Pixiu: Optimal Block Production Revenues on Cardano (Long Version)

Togzhan Barakbayeva, Soroush Farokhnia, Amir Kafshdar Goharshady, Markus Gufler, Sergei Novozhilov

Pixiu: Optimal Block Production Revenues on Cardano (Long Version)

Togzhan Barakbayeva*, Soroush Farokhnia*, Amir Goharshady*, Markus Gufler† and Sergei Novozhilov*
*Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
{tbarakbayeva, sfarokhnia, goharshady, snovozhilov}@cse.ust.hk
†Cardano Foundation, Zug, Switzerland
markus.gufler@cardanofoundation.org

***Abstract*—Cardano is a blockchain protocol based on proof-of-stake and an extended UTXO model which also supports arbitrary smart contracts. Its primary currency, Ada, is currently one of the global top ten cryptocurrencies with a market cap of more than 16 billion USD. In Cardano, new blocks are produced by stake pools. Any holder of Ada can delegate their stake to a pool. The underlying proof-of-stake consensus protocol is Ouroboros Praos, which divides time into a number of epochs and each epoch into a number of slots, each corresponding to one second. In each slot, leaders are randomly selected to produce and add new blocks to the blockchain, with their selection probability being proportional to their stake. Each block can contain a sequence of transactions and block production is rewarded in two ways: (i) transaction fees and (ii) monetary expansion. The producers have no control over (ii), but can optimize (i) by choosing which transactions to include in their blocks. Thus, they are incentivized to maximize the total transaction fees.**

In this work, we consider the natural optimization problem of forming a block with maximum transaction fees given a set of unmined Cardano transactions. We show that by exploiting the sparsity of interrelations between transactions, i.e. the small treedepth of dependency-conflict graphs, it is possible to obtain a polynomial-time algorithm that outputs optimal blocks. We implemented our algorithm in a free and open-source tool called Pixiu. Using Pixiu, we provide extensive experimental results over real-world transaction data on the Cardano blockchain demonstrating that our approach increases the block producers' revenue by almost 1,357.82 USD/day = 495,604.3 USD/year.

***Index Terms*—Cardano, Ada, Transaction Fees, Optimization**

1. Introduction

Mining. Every blockchain protocol requires a process to extend the chain by adding new blocks. This process is often called *mining* in proof-of-work blockchains such as Bitcoin [1] and entails finding a solution to a proof-of-work puzzle which is often based on inverting hash functions. While proof-of-stake blockchains rely on random leader election [2]–[11]. They and other alternative consensus mechanisms such as proof-of-space or hybrid mining [12], [13] do not require the expensive step of solving a hash puzzle, they nevertheless need rules for extending the chain. The nodes that take part in chain extension are known by various names such as validators, farmers or producers. For simplicity, in this work, we simply use the words *miner* and *producer* to refer to any node on the network who tries to add a new block to the blockchain according to the underlying consensus protocol.

Mining Steps. A miner has to first form a block by choosing a set of unmined transactions. Following Bitcoin [1], most blockchain protocols have a maximum size limit on the blocks, thus the miner has to strategically pick the transactions that are included in her block. The miner then proposes the block by publicly announcing it. The proposed block may or may not be adopted by the network, based on its rules of consensus. For example, in a proof-of-work cryptocurrency, only blocks that contain a valid solution to the hash puzzle may be accepted by the network. In this work, we focus on the Cardano blockchain, which employs proof-of-stake. However, our main emphasis is on the first step, i.e. how a block is formed, and our results are quite independent of the consensus protocol.

Cardano [14]. Cardano is an open and decentralized blockchain platform that supports many cryptocurrencies and tokens. Its main currency, Ada, is currently one of the top 10 cryptocurrencies in terms of market cap and has a value of more than 16 billion USD [15]. Similar to Bitcoin, Cardano follows the (extended) UTXO model [16], in which every transaction has a set of inputs and outputs. Each input to a transaction should be an output of a previous transaction and no output may be spent (used as input) twice. It has two major advantages over Bitcoin: (i) support for arbitrarily complex smart contracts, i.e. transactions can invoke executions of programs in an expressive programming language, and (ii) a proof-of-stake consensus protocol, namely Ouroboros Praos [17], which avoids the costly mining process of Bitcoin's proof-of-work. In this work, we consider a simplified model of a Cardano transaction that precisely captures those aspects which are important to our problem, while ignoring other details which do not affect the problem at hand. For a more thorough treatment, see [14]

or [18].

Proof-of-Stake [2], [3], [19]. In proof-of-stake protocols, a miner is randomly chosen to add the next block. Each miner's probability of being chosen is proportional to her stake in the currency, i.e. the number of coins she holds. Specifically, in Cardano's implementation of Ouroboros Praos [17], time is divided into epochs, each consisting of 432,000 slots. Each slot corresponds to one second. Thus, each epoch lasts for five days. In each slot, some miners/producers are randomly selected to propose blocks of transactions [20].

Stake Pools and Delegation [21]. Any user who holds Ada can take part in mining (block production) on Cardano. Users can also delegate their mining rights (stakes) to others, leading to stake pools which, as the name suggests, pool together stakes from many different users. The pool operator can then mine on behalf of all of its users and the probability that the pool is selected in each slot is proportional to the total stake of the members. The vast majority of Cardano blocks are produced by stake pools rather than individual stake holders.

Rewards. Producing blocks is a costly process. This is of course evident in proof-of-work blockchains, in which the miners have to solve hash puzzles requiring vast computational resources and electricity usage. Thus, Bitcoin rewards miners for every new block that they successfully add to the blockchain and also pays them transaction fees [1]. Even in proof-of-stake blockchains, there are costs associated with block production, e.g. the miner has to keep track of the whole history of the chain, constantly listen for new transactions, hold stake or convince others to delegate stakes to her, form valid blocks of transactions and announce/propose them on time. Therefore, it is necessary to incentivize block production by rewarding the miners. In Cardano, the block producers are rewarded in two ways [22]:

(i) *Transaction Fees:* Every transaction contains a fixed fee. After each epoch, the fees of all transactions mined in that epoch are divided among the block producers.
(ii) *Monetary Expansion:* For each epoch, a fixed percentage of the remaining reserve of Ada is paid to all block producers of the epoch.

In general, producers have no control over (ii). However, they can choose which transactions to include in their blocks. Thus, they have a degree of control over (i) and are collectively incentivized to maximize the total amount of transaction fees in their epoch and thus in their block.

Our Problem. In this work, we focus on solving the problem of forming an optimal block on the Cardano blockchain. Specifically, given a set of unmined transactions, i.e. transactions that are not yet added to the blockchain, our goal is to create a valid Cardano block with the maximum possible amount of transaction fees, thus maximizing the total revenue of block producers.

Related Work on Bitcoin. A similar problem has been considered in the context of Bitcoin [23], where it was shown to be NP-hard and hard-to-approximate unless P=NP. Since our setting is more general than Bitcoin, as Cardano transactions support smart contracts and many underlying tokens/currencies, our problem is also NP-hard and hard-to-approximate. [23] proposes an approach based on path decompositions to find optimal Bitcoin blocks. Unfortunately, this approach is not efficient enough for Cardano as it often takes several minutes to find the required decompositions and optimal blocks, whereas the time between any two slots/blocks in Cardano is only one second. Thus, while the problem of optimal block production is well-motivated and previously considered in the blockchain literature, there are no approaches that are efficient enough to be applicable to proof-of-stake programmable blockchains such as Cardano in which new blocks are added in short timeframes. In this work, we present the first such approach.

Our Contribution. We consider the problem of forming an optimal block of transactions, i.e. a block with maximum total transaction fee, in Cardano. Since the problem is NP-hard, we exploit the sparsity of interrelations between real-world Cardano transactions to obtain an algorithm that has polynomial runtime for real-world instances of the problem. Formally, we consider a graph of interrelations between transactions and show that when this graph has bounded treedepth, i.e. when it is sparse and resembles a shallow tree, there is a polynomial-time algorithm for finding the optimal block to mine. We then experimentally show that the small-treedepth assumption holds for real-world Cardano instances.

We implemented our algorithm as a free and open-source tool called Pixiu. Pixiu is a Chinese mythical creature, resembling a panther, that finds and eats gold and other jewelry and brings them to its master. Our algorithm does the same for Cardano block producers.

Figure 1: A Pixiu generated by DALL · E.

With the help of the Cardano Foundation, we performed a 50-day-long experiment on the Cardano blockchain in which we ran our algorithm on the same sets of transactions as those available to real-world Cardano block producers and compared the blocks produced by our approach with the ones actually added on Cardano. Our approach increased the transaction fee revenues by 1,357.82 USD/day = 495,604.3 USD/year. Thus, the benefits of producing optimal blocks

that maximize the transaction fees are highly significant in practice.

2. Preliminaries

2.1. Cardano's Transaction Model and Optimal Block Production

Most modern blockchains either follow the Unspent Transaction Output (UTXO) model, as in Bitcoin [1], or the account model, as in Ethereum [24]. To take advantage of the benefits of both models, Cardano proposes the Extended UTXO (EUTXO) model that allows having more expressive programs than simple scripts without adopting the account model. In UTXO, a transaction has a set of inputs and outputs. An input points to an output of a prior transaction that provides funds to be spent by this transaction. Moreover, every output can be used by at most one input. EUTXO follows the same policies except that there is one more output field called "datum" that carries contract-specific data. In addition, EUTXO allows the inclusion of arbitrary logic as scripts and the use of the data fields to decide if the transaction output can be spent. See [16], [18] for a more detailed treatment. Unlike Ethereum, in which gas usage directly affects fees and is subject to optimization [25]–[28], in the UTXO model each transaction carries a fixed and known fee which does not depend on its gas usage.

Block Constraints. In Cardano's EUTXO model, a block contains a sequence of valid transactions. The transactions in each block must satisfy the following requirements, otherwise the block is considered invalid and will be discarded by all nodes of the network.

- *Transaction Dependencies.* If a transaction t_2 uses an output of another transaction t_1 as one of its inputs, then t_2 depends on t_1. If the block producer decides to include t_2 in her new block, then she must also add t_1 in the same block and before t_2. See Figure 2 as an example.

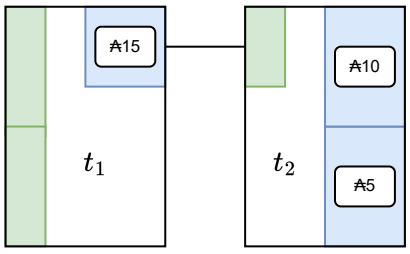

Figure 2: Alice pays 15 Ada to Bob in t_1. In t_2 Bob uses the same funds to pay 10 Ada to Carol and 5 Ada back to himself. The transaction t_2 has an input that is using an output of t_1. Thus, t_1 is a dependency of t_2.

- *Transaction Conflicts.* If two transactions t_2 and t_3 both spend the same output of a transaction t_1, since every output may be spent at most once, only one of the conflicting transactions may be selected and included in the new block. Thus, the block producer has a choice to include t_2, t_3 or neither in the new block, but she cannot include both. See Figure 3 as an example.
- *Block Size Limit.* Cardano imposes a block size limit, i.e. the total size of all transactions included in a new block must not exceed 90112 bytes = 88 kilobytes. This is to ensure that the blocks are small enough to be efficiently propagated within the network with minimal latency. The block size limit may change in each epoch, but in practice it has been consistently kept at 88 kilobytes.

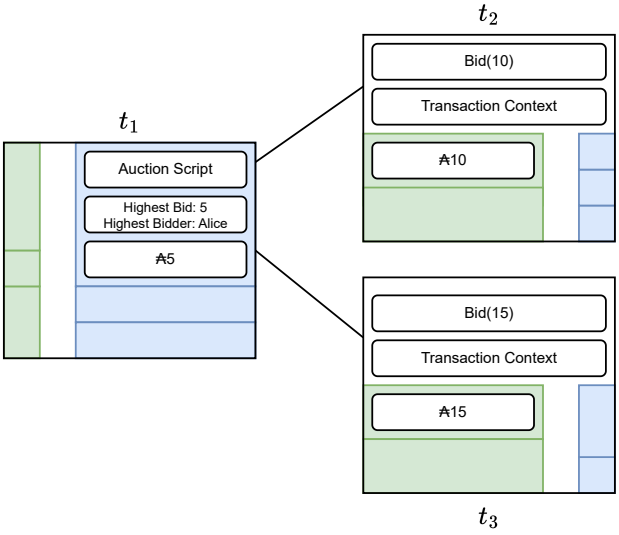

Figure 3: In an auction contract, Alice has bid 5 Ada in t_1. Both Bob and Carol are trying to bid right after Alice, hence using t_1's datum output as input to their transactions t_2 and t_3. Since every output can be used only once, t_2 and t_3 are in conflict.

Based on the discussion above, we can now formally define our problem.

Optimal Block Production. Given a block size limit $k \in \mathbb{N}$, a finite set TX of n unmined Cardano transactions in which every transaction $t \in \mathsf{TX}$ has a set of inputs, a set of outputs, a fee $\phi(t) \in [0, \infty)$ and a size $\sigma(t) \in \mathbb{N}$, find a subset $\mathsf{TX}^* \subseteq \mathsf{TX}$ of transactions such that:

- TX^* satisfies all the dependency and conflict requirements as above;
- $\sum_{t \in \mathsf{TX}^*} \sigma(t) \leq k$, i.e. all the chosen transactions fit into the block size limit; and
- $\sum_{t \in \mathsf{TX}^*} \phi(t)$ is maximized, i.e. the block consisting of the transactions in TX^* yields the maximum possible total transaction fee.

In practice, we always have $k = 90112$.

Dependency-Conflict Graphs (DCGs) [23]. Given an instance of the optimal block production problem above, we

create a graph $G = (\mathsf{TX}, E_C \cup E_D)$ in which there is a vertex corresponding to every transaction in TX and an undirected edge $\{t_1, t_2\} \in E_C$ whenever the transactions t_1 and t_2 are in conflict, as well as a directed edge $(t_1, t_2) \in E_D$ when transaction t_2 depends on transaction t_1. See Figure 4.

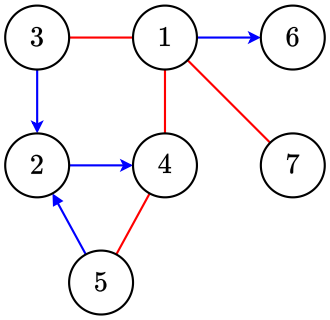

Figure 4: An example dependency-conflict graph. Dependency edges are shown in blue and conflict edges in red.

The work [23] considered DCGs in the context of Bitcoin mining. It showed that if the DCGs are sparse and tree-like or path-like, then the optimal block production problem for Bitcoin is efficiently solvable. The approach in [23] depends on the concepts of pathwidth [29] and treewidth [30], which are often used to design efficient parameterized graph algorithms [31]–[44]. Intuitively, it first computes a decomposition of the DCG into a path/tree and then uses a dynamic programming algorithm to obtain the optimal block. Unfortunately, such an approach is too slow and not applicable to our use-case in Cardano. This is because in Cardano, a new block has to be produced every second, whereas computing the decomposition notion used in [23] takes minutes. This was not a problem in Bitcoin, where a new block is added every 10 minutes, but is not scalable enough for Cardano. Therefore, in this work, we consider a stronger notion of decomposition, namely the treedepth decomposition, and provide an algorithm based on treedepth to solve the optimal block production problem in Cardano. As we will see in Section 4, our algorithm is highly scalable in practice and produces optimal blocks in less than a second, hence enabling its application in Cardano.

2.2. Treedepth

Treedepth Decompositions [45]–[47]. For a graph $G = (V, E)$, a *treedepth decomposition* is a rooted tree $T = (V, E_T)$ on the same set of vertices as G that satisfies the following requirement:

- For every undirected edge $\{u, v\} \in E$ or directed edge $(u, v) \in E$ of the original graph, either u is an ancestor of v in T or v is an ancestor of u in T.

We say that a treedepth decomposition T is optimal if it has the smallest possible depth among all decompositions of G. This smallest depth is called the *treedepth* of G. Intuitively, treedepth is a measure of graph sparsity that captures how much a graph resembles a shallow tree. Throughout this work, we always consider the treedepth d of a dependency-conflict graph G. See Figure 5.

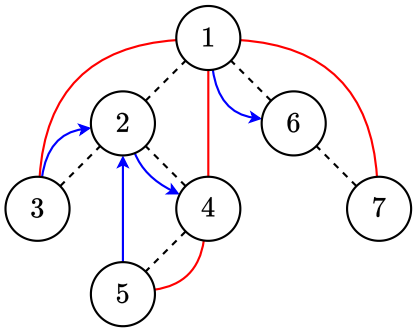

Figure 5: A treedepth decomposition of the DCG graph of Figure 4. The edges of the decomposition are dashed. Every edge of the original graph (shown in red and blue) goes between a vertex and one of its ancestors. The vertices are numbered in pre-order. This decomposition has a depth of 3, since the path from the root 1 to the farthest leaf 5 has three edges.

Computing Treedepth. For any small fixed d, there is an algorithm that decides whether an input graph has treedepth d in linear time and if so, outputs an optimal treedepth decomposition [48]. There are also well-optimized tools and libraries for computing treedepth decompositions [49]. As we will see in Section 4, DCGs in Cardano have small treedepth. Thus, in the remainder of this paper we assume, without loss of generality, that we have access to an optimal treedepth decomposition of every DCG. In practice, we use [49] to find such decompositions.

Vertex Numbering. Recall that the vertices in our DCGs are unmined Cardano transactions. We number the vertices by a pre-order (left-to-right) traversal of our treedepth decomposition. See Figure 5 as an example. This is also without loss of generality, but allows us to present our algorithm more concisely.

Ancestor Sets. Suppose that our treedepth decomposition T is rooted at vertex 1. For every vertex $v \in V$, we denote by A_v the set of ancestors of v, i.e. the set of vertices that are on the path from the root 1 to v in T. For two vertices $u, v \in V$, we define $A_{u,v} := A_u \cap A_v$ as the set of their common ancestors, i.e. vertices that are ancestors of both u and v. Since we numbered the vertices in pre-order, we have $A_{i,i-1} = A_i \setminus \{i\}$ for every vertex i.

3. Our Algorithm

In this section, we present our algorithm for finding an optimal Cardano block that maximizes the total transaction fee revenue of the block producers. Our algorithm is a dynamic programming approach based on the treedepth decomposition of the conflict-dependency graph of our unmined transactions.

Input. Suppose that we are given an optimal block production instance, consisting of the block size limit $k \in \mathbb{N}$ and a set TX of n unmined Cardano transactions as input. Each transaction $t \in \mathsf{TX}$ has a fee of $\phi(t)$ and a size of $\sigma(t)$. Additionally, we have the dependency-conflict graph $G = (\mathsf{TX}, E_C \cup E_D)$ and a treedepth decomposition $T = (\mathsf{TX}, E_T)$ of G with depth d. Our goal is to solve the optimal block production problem.

Canonical Subgraphs. We define n canonical subgraphs of our DCG G. The i-th canonical subgraph G_i consists of the first i transactions, as well as any conflicts and dependencies between them. Formally, we let

$$V_i = \{1, 2, \ldots, i\}$$

and

$$G_i = G[V_i].$$

Recall that the transactions are numbered by a pre-order traversal of T. We consider subproblems on each $G_i = (V_i, E_i)$ and show how to combine the results on these subproblems to find an optimal block for the entire DCG G. See Figure 6.

Dynamic Programming Table. For every $1 \leq i \leq n$, every partial capacity $0 \leq c \leq k$ and every subset $S \subseteq A_i$, we define a subproblem and a dynamic programming variable as follows:

$\mathrm{dp}[i, S, c] :=$ The maximum total fees $\Sigma_{t \in \mathsf{TX}^*} \phi(t)$
of a set $\mathsf{TX}^* \subseteq V_i$ of transactions in G_i
such that $\mathsf{TX}^* \cap A_i = S$
and $\Sigma_{t \in \mathsf{TX}^*} \sigma(t) \leq c$
and TX^* satisfies dependencies/conflicts in E_i.

Intuitively, we are considering subproblems in which we have only the first i vertices/transactions and their dependencies/conflicts, but we also consider the case where our capacity is $c \leq k$, i.e. part of the block is already filled and we only have c bytes of free space remaining. Finally, the set S tells us exactly which ancestors of transaction i should be taken into the solution.

Computing Values for G_1. The subgraph G_1 consists only of the root vertex 1 and has no edges. Thus, we have

$$\mathrm{dp}[1, \emptyset, c] = 0,$$

and

$$\mathrm{dp}[1, \{1\}, c] = \begin{cases} \phi(1) & \sigma(1) \leq c \\ -\infty & \sigma(1) > c \end{cases}.$$

We use $-\infty$ to show an impossible situation, i.e. when no possible set TX^* satisfying the requirements can be found.

Computing Values for Other G_i's. Suppose $i > 1$ and we intend to compute $\mathrm{dp}[i, S, c]$. Moreover, assume that the $\mathrm{dp}[j, \cdot, \cdot]$ values are already computed for all $j < i$.

- We first check if S violates any of the dependency and conflict requirements in the set A_i, i.e. between the ancestors of vertex i. Specifically, for any two vertices $u, v \in A_i$, if $\{u, v\} \in E_C$ and also $u, v \in S$, then they are in conflict but both taken in S and so we have to set $\mathrm{dp}[i, S, c] = -\infty$ since the requirements are impossible to satisfy. Similarly, if $(u, v) \in E_D$ and $v \in S$ but $u \notin S$, then the dependency requirement is violated and we set $\mathrm{dp}[i, S, c] = -\infty$. For example, in Figure 6 we set $\mathrm{dp}[4, \{4\}, 10]$ to $-\infty$ since 4 is included and depends on 2, which is not included. Similarly, $\mathrm{dp}[3, \{1, 2, 3\}, 10] = -\infty$ since 1 and 3 are in conflict.
- We then check if all the transactions in S can fit into c bytes, i.e. whether $\sum_{t \in S} \sigma(t) \leq c$. If not, we set $\mathrm{dp}[i, S, c] = -\infty$.
- Let $S' \subseteq A_{i-1}$ be a subset of ancestors of vertex $i-1$. We say that S' is *compatible* with S and write $S' \leftrightarrows S$ if $\forall u \in A_{i-1,i}$ we have $u \in S \Leftrightarrow u \in S'$. In other words, compatible subsets make the same decisions about the common ancestors of i and $i-1$. If all the checks above pass, then we consider two cases:

  (1) If $i \notin S$, then we know that our solution TX^* cannot contain the transaction i. Thus, all transactions in the solution are already present in G_{i-1}. Therefore, we set

  $$\mathrm{dp}[i, S, c] = \max_{S' \leftrightarrows S} \mathrm{dp}[i-1, S', c].$$

  (2) If $i \in S$, then we must put the transaction i into our solution TX^*. This reduces the available space to $c - \sigma(i)$ but also gives us a fee of $\phi(i)$. We should then fill out our block using the previous $i-1$ transactions. Thus, we have

  $$\mathrm{dp}[i, S, c] = \phi(i) + \max_{S' \leftrightarrows S} \mathrm{dp}[i-1, S', c - \sigma(i)].$$

Final Solution. Finally, we know that $G = G_n$ by definition. In our optimal solution, we might be taking any subset S of the ancestors of vertex n. Thus, our algorithm outputs

$$\max_{S \subseteq A_n} \mathrm{dp}[n, S, k]$$

as the maximum possible amount of transaction fees that can be obtained from a subset of TX that fits into a block of size k. As is standard in dynamic programming approaches, the optimal subset TX^* of transactions can be recovered by retracing the steps of our algorithm and finding out which choices led to the maximum values.

Theorem 1. *Given a block size limit k, a set TX of n unmined Cardano transactions with dependency-conflict graph G and a treedepth decomposition T of G with depth d, our algorithm solves the Optimal Block Production problem in time $O(n \cdot k \cdot 2^d \cdot d^2)$.*

Proof. Correctness was argued in the discussion above. Note that we always find a valid solution TX^* since the dependency/conflict requirements between any vertex i and its ancestors are enforced when we are computing $\mathrm{dp}[i, \cdot, \cdot]$. To bound the runtime, note that we define a total of $n \cdot k \cdot 2^d$ dynamic programming variables. For each of them, we check dependency and conflict requirements between elements of

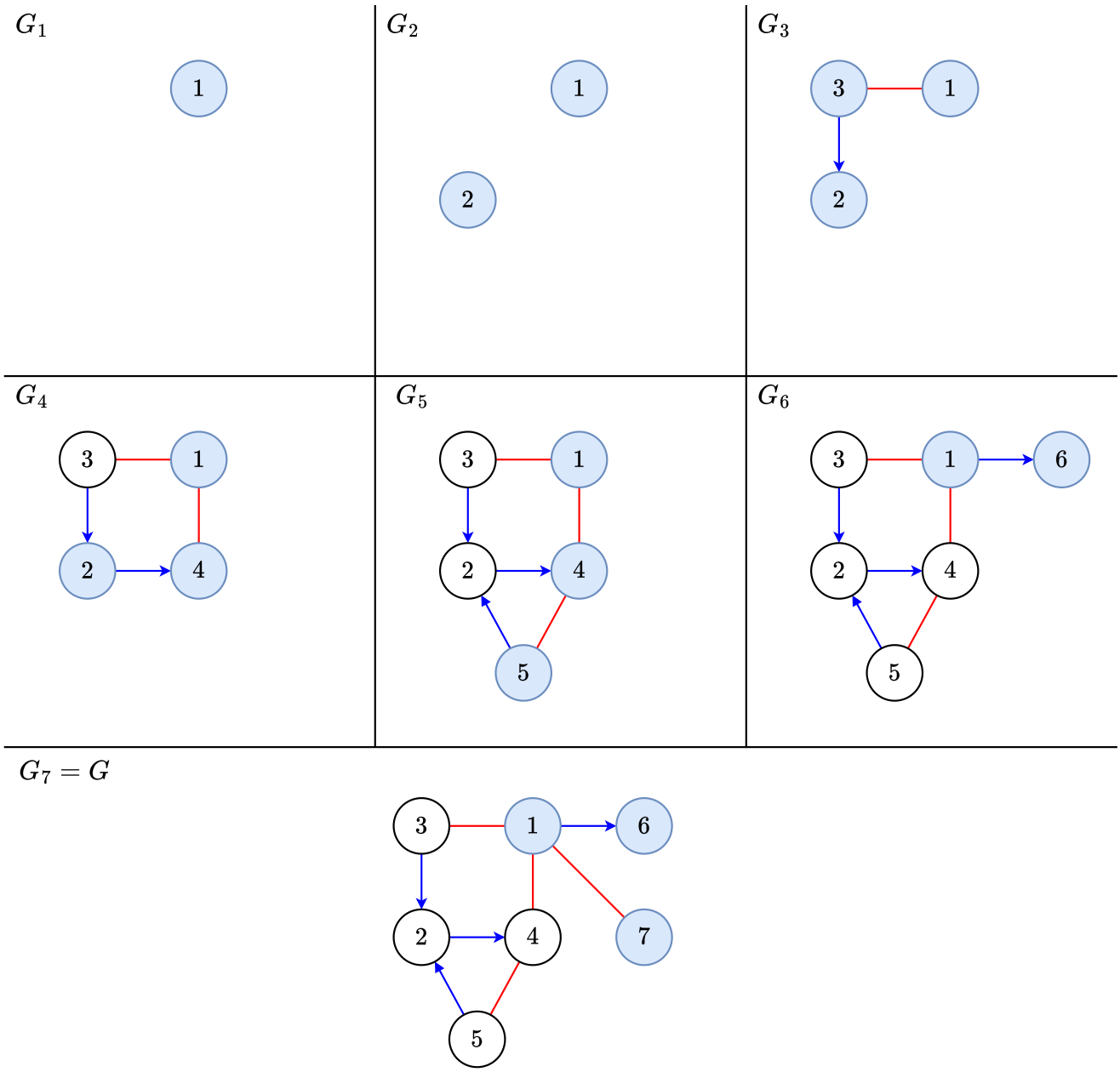

Figure 6: Canonical subgraphs of the graph G of Figure 4 based on the treedepth decomposition of Figure 5. In each G_i, the set A_i of ancestors of vertex i is shown in blue.

a set A_i which has a size of at most d since they are all ancestors of a single vertex i in T. Therefore, the total runtime for the initial checks is $O(n \cdot k \cdot 2^d \cdot d^2)$. Now consider the sums computed in parts (1) and (2) above. Consider any fixed $S' \subseteq A_{i-1}$. Based on the way we numbered our vertices in pre-order, every ancestor of vertex i, except i itself, is also an ancestor of vertex $i-1$. Formally, $A_{i,i-1} = A_i \setminus \{i\}$. Thus, each S' may contribute to the sums for at most two different compatible $S \subseteq A_i$ sets. Therefore, the total runtime of all sums in (1) and (2) is $O(n \cdot k \cdot 2^d)$. The runtime is polynomial in n and k when d is a constant. □

Parallelization. We remark that the computation of $\mathrm{dp}[i, \cdot, \cdot]$ in our algorithm only depends on $\mathrm{dp}[i-1, \cdot, \cdot]$ values. Thus, for every i, we can perfectly parallelize the computation of all $\mathrm{dp}[i, \cdot, \cdot]$ entries. In other words, if we have $p < k \cdot 2^d$ parallel cores, our runtime will be $O\left(\frac{n \cdot k \cdot 2^d \cdot d^2}{p}\right)$. Therefore, one can reduce the runtime of our algorithm arbitrarily by simply adding more computational power. In Section 4 we do not use parallelization but our runtimes are still much

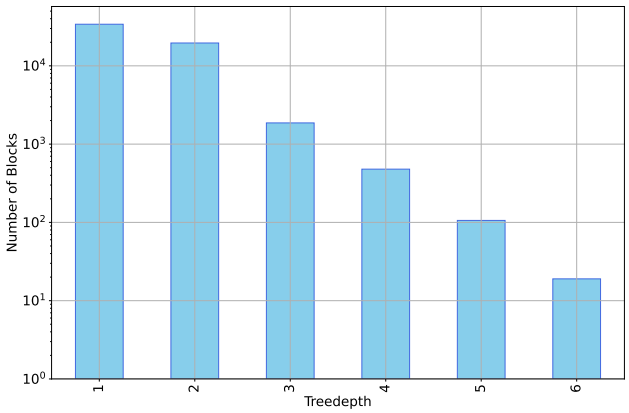

Figure 7: Histogram of treedepths of DCGs in real-world Cardano benchmarks. The y-axis is in logarithmic scale.

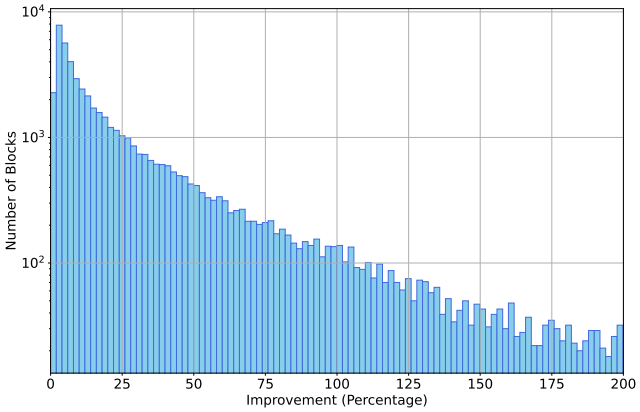

Figure 8: Histogram of the transaction fee improvements obtained over each block (in percentages). The y-axis is in logarithmic scale.

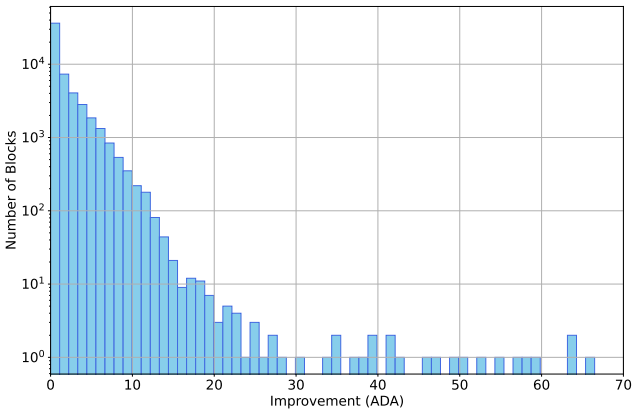

Figure 9: Histogram of the transaction fee improvements obtained over each block (in Ada). The y-axis is in logarithmic scale.

less than one second, enabling the direct application of our approach to Cardano.

4. Implementation and Experimental Results

We implemented our algorithm in C++ as a tool called Pixiu. Pixiu is free and open-source software donated to the public domain. We used FlowCutter [49] to find treedepth decompositions.

Machine. All experimental results were obtained on an Intel Xeon Gold 5115 CPU (2.40GHz, 16 cores) running Ubuntu 22.04 and 64 GB of RAM. We did not use parallelization in the runtimes reported below.

Benchmarks. With the help of the Cardano Foundation, we gathered the sets of unmined transactions before each of the blocks number 10,044,250 to 10,255,796 of the Cardano blockchain, corresponding to the timeframe of 2024-03-12 00:00:37 UTC to 2024-04-30 23:59:41 UTC (50 days). For each of the 211,547 blocks mined in this period, we ran our algorithm to obtain an optimal selection of transactions and compared the resulting total transaction fees with the fee revenue obtained by the block producers on Cardano.

Treedepth. We observed that the vast majority of benchmark DCGs had small treedepth. Figure 7 shows a histogram of the obtained treedepths. The average treedepth was 1.45. Thus, our algorithm is applicable to real-world Cardano block production and runs in polynomial time $O(n \cdot k)$ on these instances.

Increases in Revenue. Limiting the runtime to 1s, our algorithm improved the total transaction fees in 56,053 of the 211,547 blocks considered in our experiment. This suggests that many of the blocks produced on the Cardano blockchain were already optimal. This is not surprising since when the transaction load in the network is low, the miners can often include all the available transactions in their block, which would of course be optimal. Our algorithm's advantage is most pronounced when the number of available unmined transactions is much more than the capacity of a block. Over these 56,053 blocks, the average per-block improvement was 55.68 percent, corresponding to 1.55 Ada = 1.21 USD, whereas the maximum improvement was 66.48 Ada = 51.85 USD. We used the exchange rate 1 Ada = 0.78 USD. The overall improvement over the period of the experiment was 87,040.20 Ada = 67,891.35 USD. Thus, our algorithm obtains transaction fee revenue increases of 1,357.82 USD/day = 495,604.3 USD/year. Therefore, the Cardano miners would benefit immensely from applying our algorithm and ensuring that they will always produce optimal blocks that maximize their transaction fee revenue. Figures 8–10 show the histogram of obtained improvements in percentage, Ada and USD. Our average runtime was 0.31s per block.

5. Conclusion

In this work, we presented a novel algorithm that exploits the sparsity of interrelations between Cardano trans-

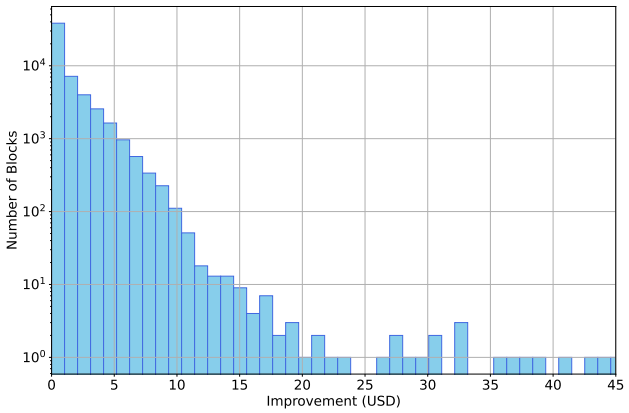

Figure 10: Histogram of the transaction fee improvements obtained over each block (in USD). The y-axis is in logarithmic scale.

actions to help block producers form a block that earns them the maximum possible revenue in transaction fees. We implemented our approach in a tool called Pixiu and performed a 50-day-long experiment over real-world Cardano blockchain data. The results showed that our approach significantly increases the revenues of Cardano block production by almost 500,000 USD/year. Thus, our approach has a clear, significant and measurable economic impact and the block producers would benefit immensely from adopting our tool.

Acknowledgments and Notes

We are grateful to the Cardano Foundation for their help in this project. The research was partially supported by the Hong Kong Research Grants Council ECS Grant Number 26208122. T. Barakbayeva was supported by the Hong Kong PhD Fellowship Scheme. Authors are ordered alphabetically.

References

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

[2] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," 2012.

[3] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017, pp. 357–388.

[4] V. Abidha, T. Barakbayeva, Z. Cai, and A. Goharshady, "Gas-efficient decentralized random beacons," in *ICBC*, 2024.

[5] T. Barakbayeva, Z. Cai, and A. Goharshady, "SRNG: An efficient decentralized approach for secret random number generation," in *ICBC*, 2024.

[6] J. Ballweg, Z. Cai, and A. Goharshady, "PureLottery: Fair leader election without decentralized random number generation," in *Blockchain*, 2023.

[7] P. Fatemi and A. Goharshady, "Secure and decentralized generation of secret random numbers on the blockchain," in *BCCA*, 2023.

[8] Z. Cai and A. K. Goharshady, "Game-theoretic randomness for proof-of-stake," in *MARBLE*, ser. Lecture Notes in Operations Research, 2023, pp. 28–47.

[9] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, "Hydrand: Efficient continuous distributed randomness," in *SP*, 2020, pp. 73–89.

[10] Z. Cai and A. K. Goharshady, "Trustless and bias-resistant game-theoretic distributed randomness," in *ICBC*, 2023, pp. 1–3.

[11] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *IEEE ICBC*, 2019, pp. 403–412.

[12] S. Park, A. Kwon, G. Fuchsbauer, P. Gaži, J. Alwen, and K. Pietrzak, "Spacemint: A cryptocurrency based on proofs of space," in *FC*, 2018, pp. 480–499.

[13] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Hybrid mining: exploiting blockchain's computational power for distributed problem solving," in *SAC*, 2019, pp. 374–381.

[14] C. Hoskinson, "Why we are building Cardano," 2017.

[15] CoinMarketCap, "Cryptocurrency prices, charts and market capitalizations," 2024. [Online]. Available: https://coinmarketcap.com/

[16] Cardano Foundation, "EUTXO handbook," 2017. [Online]. Available: https://docs.cardano.org/learn/eutxo-explainer

[17] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT*, 2018, pp. 66–98.

[18] L. Brünjes and M. J. Gabbay, "UTxO vs account-based smart contract blockchain programming paradigms," in *ISoLA*, 2020, pp. 73–88.

[19] Cardano Documentation, "Proof of stake," 2024. [Online]. Available: https://docs.cardano.org/new-to-cardano/proof-of-stake

[20] ——, "Slots and epochs," 2024. [Online]. Available: https://docs.cardano.org/learn/cardano-node/#howdoesitwork

[21] ——, "Stake pools," 2024. [Online]. Available: https://docs.cardano.org/learn/stake-pools

[22] ——, "Pledging and rewards," 2024. [Online]. Available: https://docs.cardano.org/learn/pledging-rewards

[23] M. Alambardar Meybodi, A. K. Goharshady, M. R. Hooshmandasl, and A. Shakiba, "Optimal mining: Maximizing Bitcoin miners' revenues from transaction fees," in *IEEE Blockchain*, 2022, pp. 266–273.

[24] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[25] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "*Don't run on fumes* - parametric gas bounds for smart contracts," *J. Syst. Softw.*, vol. 176, p. 110923, 2021.

[26] Z. Cai, S. Farokhnia, A. K. Goharshady, and S. Hitarth, "Asparagus: Automated synthesis of parametric gas upper-bounds for smart contracts," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA2, pp. 882–911, 2023.

[27] S. Farokhnia and A. K. Goharshady, "Reducing the gas usage of ethereum smart contracts without a sidechain," in *ICBC*, 2023, pp. 1–3.

[28] ——, "Alleviating high gas costs by secure and trustless off-chain execution of smart contracts," in *SAC*, 2023, pp. 258–261.

[29] N. Robertson and P. D. Seymour, "Graph minors. i. excluding a forest," *Journal of Combinatorial Theory, Series B*, vol. 35, no. 1, pp. 39–61, 1983.

[30] ——, "Graph minors. iii. planar tree-width," *Journal of Combinatorial Theory, Series B*, vol. 36, no. 1, pp. 49–64, 1984.

[31] K. Chatterjee, A. K. Goharshady, and E. K. Goharshady, "The treewidth of smart contracts," in *SAC*, 2019, pp. 400–408.

[32] H. L. Bodlaender, "Dynamic programming on graphs with bounded treewidth," in *ICALP*, 1988, pp. 105–118.

[33] G. Conrado, A. Goharshady, P. Hudec, P. Li, and H. Motwani, "Faster treewidth-based approximations for Wiener index," in *SEA*, 2024.

[34] G. K. Conrado, A. K. Goharshady, and C. K. Lam, "The bounded pathwidth of control-flow graphs," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA2, pp. 292–317, 2023.

[35] G. K. Conrado, A. K. Goharshady, K. Kochekov, Y. C. Tsai, and A. K. Zaher, "Exploiting the sparseness of control-flow and call graphs for efficient and on-demand algebraic program analysis," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA2, pp. 1993–2022, 2023.

[36] A. K. Goharshady and A. K. Zaher, "Efficient interprocedural data-flow analysis using treedepth and treewidth," in *VMCAI*, vol. 13881, 2023, pp. 177–202.

[37] N. Robertson and P. D. Seymour, "Graph minors. ii. algorithmic aspects of tree-width," *Journal of algorithms*, vol. 7, no. 3, pp. 309–322, 1986.

[38] F. V. Fomin, D. Lokshtanov, S. Saurabh, M. Pilipczuk, and M. Wrochna, "Fully polynomial-time parameterized computations for graphs and matrices of low treewidth," *TALG*, vol. 14, no. 3, pp. 1–45, 2018.

[39] R. Niedermeier, "Invitation to fixed-parameter algorithms," *Habilitationschrift, University of Tübingen*, vol. 19, 2002.

[40] A. Ahmadi, M. Daliri, A. K. Goharshady, and A. Pavlogiannis, "Efficient approximations for cache-conscious data placement," in *PLDI*, 2022, pp. 857–871.

[41] A. K. Goharshady and F. Mohammadi, "An efficient algorithm for computing network reliability in small treewidth," *Reliab. Eng. Syst. Saf.*, vol. 193, p. 106665, 2020.

[42] A. Asadi, K. Chatterjee, A. K. Goharshady, K. Mohammadi, and A. Pavlogiannis, "Faster algorithms for quantitative analysis of mcs and mdps with small treewidth," in *ATVA*, vol. 12302, 2020, pp. 253–270.

[43] K. Chatterjee, A. K. Goharshady, R. Ibsen-Jensen, and A. Pavlogiannis, "Optimal and perfectly parallel algorithms for on-demand data-flow analysis," in *ESOP*, vol. 12075, 2020, pp. 112–140.

[44] K. Chatterjee, A. K. Goharshady, N. Okati, and A. Pavlogiannis, "Efficient parameterized algorithms for data packing," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 53:1–53:28, 2019.

[45] J. Nešetřil, P. Ossona de Mendez, J. Nešetřil, and P. O. de Mendez, "Bounded height trees and tree-depth," *Sparsity: Graphs, Structures, and Algorithms*, pp. 115–144, 2012.

[46] Y. Iwata, T. Ogasawara, and N. Ohsaka, "On the power of tree-depth for fully polynomial fpt algorithms," *arXiv preprint arXiv:1710.04376*, 2017.

[47] J. Nešetřil and P. Ossona de Mendez, "On low tree-depth decompositions," *Graphs and combinatorics*, vol. 31, no. 6, pp. 1941–1963, 2015.

[48] F. Reidl, P. Rossmanith, F. S. Villaamil, and S. Sikdar, "A faster parameterized algorithm for treedepth," in *ICALP*, 2014, pp. 931–942.

[49] B. Strasser, "PACE solver description: Tree depth with FlowCutter," in *IPEC*, 2020, pp. 32:1–32:4.