction flag that denotes transactions as replaceable until they are confirmed in a block. 15, 59

output A script that designates an amount of bitcoins to be spendable by the owner of the recipient address, or an alternative authentication token. 5, 59

P2PK Pay-to-PubKey. 8, 59

P2PKH Pay-to-PubKey-Hash. 5, 7, 8, 11, 12, 29, 59

P2SH Pay-to-Script-Hash. 29, 59

priority Transaction metric derived from the product of age and value of the inputs. Age is measured in height, the value measured in satoshi. 12, 16, 59

satoshi A satoshi is the smallest subunit of a bitcoin. Bitcoins are divisible into $10e8$ satoshi, i.e. one satoshi is a hundred millionth of a bitcoin. 10, 31, 41, 42, 59

Segregated Witness An Bitcoin Improvement Proposal to resolve transaction malleability and rectify inefficiencies, see BIP 141[LoLW16]. 15, 59

transaction An order to the network to transfer bitcoins from UTXO referenced in inputs to newly created UTXO referenced in a list of outputs. vii, 3, 4, 7, 11, 12, 59

TXID The unique identifier for a transaction created by hashing the transaction. 5, 15, 59

UTXO An output from a transaction that has not been spent yet. UTXOs are atomic in that they can only spent in complete or not at all. v, viii, 1, 2, 5–7, 9–16, 18, 20, 22–38, 41–43, 46–50, 52, 54, 57, 59

UTXO footprint The average number of UTXOs held by a wallet throughout the scenario. viii, ix, 13, 24, 26, 35, 41, 44, 46–50, 52–55, 58, 59

UTXO pool The UTXOs belonging to one wallet. 2, 9, 12–14, 16–18, 20, 23, 26–33, 35–37, 41–47, 49, 50, 52–54, 57–59

UTXO set The global set of all UTXOs in the network. v, 2, 7, 9–12, 14, 15, 27, 41, 54, 59

wallet Software that manages the private keys of a Bitcoin user. Wallets track a user's transactions, UTXOs, and allow the user to create new payments. 7, 59

Bibliography

[Anto16] A. Antonopoulos. Segregated Witness and aligning economic incentives with resource costs. https://decentralize.today/segregated-witness-and-aligning-economic-incentives-with-resource-costs-7d987b135c00, 2016. [Online, retrieved on 2016-10-18].

[Bitc16a] Bitcoin Core developers. Bitcoin Core version 0.12.0 released. https://bitcoin.org/en/release/v0.12.0, 2016. [Online, retrieved on 2016-10-20].

[Bitc16b] Bitcoin Core developers. Revisit Coin Selection. https://github.com/bitcoin/bitcoin/issues/7664, 2016. [Online, retrieved on 2016-06-30].

[Bitc16c] Bitcoin Core developers. Segregated Witness Benefits - Reducing UTXO growth. https://bitcoincore.org/en/2016/01/26/segwit-benefits/#reducing-utxo-growth, 2016. [Online, retrieved on 2016-08-26].

[Bitc16d] Bitcoin Developer Guide. Blockchain Overview. https://bitcoin.org/en/developer-guide#block-chain, 2016. [Online, retrieved on 2016-08-05].

[Bitc16e] Bitcoin Developer Guide. Transactions Overview. http://bitcoin.org/en/developer-guide#transactions, 2016. [Online, retrieved on 2016-08-01].

[Bitc16f] BitcoinJ Developers. DefaultCoinSelector.java. https://github.com/bitcoinj/bitcoinj/blob/f416dec2eabb5941c6f7f4790cc4ef45fafd05f7/core/src/main/java/org/bitcoinj/wallet/DefaultCoinSelector.java#L34, 2016. [Online, retrieved on 2016-08-02].

[Brea16] Breadwallet. BRWallet.m. https://github.com/voisine/breadwallet/blob/847e89c8cc111639256f84fcca3766000821e7bd/BreadWallet/BRWallet.m#L461, 2016. [Online, retrieved on 2016-08-09].

[Erha14] M. Erhardt. What is the coin selection algorithm? http://bitcoin.stackexchange.com/a/29962/5406, 2014. [Online, retrieved on 2016-06-14].

[Erha15] M. Erhardt. What is meant by Bitcoin dust? http://bitcoin.stackexchange.com/a/41082/5406, 2015. [Online, retrieved on 2016-10-29].

[Hard15] D. Harding. What are the implications of Schnorr signatures? http://bitcoin.stackexchange.com/questions/34288/what-are-the-implications-of-schnorr-signatures, 2015. [Online, retrieved on 2016-08-26].

[Hava15a] R. Havar. Issue#1643: Coinselection prunes extraneous inputs from ApproximateBestSubset. https://github.com/bitcoin/bitcoin/pull/4906#issuecomment-74872926, 2015. [Online, retrieved on 2016-08-18].

[Hava15b] R. Havar. MoneyPot.com's hot wallet data. https://gist.github.com/RHavar/7cd6f3fcf2bd3e485458, 2015. [Online, retrieved on 2016-08-18].

[LoLW16] E. Lombrozo, J. Lau und P. Wuille. Segregated Witness. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, 2016. [Online, retrieved on 2016-10-18].

[Lopp15] J. Lopp. The Challenges of Optimizing Unspent Output Selection. https://medium.com/@lopp/the-challenges-of-optimizing-unspent-output-selection-a3e5d05d13ef#.gjqc5uhmz, 2015. [Online, retrieved on 2016-10-16].

[Lopp16] J. Lopp. Statoshi: Unspent Transaction Output Set. http://statoshi.info/dashboard/db/unspent-transaction-output-set, 2016. [Online, retrieved on 2016-06-15].

[Maxw12] G. Maxwell. coin selection. https://en.bitcoin.it/wiki/User:Gmaxwell/coin_selection, 2012. [Online, retrieved on 2016-10-16].

[Moor12] C. Moore. coin selection code does very badly in some cases. https://github.com/bitcoin/bitcoin/issues/1643#issue-5976682, 2012. [Online, retrieved on 2016-10-13].

[Myce16] Mycelium. StandardTransactionBuilder.java. https://github.com/mycelium-com/wallet/blob/0c7bdbb970a05beef499db0d3efbc6ed38089aa9/public/bitlib/src/main/java/com/mrd/bitlib/StandardTransactionBuilder.java#L299, 2016. [Online, retrieved on 2016-08-09].

[Naka08] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf, November 2008. [Online, retrieved on 2016-05-03].

[p2sh16] p2sh.info. Grafana - Replace by fee. http://p2sh.info/dashboard/db/replace-by-fee, 2016. [Online, retrieved on 2016-10-18].

[Ston16] A. Stone. Optimizing Bitcoin's Transaction Generation Code. https://medium.com/@g.andrew.stone/optimizing-bitcoins-transaction-generation-code-92bd222bae85, 2016. [Online, retrieved on 2016-10-18].

[Todd16] P. Todd. Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2016-May/012715.html, 2016. [Online, retrieved on 2016-10-18].

[user16] user36303. BitMonero Coin Selection. http://monero.stackexchange.com/questions/144/how-does-the-coin-selection-work-in-monero, 2016. [Online, retrieved on 2016-08-02].

[Wall12] J. Waller. Bitcoin Logo. https://github.com/jonwaller/weusecoinsjp-website/blob/master/remote/en.bitcoin.it/promotional_graphics/bitcoinLogo1000.png, 2012. [Online, retrieved on 2016-08-01].

[Wuil16] P. Wuille. UTXO breakdown per output amount. http://bitcoin.sipa.be/utxo_size.png, 2016. [Online, retrieved on 2016-07-13].