An Efficient Miner Strategy for Selecting Cryptocurrency Transactions

Chukwuocha Chukwuka Chibundo, Saulo dos Santos, Shahin Kamali, Ruppa K. Thulasiram
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada
{chukwuoc, dossants, kamalis, tulsi}@cs.umanitoba.ca

Abstract—

Cryptocurrencies like Bitcoin use the blockchain technology to record transactions in a distributed and secure way. Each block contains a cryptographic hash of the previous block in addition to a set of completed transactions that it records. The first step for a miner to add a new block to the blockchain is to select a set of pending transactions from a mempool. The total size of selected transactions should not exceed the fixed capacity of blocks. If a miner completes the computationally-hard task of finding the cryptographic hash of the formed block, the block can be added to the blockchain in which case the transactions in that block will become complete. Transaction might have a fee which is granted to the miner that has added the transaction to the blockchain. As such, when forming a new block, miners tend to select transactions that offer the best fees. Finding a set of transactions with maximum total fee that fit into a block translates to the classic knapsack problem, which is an NP-hard problem. Meanwhile, miners are in fierce competition for mining blocks and hence cannot dedicate too much computational power for selecting the best set of transactions. Most existing solutions to tackle this problem are based on sorting the set of pending transactions by the ratio between their fee and their size. While the sorting step is not a bottleneck in normal situations, the transaction can grows explosively in case of a market turbulence like that of 2017. Meanwhile, the number of transactions in the mempool has been increasing over time. As such, it is desirable to have an efficient strategy that do not rely on sorting transactions before forming a block.

In this paper, we review some of the existing strategies for selecting transactions. We also introduce a robust solution for selecting transactions that does not use sorting. We perform a theoretical and experimental analysis of our solutions to compare it with other strategies. Our results indicate that our algorithm runs faster than previous solutions, and the quality of its blocks (the total rewards in its blocks) is comparable (in fact, slightly better) than the best existing solution.

1 Introduction and Background

The use of cryptocurrency for financial transactions have risen significantly in recent years and has become a trending topic in the financial and technological sectors of the world. Bitcoin which was invented by Satoshi Nakamoto in 2008 is considered the most popular cryptocurrency as it introduced the decentralized peer-to-peer system for which cryptocurrency is associated with today [1]. The blockchain technology is defined as a decentralized public ledger that contains blocks which are made up of transactions [2]. Decentralization implies the elimination of a central authority such as the central bank in the case of financial transactions. However, one might ask how the transactions in the blockchain are validated and regulated. This is made possible by a group of network nodes called miners.

When a user carries out a transaction into the blockchain network, the transaction is broadcast to all miners and each one of them keeps a memory pool (mempool) which contains all unconfirmed transactions waiting to be validated. This validation process involves checking if the transaction is using a valid UTXO (Unspent Transaction Output) [3] and ultimately the selection of the valid transactions, which pay higher fees, from the memory pool to form a block. In order to insert a block into the blockchain the miner must aside of validated if the transactions in the block are valid also solve a complex problem called proof of work which requires a lot of computational power [1]. Each block contains its unique proof of work [4] and it is based on the hash of the most recent block in the blockchain. This mechanism prevent miner to attempt solve a block in advance. The miner that solves his proof of work first gets his block added to the blockchain. Each block inserted into the blockchain gives a reward to the miner, which is known as an incentive or mint process. Each time a block is inserted into the blockchain, new Bitcoins are created in the Bitcoin network. The Bitcoin protocol asserts that 21 million bitcoins will be mined until the year 2040, being this reward halved after 210k blocks being mined, this behaviour reduce the offer of bitcoin in the system increasing the value of this asset. As per proposed by Satoshi Nakamoto [1] the minting process in the beginning should be valuable enough to support the cost of mining process, and bring interested to the technology fading with time and become more and more rare, when the asset would have its own value and the mining will be support only by fees [5] paid in each transactions.

Most transactions rely on fees as an extra incentive in order to be selected for the next block and get confirmed quickly.

Its clear that the goal of a miner is to select the transactions that maximize his profit as quickly as possible while adhering to the size constraint of a block in the blockchain which

is in the case of Bitcoin [6] 1 megabyte.

This forms the first part of our problem statement and it is given formally as follows.

- There is a set of n transactions in the mempool and each transaction i has a size of s_i and a value of v_i.
- The block in which the transactions are put into by the miner has a maximum capacity of S which is $1.000.000 vbytes$ in this case.
- The goal is to maximize profit $\sum_{i=1}^{n} v_i$ without exceeding the weight of the block $\sum_{i=1}^{n} s_i \leq S$.

From the statements above we can infer that this is a weighted knapsack problem. The knapsack problem is a very popular problem in computer science with the goal being to maximize value under a size constraint when given a set of items that each have values and weights. Despite the numerous offline algorithms available to solve the knapsack problem we will focus on a set of algorithms that aim to solve the problem with focus in both time complexity and profitability. As miners are competing against each other to solve the PoW first the main goal of a transaction selection algorithm is to minimize time and most of the optimal offline algorithms are time consuming. Therefore, we have the following conditions for our problem statement.

- The miner maintain a local mempool, which is constantly fed with new transactions broadcast in the network.
- The miner has to choose the transactions that will be inserted into the block from the Mempool.
- The goal of the miner is still to select a set of transactions from the Mempool as fast as possible which maximize total fee value collected without exceeding the size of his block S.

The major contributions of this work are :1) Time complexity study and implementation of various algorithms suitable for transaction selection to provide a baseline for our models. 2) Creating and implementing an online advice model that uses the historic of block fees as a baseline. 4) Proposition of a new offline algorithm based on a $DensitSizeTable(DST)$ for selection of transations from the Mempool. 5) Compreensive evaluation and comparison of the performance of the 4 different algorithms for transactions selection. 6) Suggestion of a new algorithm DST which outperform the actual Heap sort used on Bitcore [**?**]

The rest of the paper is arranged as follows. In section 2 we present the related work on the implementation of online algorithms and advice models for the weighted knapsack problem. In section 3 we present the methodology and experiments we carried out on the bitcoin dataset using the offline, online and semi-online algorithms. In Section 5 we present results form our experiments and discuss them in detail. Finally, in Section 6 we conclude our work with some suggestions for future work.

2 Related Work

The offline knapsack problem was among the first few computing problems considered solvable and was defined as NP hard by Karp [7]. It is widely regarded as a pseudo polynomial time problem using dynamic programming. Marchetti-Spaccamela and Vercellis [8] first considered its online setting and concluded that no pure deterministic online algorithm for the knapsack problem was competitive. This result made researchers take liberties and postulate assumptions while considering the online setting of the classic weighted knapsack problem which led to the development of several variants of the problem. The online removable variant of this problem where items already placed in the bin could be removed later was postulated by Iwama and Taketomi [9].

Dobrev et al [10] postulated that the only way to make these online algorithms competitive was to introduce an advice model. A lot of different advice models were introduced to solve different variants of the online knapsack problem and Bckenhauer et al [11] came up with the theorem that for an online algorithm of the classic weighted knapsack problem to come close to the optimal solution it needs at least $O(\log n)$ bits. Bckenhauer et al [12] also proved that for the unweighted variant of the knapsack problem, a single bit could make its competitive ratio 2 however in this work our focus is more on the weighted variant.

The public nature of the bitcoin blockchain has led to several researchers focusing on the transactions that occur in the blockchain and the mechanisms behind its selection. In its early stages the transactions for the next block were predefined and the choice wasn't for the miners to make however with recent significant advancements, transaction selection has become entirely up to the miner [13]. Selecting transactions from the mempool is usually defined by a policy [14] which by default involves ranking the transactions from largest to smallest in terms of fee-per-kilobyte. Then transactions are selected in a greedy format until size limit of the block is reached. Most researchers have tried to improve on this selection policy by using feature extraction techniques. [15] Fiz modelled this selection policy as a classification problem and proved that the most important features of the transactions that can be used to solve this was their size and fee value.

For our work, We assume we are given a set of n transactions each given as a tuple (key, size, density). Key is the unique identifier while weight and density are the real attributes of the transaction. The size of a transaction is a value in the range (0,1] which indicates the portion of a block that the transaction will occupy (divide the actual size by the maximum 1M vbytes capacity of the block). Density is a positive value which indicates the ratio between the benefit (rewards) and the size of the transaction. From the historical data we can estimate an upper-bound P for the density of a majority of transactions. The *scaled density* of a transaction is the ratio between the actual density of the transaction and P. All transactions, except an outlier number of them, have scaled density in the range (0,1] while the outliers have scaled density larger than 1 (they are more desirable since their reward is unproportionality large compared to their size). In the remainder of this section, we use "density" to refer to "scaled density".

Given a set of n transactions with different weights and densities, we would like to select a subset of them with size at most 1. Recall that size of all transactions is scaled

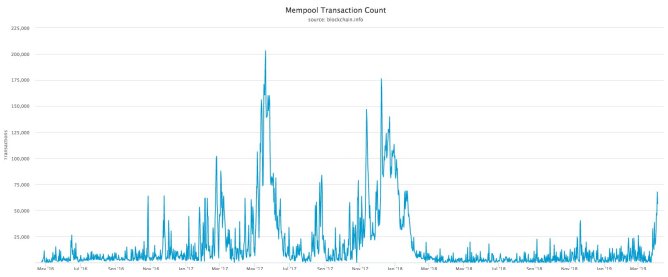

Fig. 1. Sample Input of the bitcoin unconfirmed transactions dataset

into range (0,1] and the size of a block is assumed to be 1. The objective is to select a subset of subset of transactions whose total reward is maximized. This clearly maps to the knapsack problem which is known to be NP-hard. In what follows, we review a few practical algorithms for this problem. As we will see, these algorithms all perform poorly in the worst-case scenarios. We define *quality ratio* of an algorithm as the ratio between the sum of the profits of the algorithm and that of an optimal algorithm. Quality ratio serves as a measure for the quality of algorithms.

3 Methodology and Experiments

Our methodology, algorithms and experiments are discussed in this section as follows:

3.1 Input Gathering

For any experimental analysis, obtaining the input for the experiment might be the most important part of the project. This is because the input dictates the accuracy and legitimacy of the results of the work. In order to carry out this analysis, we needed to find a way to obtain a bitcoin mempool data in our local system and this proved a herculean task because most researchers usually deploy a custom node which requires download the whole blockchain data (roughly 208 GB) for carrying out these experiments [13]. However, our approach was to use the data provided by [16], which a bitcoin transactions dataset is provided in text format and contained about 21,000 unconfirmed transactions that were in a mempool in the month of November. Each transaction in our input data has the following characteristics: Mempool number, hash Id, time, input count, output count, input total (in btc), output total (in btc), fees (in btc), fees (in dollars) and size (in bytes). Obviously for the purpose of our experiments we focus on the fees (in dollars) and size (in bytes) which are the last two columns in our input data. A general sample of the input is given in Figure 1.

3.2 Study and implementation of the algorithms

When describing an algorithm, in addition to the quality of their solution, we will have an eye on their time complexity. In addition to the time that it takes to select transactions, we consider pre-processing time (the time that it takes to build the underlying data structure from the pool of transaction) and the update time (time that it takes to insert/delete items from the mempool).

Algorithm 1 Online greedy algorithm for the blockchain knapsack problem

```
for Each transaction do
    if (size ≤ Blocksize) then
        accept transaction
        update Blocksize
    end
    else
        reject transaction
    end
end
```

3.2.1 Greedy Algorithm

The simplest solution is to process transactions one by one and greedily choose any one that fits. This algorithm is oblivious to the density of transactions. Clearly, this algorithm performs quite poorly in the worst case. For example, if the first transaction has size 1 and density almost 0, it will select it while there might be relatively small transactions with arbitrarily larger rewards. In fact, any 'online algorithm' which places items without any priory information about them can be arbitrarily bad. In terms of speed, the algorithm runs in $O(n)$ for processing transactions (it might need to check all transactions). Note that there is no pre-processing time, and update queries can be performed in $O(1)$. The algorithm for this is given in Algorithm 3.2.1.

3.2.2 Greedy Algorithm with Advice

This algorithm works almost identical to the greedy algorithm except that it receives certain information from an omniscient oracle [11]. Our advice model consist of the density α that is obtained from the total fees $OptValue$ collected from a block mined by an optimal solution divided by the $BlockSize$. This setup only accepts transactions which have density larger than the one indicated in the advice. The quality of the solutions are expected to improve over the greedy algorithm. But the presence of outliers result in the unbounded quality measure (elaborate plz). The running time will be similar to the greedy algorithm. (It will never reduce the density overtime, if it does not have a mechanism to accept lower density transactions in case of a huge amount of space available and a few transactions in the network. Discuss it)

3.2.3 Sorting based algorithm

This is the algorithm used in practice. Similar to the sorting approach, a heap is maintained that maintains transactions by their density. To form a block, the largest transaction (the one with maximum density) is repeatedly extracted and placed in the block (if the block has space for it). If the block does not have space for an item extracted from the heap, it is stored in an array of rejected transactions. When the number of rejected transactions becomes 50, the process of selecting transaction is aborted. Note that this strategy also has unbounded quality measure (elaborate). Forming the heap can be done in $O(n)$ (pre-process time) while the update queries each take upto $O(\log n)$ time.

p.s1. please experience with values other than 50. p.s2. Implement this algorithm using heap.

Algorithm 2 Semi-online Algorithm for the blockchain knapsack problem

$\alpha = Optvalue/Blocksize$
for *Each transaction* **do**
  $Profitdensity = fee/size$
  if $(size \leq Blocksize)$ **then**
    if $(Profitdensity \geq \alpha)$ **then**
      accept transaction
      update $Blocksize$
    end
    else
      reject transaction
    end
  end
  else
    reject transaction
  end
end

Algorithm 3 Offline greedy algorithm for the blockchain knapsack problem using profit density

for *Each transaction in the mempool* **do**
  $Profitdensity = fee/size$
end
Arrange transactions in descending order of $Profitdensity$
for *Each transaction* **do**
  if $(size \leq Blocksize)$ **then**
    accept transaction
    update $Blocksize$
  end
  else
    reject transaction
  end
end

4 Density Size Table (SDT)

We introduce a new mechanism called Density-Size Table (SDT) approach for transaction selection. The algorithm has two parameters k_1, k_2 (e.g., $k_1 = k_2 = 50$).

We partition transactions into different classes based in their size and densities. A transaction is in size-class s iff its size is in the range $(s/k_1, (s+1)/k_1]$; here s is an integer in the range $[0, k_1)$. Similarly, a transaction is in density-class d if its density is in the range $[d/(k_2-1), (d+1)/(k_2-1))$ for $d \in [0, k_2-1)$. We define a special density class $k_2 - 1$ for outlier transactions with density larger than 1.

SDT maintains a matrix of size k_1, k_2; the index (i, j) of this matrix holds a dictionary that include all transactions of size-class i and density-class j. We assume this dictionary is maintained using a hash table.

Given a mempool of n transaction, the first step of SDT is to pre-process transactions and places them in the table. For each transaction x, finding the size-class s and density-class d can be done in $O(1)$. The insertion into the dictionary at index (s, d) takes constant time (assuming a reasonable hash table). So, in total, inserting all n transactions to the table takes $O(1)$. Note that insertion and deletion of transactions form the data structure also takes $O(1)$ per transaction (find the right index in the table and apply insert/delete to the dictionary in constant time).

In what follows, we explain how transactions are selected for a new block. The algorithm works in "rounds". Each round involves selecting one transaction. At the beginning of each round, the block has a remaining capacity cap (at the first round we have $cap = 1$). Let si (size-index) denote $cap \times k_1$. All transactions with size-class smaller than si fit in the block (given its current capacity) while any transaction with size-class larger than si cannot fit in the block. Transactions with size-class si might or might not fit in the block (their size might be less than or more than cap).

In order to select the next transaction, let j initially be equal to $k_2 - 1$. Here j will be an iterator over density classes, and initially points to the class with maximum density (i.e., k_2). We iterate i on size classes from 0 to $si - 1$ inclusive. If the dictionary at index (i, j) is non-empty, any transaction from that dictionary is selected as the next transaction in the block, it is deleted from its dictionary, and the round is terminated. Otherwise (when all the dictionaries were empty so far), the transactions at index (i, s_i) are probed; here for each probed transaction, we should check whether it fits in the block (its size is less than cap). As soon as a transaction that fits is found, the round ends (the selected transaction is removed from its dictionary and added to the block). Finally, if the dictionary at index (i, s_i) does not contain an item that fits in the block, j is decremented and the whole process is repeated. Decrementing j implies that we did not find a transaction with density-class j; so, we decrement j to search among transactions with smaller density.
Algorithm 4 illustrates the transaction selection process.

4.0.1 Quality and Time Analysis

The quality is almost as good as sorting algorithm (a constant away from it depending on the size of dictionaries). The time is close to constant (at least when sizes are almost uniformly distributed). We elaborate these later.

5 Experimental Results

DO A REVIEW OF THIS ENTIRE SECTION¡DATA OUTDATED¿

For our experiments, we executed each of the algorithms listed above on a local system using python. The benchmark as stated earlier was a bitcoin mempool data from November 2018. The dynamic programming algorithm wasn't able to be executed due to the size of our input dataset and the memory capacity of our local system. So, it was left for future tests on more capable systems. We used the offline profit-density greedy algorithm as our optimal baseline in the tests we ran on the input dataset. The offline profit-density algorithm provided a better profit value for the miners than the other algorithms but was the slowest because it is an offline strategy.
The pure online algorithm had the lowest profit value but was faster than all the offline strategies. The advice model had a better profit value than the online greedy algorithm and was also the fastest algorithm of the three proving that in the online setting, the best algorithms for our blockchain transaction knapsack problems are the online algorithms with advice. Our results are summarized in the tables below.

Algorithms	# of Transactions in mempool	# of Transactions selected	Profit value in dollars
Offline profit density greedy algorithm	21,709	2398	2033.8212 dollars
Online greedy algorithm	21,709	2095	1649.0934 dollars
Semi-Online algorithm with advice	21,709	2235	1872.6253 dollars

TABLE 1
Quality analysis of the various algorithms in terms of profit value

Algorithms	# of Transactions in mempool	# of Transactions selected	Time of execution
Offline profit density greedy algorithm	21,709	2398	5.4564 secs
Online greedy algorithm	21,709	2095	0.1506 secs
Semi-Online algorithm with advice	21,709	2235	0.1496 secs

TABLE 2
Performance analysis of the various algorithms in terms of speed

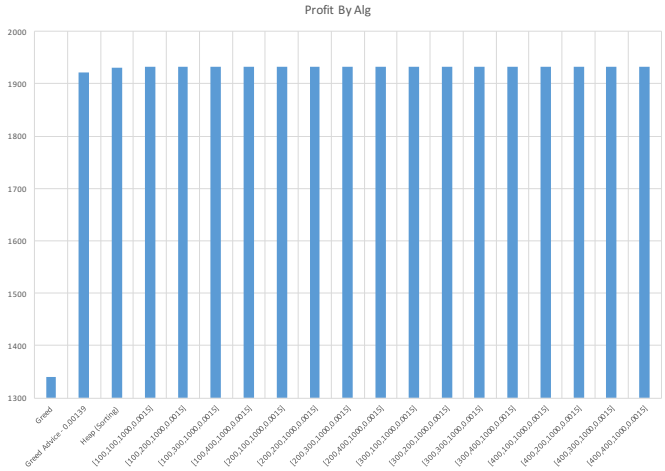

Fig. 2. Amount of fee collect by each algorithm for the same set of transaction in the Mempool.

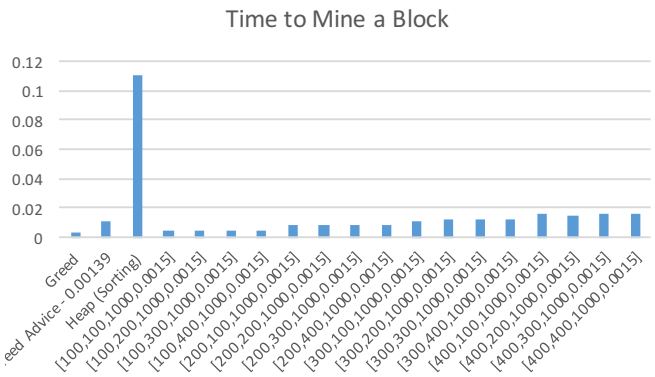

Fig. 3. Average time to form a block based on a set of transaction already in the Mempool data structure.

Table 1 summarizes and compares the quality of our various algorithms in terms of the profit value obtained by the miner which is our focus for this project. For a total number of 21,709 transactions the offline algorithm gives the miner a profit value of about 2,033 dollars while selecting 2,398 transactions to be added in the block. The online algorithm has the lowest output quality giving the miner almost 400 dollars less while selecting 2,095 transactions. Our advice model however improves the quality of the online setting by giving the miner more profit value (1,872 dollars) than the pure online algorithm while selecting 2,235 transactions.

Table 2 summarizes and compares the performance of our various algorithms in terms of speed. Note that speed was our motivation for implementing the online setting. The offline algorithm is almost 36 times slower than both the online greedy algorithm and the semi-online algorithms. This speed discrepancy can make a lot of difference when considering memory pools with millions of unconfirmed transactions. Hence, it's infeasibility for the online setting. Another interesting detail from the Table 2 is that the semi-online algorithm is slightly faster than the online greedy algorithm. This makes it a better choice in the online setting both in terms of output quality and performance speed.

TODO IMPROVE THE WRITING OF THE SECTION BELOW (DATA BASED ON 10 SIMULATIONS, 1000 IS RUNNING OVERNIGHT) As shown in the figure: 2 the Pure Greedy algorithm perform poorly compared with the sorted version (HEAP), once it only a accept all transactions as soon as it appears wasting space with possible low fees. An improvement can be noticed when we add a advice on the greedy approach. In this case the greedy only accept transactions which has density (value/size) greater or equal than a given advice, in this case the profit become close to the HEAP but yet lower than Optimal. The DST algorithm in other hands, consistent presents a slightly better results than the Optimal. It happens because it first pick the transactions which has high density class starting from the biggest size to the small ones.

In the figure 3 it is clear how the HEAP solution is surpassed by all the proposed solution. The SDT table as expected by definition has a cost of extraction close to the greedy once booth are linear. The time to mine a block linear increases once the size of the table growth.

In the figure 4 we take in consideration the total time to run 1000 simulations that consist in insert the transactions in the data structure of each Algorithm and select a set of transactions to form a block of a limited size.

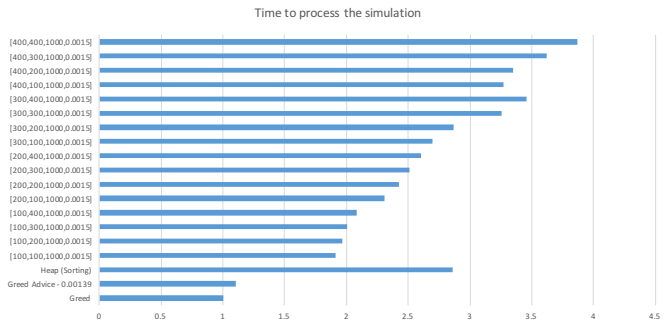

Fig. 4. Total time to simulate insertion of 29704 transactions in the Mempool and mine one block 1000 times.

6 Conclusions and Future Work

In our project we have implemented several algorithms for the transaction selection problem which we defined as a classic knapsack weighted problem. We have shown that the offline strategies are infeasible during transaction selection since the miners are in a race against other miners (and time) to see whose block is added next in the blockchain. This was also complimented by the fact that the dynamic programming algorithm could not run on my local system in polynomial time.

We postulated that the online setting was more practical and we implemented the online greedy algorithm which had a poor output quality when compared to the offline models. This led us to create an advice model that reveals the optimal profit value which was obtained from offline algorithms to the online model. This led to significant improvements both in output quality and speed. Hence, we concluded that the semi-online model with advice was the best algorithm for the online setting of the transaction selection problem. This is also in line with Bckenhauer et al [11] postulation that $O(\log n)$ bits is sufficient to make an online algorithm of weighted knapsack problems competitive. In the future we can work to increase the quality of our output value in the advice model. This can be done by obtaining a better optimal profit value from executing the dynamic programming algorithm on a more powerful system. We then use that optimal profit value as a parameter of our function α in the semi-online model algorithm.

We can also compare our transaction selection techniques to other techniques such as feature extraction and other machine learning techniques in the nearer future.

Acknowledgements

The last two authors acknowledge Natural Sciences and Engineering Research Council (NSERC) Canada for partial financial support for this research through Discovery Grants. The first author acknowledges the GETS funding from Faculty of Graduate Studies. The first and third authors acknowledge the International Graduate Student Scholarship from the Faculty of Graduate Studies, University of Manitoba.

References

[1] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] M. Iansiti and K. R. Lakhani, "The truth about blockchain," *Harvard Business Review*, vol. 95, no. 1, pp. 118–127, 2017.

[3] Utxo. [Online]. Available: https://www.investopedia.com/terms/u/utxo.asp

[4] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[5] Miner fees - bitcoin wiki. [Online]. Available: https://en.bitcoin.it/wiki/Miner_fees

[6] Bitcoin - open source p2p money. [Online]. Available: https://bitcoin.org/en/

[7] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_9

[8] A. Marchetti-Spaccamela and C. Vercellis, "Stochastic online knapsack problems," *Mathematical Programming*, vol. 68, no. 1, pp. 73–104, Jan 1995. [Online]. Available: https://doi.org/10.1007/BF01585758

[9] K. Iwama and S. Taketomi, "Removable online knapsack problems," in *Automata, Languages and Programming*, P. Widmayer, S. Eidenbenz, F. Triguero, R. Morales, R. Conejo, and M. Hennessy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 293–305.

[10] S. Dobrev, R. Krlovi, and D. Pardubsk, "Measuring the problem-relevant information in input," *RAIRO - Theoretical Informatics and Applications*, vol. 43, no. 3, p. 585613, 2009.

[11] H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith, "On the advice complexity of the knapsack problem," in *LATIN 2012: Theoretical Informatics*, D. Fernández-Baca, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 61–72.

[12] H.-J. Bckenhauer, D. Komm, R. Krlovi, and P. Rossmanith, "The online knapsack problem: Advice and randomization," *Theoretical Computer Science*, vol. 527, pp. 61 – 72, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397514000644

[13] B. B. F. Pontiveros, R. Norvill, and R. State, "Monitoring the transaction selection policy of bitcoin mining pools," *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018.

[14] H. Nicolas, "The economics of bitcoin transaction fees," *SSRN Electronic Journal*, 2014.

[15] B. Fiz, S. Hommes, and R. State, "Confirmation delay prediction of transactions in the bitcoin network," *Advances in Computer Science and Ubiquitous Computing Lecture Notes in Electrical Engineering*, p. 534539, 2017.

[16] "Bitcoin explorer." [Online]. Available: https://blockchair.com/bitcoin

Algorithm 4 Selecting transactions using SDT

Data: A set A of n transactions. A transaction $A[i]$ has a key, a size, and a density. The size and density of items is in the range (0,1] except for density of items in the last density class.

A table T of size $k1 \times k2$ containing dictionaries of transactions.

$T[i][j]$ is (a pointer to) the dictionary of transactions with size-class i and density-class j.

Result: A set B of selected transactions to be placed in the block

```
let cap = 1 // remaining capacity of the block
let max_no_transactions = ∞ // maximum number of transactions that we allow placing in the block (I guess there might be an upper bound for that)
let j = k_2 − 1 // points to the highest density class. let m = 0 // number of selected transactions
while m < max_no_transactions and terminate = false do
    // selecting the m'th transaction
    let si = cap × k_1
    let selected = false
    let i = si − 1 // start with the highest size-class
    // first, try to find any transaction with density class j among size classes guaranteed to fit
    while i > 0 and !selected do
        if T[i][j] is not empty then
            let transaction x be any member of T[i][j]
            add x to B
            remove x from T[i][j]
            cap = cap − x.size
            selected = true
        else
            i = i − 1
        end
    end
    // if failed to find a suitable transaction, try all transactions at size class s_i (they might fit or not)
    let t = 0
    while t < T[si][j].size and !selected do
        let x = T[si][j].next
        if x.size ≤ cap then
            add x to B
            remove x from T[si][j]
            cap = cap − x.size
            selected = true
        else
            t = t + 1
        end
    end
    // if failed to find a suitable transaction, that means there was no good transaction in the j'th density class; decrement j
    if !selected then
        let j = j − 1
        if j = −1 then
            terminate = true // no transaction in the entire table fit
        end
    else
        let m = m + 1
    end
```