Analysis of the Bitcoin UTXO set

Sergi Delgado-Segura, Cristina Pérez-Solà,
Guillermo Navarro-Arribas, Jordi Herrera-Joancomartí

Department of Information Engineering and Communications,
Universitat Autònoma de Barcelona
{sdelgado, cperez, gnavarro, jherrera}@deic.uab.cat

Abstract. Bitcoin relies on the Unspent Transaction Outputs (UTXO) set to efficiently verify new generated transactions. Every unspent output, no matter its type, age, value or length is stored in every full node. In this paper we introduce a tool to study and analyze the UTXO set, along with a detailed description of the set format and functionality. Our analysis includes a general view of the set and quantifies the difference between the two existing formats up to the date. We also provide an accurate analysis of the volume of dust and unprofitable outputs included in the set, the distribution of the block height in which the outputs where included, and the use of non-standard outputs.

1 Introduction

Bitcoin makes use of the Unspent Transaction Output (UTXO) set in order to keep track of output transactions that have not been yet spent and thus can be used as inputs to new transactions. Bitcoin full nodes keep a copy of the UTXO set in order to validate transactions and produce new ones without having to check the whole blockchain. This allows, for instance, the use of so called pruned nodes (introduced in Bitcoin Core v0.11 [1]), which can operate without having to persistently store the full blockchain.

The UTXO set is thus a key component of Bitcoin. The format, content, and operation of this set has an important impact on Bitcoin nodes' operations. The size of the UTXO set directly impacts on the storage requirements of a Bitcoin node, and its efficiency directly determines the node validation speed.

We believe that a deep understanding of the Bitcoin UTXO set is needed to clearly understand the operation of Bitcoin, helping to find potential scalability and efficiency problems. To that end, we present STATUS (STatistical Analysis Tool for UTXO Set), a tool to analyze the UTXO set of Bitcoin. To the best of our knowledge there is no clear description in the literature of the UTXO set, its format, and how to actually analyze it. We provide such description along with a deep analysis of the set, and the tools needed to perform it.

The paper is organized as follows. Section 2 describes the UTXO set, its format, and introduces the STATUS analytical tool. Section 3 provides the actual analysis, including a general overview, the analysis of dust and unprofitable UTXOs, the distribution of the block height in which the outputs were included and the use of non-standard outputs. Finally, Section 4 concludes the paper.

2 The UTXO set

The Unspent Transaction Output (UTXO) set is the subset of Bitcoin transaction outputs that have not been spent at a given moment. Whenever a new transaction is created, UTXOs are used to claim the funds they are holding, and new UTXOs are created. Basically, transactions consume UTXOs (in their inputs) and generate new ones (in their outputs). Therefore, transactions produce changes in the UTXO set.

Since the UTXO set contains all unspent outputs, it stores all the required information to validate a new transaction without having to inspect the full blockchain. As the name already suggests, UTXOs are indeed Bitcoin outputs, and, as such, they consist of two parts: the amount transferred to the output and the locking script (`scriptPubKey`) that specifies the conditions to be met in order to spend the output.

The UTXO set is stored in the chainstate, a LevelDB database that provides persistent key-value storage. LevelDB [2] is used to store the chainstate database since Bitcoin v0.8. Apart from the UTXO set, the chainstate database stores two additional values: the block height at which the set is updated and an obfuscation key that is used to mask UTXO data [3,4]. Such an obfuscation key is used to obtain a different file signature of the UTXO set file for every different wallet in order to avoid false-positives with antivirus software.

The format of the chainstate database changed in version v0.15 of the Bitcoin Core. We will refer to the previous format as 0.14, although it has been used in versions from 0.8 to 0.14.

2.1 The UTXO Bitcoin Core 0.14 format

The chainstate database of Bitcoin Core v0.14 uses a per-transaction model: there exists a record in the database (i.e., a key-value pair) for each transaction that has at least one unspent output. Multiple UTXOs belonging to the same transaction are thus stored under the same key. The key of the record is the 32-byte transaction hash, preceded by the prefix "c". This prefix is needed to distinguish transactions from other data that are also stored in the database, and is also used to discriminate v0.14 format from the recently released v0.15[1].

The value of the record stores metadata about the transaction (version, height and whether it is coinbase or not) and a compressed representation of the UTXOs of the transaction [5].

Regarding the UTXOs, the encoding first identifies the indexes of the ouputs of the transaction that are unspent and then includes information about those outputs. The encoding is optimized to favor the first two outputs. UTXOs are then encoded taking into account their type. Six different types are specially established, that allow to efficiently store P2PKH, P2SH and four different cases of P2PK scripts. For these types, only the required data are stored since there is no need to store the full script (the type uniquely determines it). For instance, for

[1] Bitcoin Core v0.15.0 was released on 14th of September 2017.

P2PKH outputs only the address is stored. For scripts other than these specific types, the full output script is stored. Additionally, for each output regardless of its type, a compact representation of the amount of bitcoins is also stored.

2.2 The UTXO Bitcoin Core 0.15 format

One of the main changes from the last Bitcoin Core's major release (v0.15) has been a change of the internal representation of the chainstate in favor of a better performance both in reading time and memory usage [6,7].

This new format uses a per-output model in contrast to the previously defined per-transaction model, that is, every entry in the chainstate now represents a single UTXO, instead of a collection of all the UTXOs available for a given transaction. To achieve this, the key-value (known as *outpoint-coin* in the source code) structure has been modified. Keys encode both the 32-byte transaction hash and the index of the unspent output, preceded by the prefix "C". Regarding *coins*, each one encodes a code, that contains metadata about the block height and whether the transaction is coinbase or not (notice that the transaction version has been dropped), a compressed amount of bitcoins, and the output type and script encoded in the same way as the version v0.14.

Storing unspent outputs one by one instead of aggregated in a same transaction greatly simplifies the structure of the *coin* and reduces the UTXOs accessing time. By using the previous structure when a transaction with more than one unspent output was accessed all data needs to be decoded, and all the non used outputs encoded and written back into the database. However, this new format has the downside of increasing the total size of the database [7].

2.3 *STATUS*: The UTXO analytic tool

We have created *STATUS (STatistical Analysis Tool for Utxo Set)*, an open source code tool that provides an easy way to access, decode, and analyze data from the Bitcoin's UTXO set[2] STATUS is coded in Python 2 and works for both the existing versions of Bitcoin Core's UTXO set, that is, the first defined format (versions 0.8 - 0.14) and the recently defined one (version 0.15). STATUS reads from a given chainstate folder and parses all the UTXO entries into a file. From the parsed file STATUS allows you to perform two types of analysis: a UTXO based one, and a transaction based one, by decoding all the parsed information from the chainstate.

In the UTXO based analysis, apart from the data mentioned in Sections 2.1 and 2.2 that STATUS directly decodes, it also creates additional meta-data about each parsed entry, such as dust and unprofitable fee rate limit, that will be deeply analyzed in Section 3. Regarding transaction based analysis, STATUS aggregates all the parsed UTXOs that belong to the same transaction, providing

[2] It can be found under a bigger Bitcoin Tools library at `https://github.com/sr-gi/bitcoin_tools/tree/v0.1/bitcoin_tools/analysis/status`.

additional meta-data such as total number of UTXOs from a given transaction, total unspent value of the transaction, etc. Finally, STATUS uses *numpy* and *matplotlib* Python's libraries to provide several statistical data analyses and charts for all the analyzed data.

3 UTXO set analysis

In this section we analyze the UTXO set of the blockchain state at block 491,868, corresponding to the 26th of October 2017 at 13:13:38 using the STATUS tool. First, we provide a general view of the data included, regarding the total number of outputs and their size depending on the Bitcoin Core UTXO set format. We also analyze different output subsets within the UTXO set that could be interesting to measure in order to provide some hints whether a more efficient UTXO set codification could be used.

3.1 General view

Using STATUS, we can retrieve details related to the general numbers behind the UTXO set. Table 1 presents a summary of such basic facts of the analyzed UTXO set. There are 52 and a half million UTXOs in the set, belonging to more than 23 million different transactions. Although this gives an average of 2.26 UTXOs per transaction, the distribution is very skewed, with most of the transactions having just one unspent output.

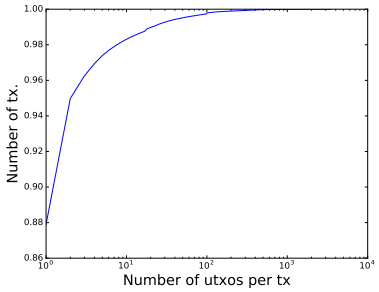
	v0.14	v0.15
Num. of tx	23,241,914	
Num. of UTXOS	52,543,649	
Avg. num. of UTXOS per tx	2.26	
Std. dev. num. of UTXOS per tx	18.27	
Median num. of UTXOS per tx	1	
Size of the (serialized) UTXO set	2.02 GB	3.00 GB
Avg. size per register	93.45 B	61.46 B
Std. dev. size per register	443.20	7.65 B
Median size per register	62	61

Table 1: Summary

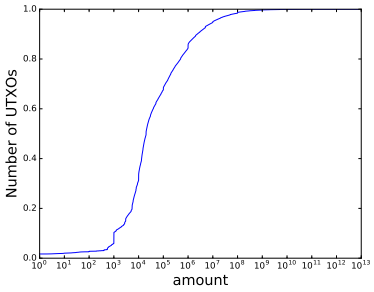
Figure 1a shows a cumulative distribution function (cdf) of the number of UTXOs per transaction.[3] Note that 87.9% of the transactions have only 1

[3] All the analysis plots included in this section show cumulative distribution functions. Therefore, a point (x, y) in the plot shows the probability y that a given variable (depicted in the x axis label) will take a value less than or equal to x.

UTXO[4] and 94.97% have less than 3. The maximum number of UTXOs per transation is 3,452 [8] which originally had 5,419 outputs.



(a) Number of UTXOs per transaction



(b) Amount per UTXO (in satoshi)

Differences between both data formats (v0.14 and v0.15) are clear regarding the serialized UTXO set size (see Table 1). While the v0.14 format uses 2.02GB with an average size per record of 93.45 bytes (a total of $23,241,914$ records), the 0.15 format expands the information to 3.00GB which represents an average size per record of 61.46 bytes (with $52,543,649$ records). Such a difference is due to the way outputs are stored in both formats, as detailed in Section 2. However, the median size per register of both versions is very similar, with most registers occupying between 59 and 64 bytes. Such measurement is sound since both versions store the 32-byte transaction id and some identifier of the output, so the size difference for every register is only significant when the transaction has more than a single UTXO. Whereas the number of registers with less than 59 bytes is negligible (just 30 of them for v0.14 and 222 for v0.15), 83.25% of them in v0.14 and 99.0% in v0.15 are $\leq$ 63-byte long.

As a matter of fact, the smallest stored register in v0.14 is just 41-byte long [9] and contains a single non-standard UTXO with a 1-byte length script containing an invalid opcode. This UTXO is also one of the smallest registers in v0.15, with 40 bytes (12 additional registers have also the same size in v0.15). Section 3.4 provides an exploration of non-standard transactions in the UTXO set.

Another interesting information of the UTXO set that can be retrieved with STATUS is the amount of UTXOs of each type, as detailed in Table 2. Notice that UTXOs are classified between the different standard types also providing a distinction between compressed and uncompressed keys for the P2PK type. As data show, more than 99% of the UTXO set are P2PKH and P2SH outputs, being P2PKH the vast majority of stored outputs. In Section 3.4 we provide detailed information regarding the 0.8% of UTXOs classified as others.

[4] Notice that such measure indicates that, although the average number of outputs in regular Bitcoin transactions is higher, the number of outputs that remain unspent is, mostly, only one.

Num. of utxos	52,543,649	100%
Pay-to-PubkeyHash (P2PKH)	43,079,604	81.99%
Pay-to-ScriptHash (P2SH)	8,987,799	17.11%
Pay-to-Pubkey (P2PK)	66,759	0.12%
Compressed	29,977	0.06% (44.90%)
Uncompressed	36,782	0.07% (55.10%)
Others	409,487	0.8%

Table 2: UTXO types

Figure 1b provides information about the amount of satoshi deposited in each UTXO, showing that 98.46% of the UTXOs store less than one Bitcoin, with an average of 0.32฿ per UTXO.

3.2 Dust and unprofitable UTXOs

An interesting type of outputs included into the UTXO set are those whose economical value is small enough to represent a problem when they have to be spent. One well identified type of these UTXOs is tagged as dust. According to the Bitcoin Core reference implementation [10], **a dust output** is the output of a transaction in which the fee to redeem it is greater than 1/3 of its value. Besides this well known definition we also define **an unprofitable output** as the output of a transaction that holds less value than the fee necessary to be spent, resulting in financial loses when used in a transaction.

In order to identify both types of outputs, it is important to recall that the amount of fees a transaction has to pay to be included in a new block depends on two factors: the fee-per-byte rate that the network is expecting at the time of creating the transaction and the size of the transaction. The fee-per-byte rate, measured in satoshi, is a highly variable factor that depends on the transaction backlog (i.e. how many transactions are pending to be included in new blocks).

Since fees depend on the transaction size, in order to label the outputs in the UTXO set as a dust or unprofitable, we need an estimation of the size of data needed to spend such output. In order to identify the minimum information needed, we can consider an already standard transaction with its inputs and its outputs and enough fees to be relayed. Then, we define the **minimum-input of a UTXO** as the smallest size input that spends such UTXO. The size of such minimum-input, together with the value held in the output and the fee rate, will determine whether a UTXO may be included into the dust or unprofitable categories.

In order to measure the size of such minimum-input, we need to review the structure of a Bitcoin transaction. As depicted in Figure 2, all transactions follow a standard structure containing some fixed length parameters that determine a minimum transaction size, and some variable length parameters, depending on the transaction type. When a transaction is created, inputs are defined referring to some UTXOs. Such inputs have different size depending on the output type

they are related to. On the other hand, new outputs are generated for every new transaction, and thereby some additional size, which will depend on the new output type, will be added to the transaction.

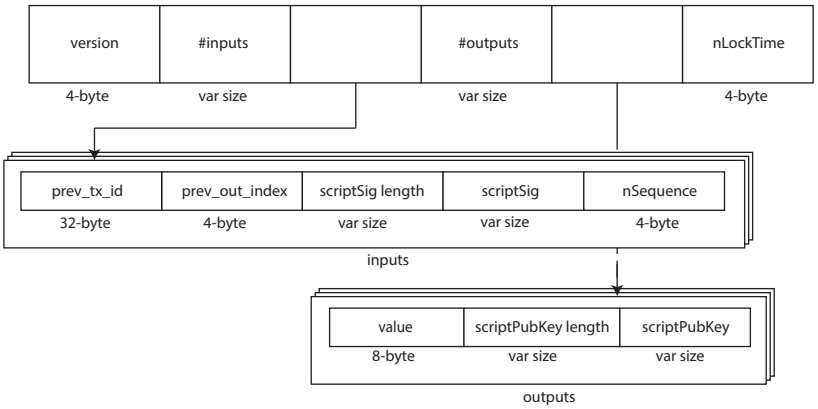


Fig. 2: Generic transaction structure

Depending on the UTXO type, its minimum-input size will be different. Such measure can be split in two parts: fixed size and variable size. Regarding the fixed size, as depicted in Figure 2 (taking into account only the input box), we can identify three fields: `prev_tx_id`, `pev_out_index` and `nSequence`. Therefore, for every UTXO, its minimum-input will be at least 40-byte long independently of its type. On the other hand, the content and length of the fields `scriptSig` and `scriptSig lenght` depend on the UTXO type, specified in the field `scriptPubKey` of the UTXO.

The different types of outputs, with their corresponding size, can be classified as follows:

Pay-to-PubKey (P2PK) outputs: The minimum-input of this type of UTXO specifies just a digital signature to redeem the output and the `scriptSig` includes the following data:

```
PUSH  sig (1 byte) + sig (71 bytes)
```

Bitcoin uses DER encoded ECDSA signatures in the scripts of its transactions, which can be between 71 and 73 bytes long depending on their `r` and `s` components. Such variability comes from the randomness of the `r` parameter, so by iterating the signature generation it is possible to craft an specific signature within 71 bytes.[5] Hence, minimum-input size for a P2PK UTXO will be

[5] Notice that this procedure assumes, in contrast to the normal behaviour of standard wallets, that the ECDSA implementation does not use a deterministic function to compute `r`.

71-bytes long and `scriptSig len` field will be 1-byte long, so a total of 72 bytes.

Pay-to-PubkeyHash (P2PKH) outputs: For this UTXO to be redeemed, both a signature (sig) and a public key (pk) are needed in the `scriptSig`, as shown below:

```
PUSH  sig (1 byte) + sig (71 bytes) + PUSH pk (1 byte) +
pk (33-65 bytes)
```

Regarding the signatures, the same assumptions as for P2PK outputs applies, that is, 71-byte length can be considered. Regarding public keys used by Bitcoin, they can either be compressed or uncompressed, which will significantly vary their size:

- Uncompressed keys: Such keys were used, by default, in the first versions of the Bitcoin core client, and they are 65-byte long.
- Compressed keys: By 30th March 2012 (around block height 173480) Bitcoin core started using this more efficient type of keys, which are almost half size of the previous ones (33 bytes), and therefore make smaller scripts.

So, the size for the `scriptSig` varies from 106 to 138 and then the `scriptSig length` field will be 1-byte long, resulting in a total minimum-input size between 107 and 139 bytes.

Pay-to-multisig (P2MS) outputs: the size of the minimum-input to redeem such a script highly varies depending on the number of signatures required, which ranges up to 20 (20-of-20 multisig)[6], so the `scriptSig` for redeeming such output is as follows:

```
OP_0  (1 byte) +  (PUSH  sig (1 byte) + sig (71 bytes)) *
required_signatures (1-20)
```

Thus, the size of the `scriptSig` field will range between 73 and 1441 bytes, making the `scriptSig len` field range between 1 and 2 bytes, so the total minimum-input size will be between 74 and 1443.

Pay-to-ScriptHash (P2SH) outputs: unlike any previous output type, input size created from P2SH outputs can not be straightforwardly defined in advance. P2SH outputs hide the actual input script behind a hash, in order to make smarter outputs, by making them smaller and thus, allowing the payer to pay lower fees. However, the scripts held by those UTXOs give us no clue about how the minimum-input should be build.

Table 3 summarizes the sizes of the minimum-input for each UTXO type.

[6] Although the standard considers a maximum number of 3 signatures in a P2MS output, up to 20 are valid regarding the consensus rule [11] so they could potentially be found in the UTXO set.

UTXO type	Fixed size	scriptSig length	scriptSig			Total size
			sig	pk	push data	
P2PK	40	1	71	-	1	113
P2PKH	40	1	71	33-65	2	147-179
P2MS	40	1-2	71-1420	-	2-21	114-1483
P2SH	40	?	?	?	?	40-?

Table 3: minimum-input size summary

Notice that the previous analysis does not take into account the new SegWit transaction format [12]. The minimum-input size for such type of outputs needs an extended analysis. However, at present time, the total outputs in the UTXO set that correspond to a SegWit output is upper bounded by a 2.26% (see Section 3.1 and Section 3.4) so, giving such small amount of data, the results presented here will not significantly change, we leave such analysis for further research.

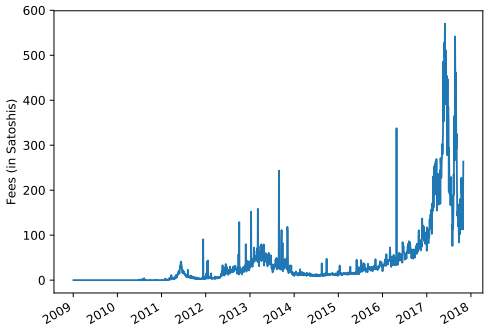


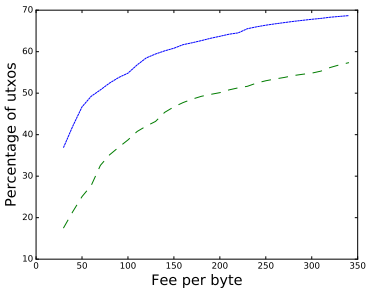
Fig. 3: Evolution of fees (Source: Blockchain (`https://www.blockchain.info`).

Once we determined the amount of data of the minimum-input for each type of UTXO, based on a defined fee-per-byte rate, we can identify those outputs form the UTXO set that fall into both the dust and the unprofitable categories. To obtain the data, the following considerations have been taken. The minimum-input size for P2PK and P2MS outputs have been precisely computed since the information to determine the exact size of the minimum-input can be derived from the output data itself. However, it is not possible to exactly determine such value for the P2PKH neither for the P2SH. In the first case, we have taken the following approach. For those outputs up to block 173480 we have considered uncompressed addressed and for the newer ones we have take the of most conservative approach, assuming that all public keys from that point

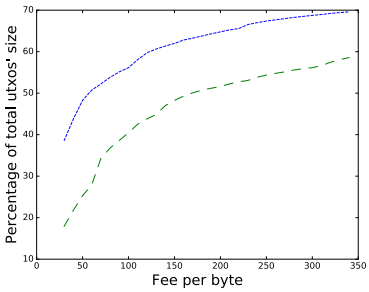
onward are in compressed form (33 bytes) so reducing the number of UTXO that fall into both categories. For the P2SH, being not able to set a proper lower bound for the variable part, we have performed the analysis assuming only the fixed 40 bytes.

Finally, the last parameter to set is the fee-per-byte rate. As depicted in Figure 3, such rate is far from fixed and has hight variability. Thus, in order to measure different possible scenarios, we have considered a wide fee-per-byte spectrum, ranging from 30 to for 340 satoshi/byte.

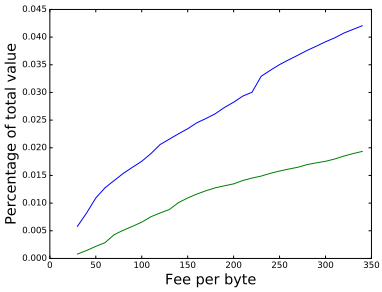
The volume of both dust outputs and unprofitable outputs (blue and dotted green lines respectively) in the UTXO set are depicted in Figure 4.



(a) % of dust/unprofitable UTXOs w.r.t. fee-per-byte.



(b) % of occupied space w.r.t. fee-per-byte rate.



(c) % of economic value w.r.t. fee-per-byte rate.

Fig. 4: Dust and unprofitable analysis (blue and green lines respectively).

Figure 4a shows the relative size of dust and unprofitable output sets within the total UTXO set. Notice that for a fee-per-byte as small as 80 satoshi/byte onwards, more than the 50% of UTXOs (26.29 million outputs) from the set can be considered dust, whereas the same 50% size for the unprofitable set is reached for 240 satoshi/byte onward. Regarding the size that such data, Figure 4b shows how those UTXOs represent a relevant part of the total size from the set (more

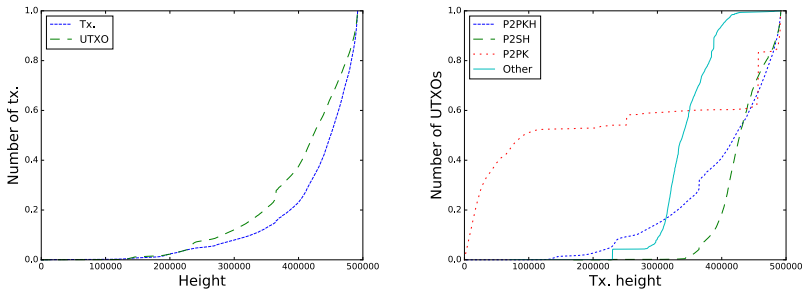
than the 50% for around 70 satoshi/byte onward), while the same can be seen for unprofitable UTXO for a rate of 200 satoshi/byte onward. Finally, from an economic point of view, Figure 4c shows, as expected, how those dust and unprofitable UTXOs represent a negligible amount from the total value of the UTXO set, that is the total number of bitcoins in circulation.

3.3 Height

Another interesting type of UTXO outputs are those that were created a long time ago. Although it is difficult to determine the average time in which a UTXO will be spent, some old UTXOs may belong to keys that are lost, so such old UTXOs may will never be spent.

Figure 5a depicts the height of the block where the transaction is included in a per transaction (v0.14 register, blue line) and per UTXO (v0.15 register green line) fashion. Half of the stored UTXOs are older than January 2017 (block 449,896 corresponds to the median), whereas the other half are younger. This means that almost half of the current UTXO set is used by UTXOs created in the current year (2017). On the other hand, there are still very old UTXOs: 2% of them are older than August 2012 (block height 194,635).

In Figure 5b we can see the evolution in time of the different types of outputs in the UTXO set. Notice that P2PKH and P2SH show a stable distribution in time. On the other hand, outputs labelled as "others" are mainly from old transactions since 95% of them are older than March 2016 (block height 403,052). Finally, the graphic also shows that P2PK outputs have an irregular behaviour. 50% of them were created before block 91,542 which is an expected result since P2PKH were developed afterwards as an improvement of P2PK. However, it is interesting to see that, after a long time with very few outputs of this type, around March 2017 and during 324 blocks, 15% of the actual P2PK outputs included in the UTXO set were created.



(a) Height per transaction/UTXO. (b) UTXO type evolution by height.

Fig. 5: Output age-based analysis

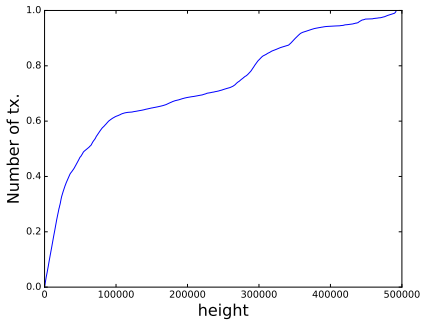


Fig. 6: Coinbase evolution by height.

Figure 6 shows an already known fact that indicates that most of the bitcoins created at the beginning of the cryptocurrency are still pending to redeem. More precisely, 75% of the coinbase outputs in the UTXO set were created before block 274,946 (December 2013). In contrast, just 6% of the current UTXOs were created before that block (see Figure 5a).

3.4 Non-standard

As shown in Table 2, we have labelled as "others" 409,487 UTXOs from the UTXO set since they do not fall into the main categories P2PK, P2PKH and P2SH. A detailed analysis of such UTXOs, provided in Table 4, shows that almost all UTXOs correspond to a Pay-to-Multisig (P2MS) outputs being the configuration of 1-2 and 1-3 the most popular cases. Notice that, the UTXOs included are those with configuration up to three public keys, which is sound according to the fact that this is the upper bound for a multisignature output to be considered standard by the Bitcoin network transaction relaying policies. Finally, it is worth to mention that there exist 828 UTXOs with 1-1 configuration, a fact that does not make much sense since it is an output with functionality equivalent to a P2PK but with a larger script size and so highers fees may be needed to spent it.

Regarding the 1,169 outputs labeled as others in Table 4, 34.05% of them (398) are new native SegWit type outputs. More precisely, Pay-to-Witness-Public-Key-Hash (P2WPKH) account for 40 outputs and Pay-to-Witness-Script-Hash (P2WSH) accounts a total of 358.

4 Conclusions and Further research

In this paper we have introduced STATUS, a tool to analyze the UTXO set of Bitcoin (based on the Bitcoin Core implementation), and we have provided an analysis of such set, paying special attention to dust and unprofitable transactions. We have also provided a detailed description of the UTXO set format,

1-1	828	0.20%
1-2	199,904	48.81%
2-2	1,353	0.33%
1-3	206,096	50.33%
2-3	117	0.02%
3-3	20	0.005%
Others	1,169	0.28%

Table 4: Multisig analysis.

including the new format introduced in Bitcoin Core v0.15. The use of this format as compared to the previous one does not have an impact on the analysis we have presented in this paper. The new version provides more efficient access to the UTXO information at the expense of slightly higher storage requirements. Additionally, we provide interesting data that shows the high percentage of "static information" (in the sense that is not going to be spent -dust and unprofitable-) included in the UTXO set that reduces the efficiency of the database in terms of space. Finally, it is interesting to notice that currently there is a very low percentage of SegWit UTXO, upper bounded by a 2.26% of the total outputs stored in the UTXO set. As this will possibly increase in the future, the analysis of dust and unprofitable transactions will need to be revisited in further research in order to update the results with these new types of outputs.

Acknowledgements

This work is partially supported by the Spanish ministry under grant number 785 TIN2014-55243-P and the Catalan *Agència de Gestió d'Ajuts Universitaris i de Recerca* (AGAUR) grant 2014SGR-691.

References

1. Bitcoin Core. Bitcoin core 0.11.0 release notes. `https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md`, July 2015.
2. S Ghemawat and J Dean. Leveldb. `https://github.com/google/leveldb` (last accessed Oct. 2017), 2014.
3. Bitcoin Core. Bitcoin core 0.12.0 release notes. `https://bitcoin.org/en/release/v0.12.0`, February 2016.
4. Bitcoin Core. Obfuscate database files. Bitcoin Core Github Issue 6613, `https://github.com/bitcoin/bitcoin/issues/6613`, July 2015.
5. The Bitcoin Core developers. Bitcoin core 0.14 source code: `coins.h`. Github: `https://github.com/bitcoin/bitcoin/blob/0.14/src/coins.h`, 2017.
6. Bitcoin Core. Bitcoin core 0.15.0 release notes. `https://bitcoin.org/en/release/v0.15.0`, September 2017.

7. Greg Maxwell. A deep dive into bitcoin core 0.15. SF Bitcoin Developers Meetup, September 2017. http://diyhpl.us/wiki/transcripts/gmaxwell-2017-08-28-deep-dive-bitcoin-core-v0.15/.
8. Blockcypher. Bitcoin transaction d8505b78a4cddbd058372443bbce9ea74a313c27c586b7bbe8bc3825b7c7cbd7. https://live.blockcypher.com/btc/tx/d8505b78a4cddbd058372443bbce9ea74a313c27c586b7bbe8bc3825b7c7cbd7/ (last accessed Oct. 2017).
9. Blockcypher. Bitcoin transaction 8a68c461a2473653fe0add786f0ca6ebb99b257286166dfb00707be24716a3f. https://live.blockcypher.com/btc/tx/8a68c461a2473653fe0add786f0ca6ebb99b257286166dfb00707be24716af3a/ (last accessed Oct. 2017).
10. Bitcoin Core developers. Bitcoin core 0.10.0rc3 source code: transaction.h, line 137. Github: https://github.com/bitcoin/bitcoin/blob/v0.10.0rc3/src/primitives/transaction.h#L137, December 2014.
11. Peter Wuille. Answer to: What are the limits of m and n in m-of-n multisig addresses? Bitcoin StackExchange, 2014. https://bitcoin.stackexchange.com/a/28092/30668.
12. Eric Lombrozo, Johnson Lau, and Pieter Wuille. Segregated witness (consensus layer). Technical Report BIP-141, Bitcoin Improvement Proposal, 2015.