Fairness-based Packing of Industrial IoT Data in Permissioned Blockchains

Shan Jiang, Jiannong Cao, Hanqing Wu, Yanni Yang

***Abstract*—In recent years, blockchain has been broadly applied to industrial Internet of things (IIoT) due to its features of decentralization, transparency, and immutability. In existing permissioned blockchain based IIoT solutions, transactions submitted by IIoT devices are arbitrarily packed into blocks without considering their waiting times. Hence, there will be a high deviation of the transaction response times, which is known as the lack of fairness. Unfair permissioned blockchain decreases the quality of experience from the perspective of the IIoT devices. Moreover, some transactions can get timeouts if not responded for a long time. In this paper, we propose FAIR-PACK, the first fairness-based transaction packing algorithm for permissioned blockchain empowered IIoT systems. First, we gain the insight that fairness is positively related to the sum of waiting times of the selected transactions through theoretical analysis. Based on this insight, we transform the fairness problem into the subset sum problem, which is to find a valid subset from a given set with subset sum as large as possible. However, it is time-consuming to solve the problem using a brute-force approach because there is an exponential number of subsets for a given set. To this end, we propose a heuristic and a min-heap-based optimal algorithm for different parameter settings. Finally, we analyze the time complexity of FAIR-PACK and conduct extensive experiments. The results reveal that FAIR-PACK is time-efficient and outperforms the existing algorithms significantly in terms of both fairness and average transaction response time.**

***Index Terms*—Blockchain, industrial Internet of things, transaction packing, fairness.**

I. INTRODUCTION

Recently, blockchain technology has been attracting extensive attention from both industry and academia, since it enables trustless data storage with auditability [1]. In industrial Internet of things (IIoT), blockchain has shown its great potential in vehicular networks [2], smart grid [3], crowd sourcing [4], mobile edge computing [5], etc. Generally, a blockchain is an append-only list of blocks, each of which includes a set of transactions, managed by a peer-to-peer network adhering to a protocol for inter-node communication and validating new blocks [6]. The magic of blockchain lies in the protocol of validating new blocks, i.e., consensus mechanism. In permissioned blockchains, the consensus mechanism is performed round by round, and each round consists of three phases, i.e., leader election, transaction packing, block propagation. To begin a round, all the blockchain nodes run the same leader election algorithm to elect a transaction packer. Then, the packer selects several transactions from its memory pool and pack them into a block. Finally, the generated block is broadcast to the network, and the transactions in the block get confirmed.

For IIoT systems, blockchain can be regarded as a service for data storage. In particular, the IIoT devices send the data, or transactions, to the blockchain, and receive the operational results, i.e., acceptance or rejection. To improve the quality of service, the research communities have been attempting to propose more efficient [7] and reliable [8] algorithms for leader election. Since the throughput of blockchain is limited, the blockchain service is supposed to be fairly shared among multiple IIoT devices. In permissionless blockchain, the IIoT devices pay fees, in forms of native cryptocurrencies, for their transactions. The nodes strategically pack those transactions with high transaction fees into a block to earn more monetary rewards. The fairness among the IIoT devices is naturally achieved because the transactions with higher transaction fees tend to be served first.

The fairness in permissioned blockchain empowered IIoT systems differs due to the lack of native cryptocurrencies and transaction fees. In this paper, we consider fairness in permissioned blockchain from the perspective of transaction response time. For each transaction, its response time is the duration from its submission to the time when a block containing this transaction is confirmed. A permissioned blockchain is considered to be fair if the response times of the transactions are close to each other. That is, the transactions which are submitted first are expected to be packed into blocks first. Unfairness leads to a high deviation of the transaction response times. As a result, the transactions incurring long delays will suffer from undesirable quality of experience, which is particularly important in cognitive IIoT [9]. More seriously, some transactions get timeouts and discarded if their response times exceed a certain period. In time-sensitive IIoT applications, e.g., energy trading [10] and manufacturing operation [11], the discarded transactions can lead to income loss and even safety issues.

Little attention has been paid to the fairness issue in permissioned blockchain empowered IIoT systems although it is vital. Transaction packing is the key to enhance the fairness because it directly decides which transactions are packed into blocks. The first idea is possibly first-come-first-serve (FCFS) [12]. In permissioned blockchain, the FCFS strategy is to select the transactions with long waiting times and pack them into a block. However, the selected subset of transactions can be invalid to be packed into a block. For example, one transaction cannot be packed if its dependent transactions

Shan Jiang, Jiannong Cao, Hanqing Wu, and Yanni Yang were with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China.
E-mail: {cssjiang, csjcao, cshwu, csynyang}@comp.polyu.edu.hk
Corresponding author: Jiannong Cao

are not confirmed yet. On this circumstance, the next choice is supposed be generated by the transaction packing algorithm while FCFS strategy fails. In existing permissioned blockchains, transactions are arbitrarily packed into blocks. An intuitive approach is to choose subsets one after another randomly. This approach cannot achieve preferable fairness since transactions with long waiting times are not considered first. In conclusion, traditional FCFS strategy fails to continuously generate subsets of transactions, and the random strategy cannot achieve satisfactory fairness.

In this paper, we propose FAIR-PACK, a fair transaction packing algorithm for permissioned blockchain empowered IIoT systems. First, we quantify the fairness according to Jain's fairness index [13] of response times, and define the fairness problem in permissioned blockchains formally. Then, we gain the insight that fairness is positively related to the sum of waiting times of the selected transactions. In this way, we transform the fairness problem into the subset sum problem, which is to find a valid subset with subset sum as large as possible. However, the number of subsets of a given set is exponential, which makes it non-trivial to solve the subset sum problem. We divide the subset sum problem into two individual problems depending on the relationship between the maximum size of the subset and the size of the given set. Furthermore, we figure out the partial/global orders of the subsets according to the subset sum, and propose a heuristic algorithm and a min-heap-based algorithm to solve the two problems separately. Finally, we analyze the time complexity of FAIR-PACK and extensively evaluate its performance in terms of fairness and average response time. Based on the experiments, we conclude that FAIR-PACK not only achieves better fairness, but reduces the average response time as well. The main contributions of this paper are as follows:

- We define the fairness problem in permissioned blockchain empowered IIoT systems. We propose an overall transaction packing algorithm FAIR-PACK and transform it into two subset sum problems via theoretical analysis. To the best of our knowledge, this is the first work on the transaction fairness problem in permissioned blockchain.
- Inside FAIR-PACK, we propose a heuristic and a min-heap-based optimal algorithm to solve the two subset sum problems separately. The performance and time complexity of the two algorithms are formally analyzed.
- We carry out extensive experiments on how the performance of FAIR-PACK is influenced by the transaction incoming rate, block generation time, block size, and block validity ratio. The results indicate that FAIR-PACK achieves better fairness and less average response time compared to the existing works.

II. SYSTEM MODEL AND PROBLEM STATEMENT

In this section, we first introduce the system model of permissioned blockchain empowered IIoT systems. Then, we define the fairness problem in permissioned blockchains with concrete explanations of the input, assumptions, and objective.

The block generation in a permissioned blockchain proceeds round by round as shown in Fig. 1. At the beginning of each round, the blockchain network runs the leader election algorithm to elect a leader, node i. Then, node i invokes transaction packing algorithm to select a subset of transactions from its local memory pool and pack them into block i. Finally, block i is propagated in the blockchain network through broadcasting. From the perspective of the IIoT devices, they submit their transactions to a random node in the blockchain network. Upon receiving the transactions, the node stores them in the local memory pool and broadcast the transactions to other nodes. Because broadcasting incurs network delay, the memory pools for different nodes may be different.

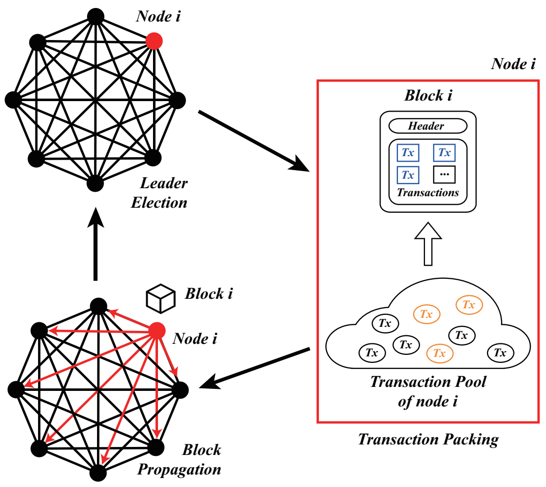


Fig. 1. Block generation of permissioned blockchains

The *waiting time* and *response time* of a transaction are defined as follows.

Definition 1. *Suppose a transaction x_i is submitted to blockchain network at time s_i and x_i is in memory pool at the current time t_c, then the waiting time of x_i is $a_i = t_c - s_i$.*

Definition 2. *Suppose a transaction x_i is submitted to the blockchain network and packed into blocks at time s_i and e_i, respectively, then the response time of x_i is $t_i = e_i - s_i$.*

As the permissioned blockchain runs round by round, there are more and more transactions packed into blocks. This paper studies the fairness according to Jain's fairness index [13]. Note that the fairness index is between 0 (exclusive) and 1 (inclusive). A larger fairness index means better fairness and the fairness index equals to 1 if the response times of all the transactions are the same.

Definition 3. *Suppose there are n transactions $\mathcal{X} = \{x_1, \cdots, x_n\}$ packed into blocks with response times $t_1, \cdots, t_n$, then the fairness among the n transactions is defined as: $\mathcal{J}(\mathcal{X}) = \frac{(\Sigma_{i=1}^n t_i)^2}{n \cdot \Sigma_{i=1}^n t_i^2}$.*

To maximize the overall fairness, we should consider not only the response times of the transactions in blocks but also the waiting times of the transactions in memory pool. However, the number of transactions in blocks increases infinitely as the permissioned block runs, which hinders the development of a time-efficient transaction packing algorithm. In this paper, we only consider the waiting times of the transactions in memory pool in a single round and the expected fairness of a given packing strategy.

Definition 4. *Suppose there are n transactions $\mathcal{X}$, the maximum number of transactions in a block is k, and the time to make a packed block to be committed is r. Consider a packing strategy which packs a subset $\mathcal{X}'$ of transactions into a block in a round. Suppose the waiting times of $\mathcal{X}'$ in the round are $s_1, \cdots, s_l$ and the waiting times of the remaining transactions $\mathcal{X} \setminus \mathcal{X}'$ are $t_1, \cdots, t_m$, where $l + m = n$, $l < k$, and $t_1 \geq \cdots \geq t_m$. Then, the expected fairness of the packing strategy in the round is defined to be:*

$$\mathcal{J}(\mathcal{X}) = \frac{(\Sigma_{i=1}^{l}(s_i + r) + \Sigma_{i=1}^{m}(t_i + r + \lceil \frac{i}{k} \rceil \cdot r))^2}{n \cdot (\Sigma_{i=1}^{l}(s_i + r)^2 + \Sigma_{i=1}^{m}(t_i + r + \lceil \frac{i}{k} \rceil \cdot r)^2)} \quad (1)$$

In short, the expected fairness assumes that the remaining transactions in the memory pool are packed into blocks in FCFS order. To this end, we aim at finding a packing strategy with the maximum expected fairness in each round, which is formally defined as follows:

Definition 5. *Problem Ori-Fair: Given 1) a set of n transactions $\mathcal{X}$ in memory pool at time t_c with submission times $s_1, \cdots, s_n$, respectively and 2) the maximum number k of transactions that can be packed into a block, assuming 1) a leader is already elected for transaction packing, 2) a subset of $\mathcal{X}$ can be valid or invalid to be packed into a block, and 3) the validity of all subsets of $\mathcal{X}$ are unknown before generation, we aim to develop a transaction packing strategy to continuously generate subsets of $\mathcal{X}$ until a valid subset is generated with the expected fairness $\mathcal{J}(\mathcal{X})$ as large as possible.*

In the problem, the maximum number of transactions that can be packed into a block is given as k. The reason why k is bounded is that a large value of k leads to high network congestion when the block is propagated in the network. For example, the value of k in Bitcoin is around $3,000$ due to the limit of block size and average transaction size. In this paper, we consider k as an adjustable parameter.

In the following, we explain the reasonability of the assumptions. First, leader election and transaction packing are conducted in sequence. As a result, we can use existing leader election methods, such as [14] and [15], to select a transaction packer to fit our assumption 1). Second, a subset of transactions can be valid or invalid to be packed into a block for various reasons. For example, one transaction cannot be packed if its dependent transactions are not confirmed yet. Finally, we assume the validities of all subsets of transactions are unknown before the generation to separate transaction packing with block verification. After separation, the transaction packing algorithm will be an independent component in the blockchain, which makes blockchain more modularized.

Our target is to develop a fair transaction packing algorithm. The algorithm is supposed to continuously generate subsets of transactions because a subset of transactions can be invalid and its validity is unknown in advance.

III. Fair-Pack: a Fairness-based Transaction Packing Algorithm

In this section, we prove that the fairness index is positively related to the sum of waiting times of the packed transactions in Ori-Fair. Then, the proved property is used to transform Ori-Fair into the subset sum problem. Finally, we propose an overall solution Fair-Pack towards solving Ori-Fair.

Theorem 1. *Given a set of n transactions $x_1, \cdots, x_n$ in pool with waiting times $a_1, \cdots, a_n$, respectively and k transactions are supposed to be packed, in each round, the larger the sum of the waiting times of the packed transactions, the larger the fairness of the packing strategy.*

Proof. Consider two permutations σ and τ of $(1, \cdots, n)$, where $\sigma = (\sigma_1, \cdots, \sigma_n)$ and $\tau = (\tau_1, \cdots, \tau_n)$. The two packing strategies σ-Pack and τ-Pack pack transactions in the order of $(x_{\sigma_1}, \cdots, x_{\sigma_n})$ and $(x_{\tau_1}, \cdots, x_{\tau_n})$, respectively.

Assume by contradictory σ-Pack packs transactions with larger sum of waiting times each round while τ-Pack achieves larger fairness. By definition, we have the following properties:

$$\Sigma_{i=1}^{k} a_{\sigma_i} > \Sigma_{i=1}^{k} a_{\tau_i} \quad (2)$$

$$\forall 2 \leq j < \lceil \frac{n}{k} \rceil, \Sigma_{i=1}^{jk} a_{\sigma_i} \geq \Sigma_{i=1}^{jk} a_{\tau_i} \quad (3)$$

$$\Sigma_{i=1}^{n} a_{\sigma_i} = \Sigma_{i=1}^{n} a_{\tau_i} \quad (4)$$

Notate the time to commit a packed block as t_p. Then the transaction response times using σ-Pack are $a_{\sigma_1} + t_p, a_{\sigma_2} + t_p, \cdots, a_{\sigma_{k+1}} + 2t_p, \cdots, a_{\sigma_n} + \lceil \frac{n}{k} \rceil \cdot t_p$. The transaction response times using τ-Pack are $a_{\tau_1} + t_p, a_{\tau_2} + t_p, \cdots, a_{\tau_{k+1}} + 2t_p, \cdots, a_{\tau_n} + \lceil \frac{n}{k} \rceil \cdot t_p$. To this end, the fairness of σ-Pack and τ-Pack and their relationship are as follows:

$$\mathcal{J}_{\sigma-Pack} = \frac{(\Sigma_{i=1}^{n}(a_{\sigma_i} + \lceil \frac{i}{k} \rceil \cdot t_p))^2}{n \cdot \Sigma_{i=1}^{n}(a_{\sigma_i} + \lceil \frac{i}{k} \rceil \cdot t_p)^2} \quad (5)$$

$$\mathcal{J}_{\tau-Pack} = \frac{(\Sigma_{i=1}^{n}(a_{\tau_i} + \lceil \frac{i}{k} \rceil \cdot t_p))^2}{n \cdot \Sigma_{i=1}^{n}(a_{\tau_i} + \lceil \frac{i}{k} \rceil \cdot t_p)^2} \quad (6)$$

$$\mathcal{J}_{\tau-Pack} > \mathcal{J}_{\sigma-Pack} \quad (7)$$

Since the algorithms are running on the same set of transactions, we have

$$\Sigma_{i=1}^{n} a_{\sigma_i}^2 = \Sigma_{i=1}^{n} a_{\tau_i}^2 \quad (8)$$

$$\Sigma_{i=1}^{n}(a_{\sigma_i} + \lceil \frac{i}{k} \rceil \cdot t_p) = \Sigma_{i=1}^{n}(a_{\tau_i} + \lceil \frac{i}{k} \rceil \cdot t_p) \quad (9)$$

According to Eq. 5,6,7,9, we have:

$$\Sigma_{i=1}^{n}(a_{\sigma_i} + \lceil \frac{i}{k} \rceil \cdot t_p)^2 > n \cdot \Sigma_{i=1}^{n}(a_{\tau_i} + \lceil \frac{i}{k} \rceil \cdot t_p)^2 \quad (10)$$

Expand Eq. 10, we get:

$$\begin{aligned} \Sigma_{i=1}^{n} a_{\sigma_i}^2 + \Sigma_{i=1}^{n}(\lceil \frac{i}{k} \rceil \cdot t_p)^2 + 2\Sigma_{i=1}^{n}(a_{\sigma_i} \cdot \lceil \frac{i}{k} \rceil \cdot t_p) > \\ \Sigma_{i=1}^{n} a_{\tau_i}^2 + \Sigma_{i=1}^{n}(\lceil \frac{i}{k} \rceil \cdot t_p)^2 + 2\Sigma_{i=1}^{n}(a_{\tau_i} \cdot \lceil \frac{i}{k} \rceil \cdot t_p) \end{aligned} \quad (11)$$

According to Eq. 8,11, we have:

$$\Sigma_{i=1}^{n}(a_{\sigma_i} \cdot \lceil \frac{i}{k} \rceil) > \Sigma_{i=1}^{n}(a_{\tau_i} \cdot \lceil \frac{i}{k} \rceil) \quad (12)$$

Adding Eq. 2 and all the inequations in Eq. 3, we have

$$\Sigma_{i=1}^{\lceil\frac{n}{k}\rceil-1}\Sigma_{j=1}^{ik}a_{\sigma_j} > \Sigma_{i=1}^{\lceil\frac{n}{k}\rceil-1}\Sigma_{j=1}^{ik}a_{\tau_j} \tag{13}$$

Adding the inequations in Eq. 12 and Eq. 13, we have:

$$\begin{aligned}\lceil\frac{n}{k}\rceil\Sigma_{i=1}^{n}a_{\sigma_i} &= \Sigma_{i=1}^{\lceil\frac{n}{k}\rceil-1}\Sigma_{j=1}^{ik}a_{\sigma_j} + \Sigma_{i=1}^{n}(a_{\sigma_i}\cdot\lceil\frac{i}{k}\rceil)\\ &> \Sigma_{i=1}^{\lceil\frac{n}{k}\rceil-1}\Sigma_{j=1}^{ik}a_{\tau_j} + \Sigma_{i=1}^{n}(a_{\tau_i}\cdot\lceil\frac{i}{k}\rceil)\\ &= \lceil\frac{n}{k}\rceil\Sigma_{i=1}^{n}a_{\tau_i}\end{aligned} \tag{14}$$

It is clear that Eq. 14 is contradictory to Eq. 4. As a result, the assumption does not hold and σ-PACK achieves larger fairness than τ-PACK. □

According to Thm. 1, it achieves better fairness to pack transactions with the larger sum of waiting times. Therefore, the best strategy is to pack transactions with top-k waiting times, which is FCFS. However, such a transaction subset can be invalid and we need to continuously generate transaction subsets. According to Thm. 1, the sum of waiting times can be treated as the heuristic to generate transaction subsets. That is, we are supposed to find the transaction subset with the 1-st, 2-nd, $\cdots$, m-th largest sum of waiting times.

Because the transaction waiting times are known real numbers, the problem is to find a subset of no more than k elements from a set of n real numbers with the m-th largest subset sum among all the feasible subsets. Here, the feasibility means the subsets contain no more than k elements. If k is smaller than n, there are $\Sigma_{i=0}^{k}\binom{n}{i}$ feasible subsets. Similarly, there are 2^n ones if k is no smaller than n. In this paper, we consider the two conditions separately with problem statements as follows.

Definition 6. *Problem SM-Sum: Given a set of n positive real numbers $\mathcal{W}$, a positive integer k where $k < n$, and a positive integer m where $m \leq \Sigma_{i=0}^{k}\binom{n}{i}$, there are $\Sigma_{i=0}^{k}\binom{n}{i}$ distinct subsets of $\mathcal{W}$ of size no larger than k. Among the subsets, find the one with the m-th largest sum.*

Definition 7. *Problem LM-Sum: Given a set of n positive real numbers $\mathcal{W}$, a positive integer k where $k \geq n$, and a positive integer m where $m \leq 2^n$, there are 2^n distinct subsets of $\mathcal{W}$ of size no larger than k. Among the subsets, find the one with the m-th largest sum.*

If the problems SM-SUM and LM-SUM are solved, then the problem ORI-FAIR can be solved by Algo. 1.

In Algo. 1, we assume that the problem SM-SUM and LM-SUM are solved by SUM-INDEX and MIN-HEAP-OP, respectively. First, we compute the transaction waiting times based on the submission times and the current timestamp. Then, the transactions are sorted according to the waiting times in a non-increasing order. That is, w_i will be no smaller than w_j if $i < j$ after sorting. In the for-loop, we transform the problem of finding the transaction subset with the 1-st, 2-nd, $\cdots$, m-th largest sum of waiting times to SM-SUM or LM-SUM depending on the relationship between k and n. Both the procedures of SUM-INDEX and MIN-HEAP-OP will return the indexes of the selected elements. Finally, we derive the transaction subset based on the index set as the output of FAIR-PACK.

Algorithm 1 FAIR-PACK: a fairness-based transaction packing algorithm for problem ORI-FAIR

Input: n: the memory pool size; $\mathcal{X} = \{x_1, \cdots, x_n\}$: the transactions in memory pool; $\mathcal{S} = \{s_1, \cdots, s_n\}$: the transaction submission times; k: the maximum number of transactions in a block; t_c: the current timestamp; IS-VALID($\mathcal{U}$): a procedure to check the validity of transaction subset $\mathcal{U}$; SUM-INDEX($\mathcal{W}, n, k, m$): a procedure to solve SM-SUM; MIN-HEAP-OP($\mathcal{W}, n, k, m$): a procedure to solve LM-SUM
Output: a valid transaction subset of $\mathcal{X}$ or NIL in case that all subsets are invalid

```
1: $\mathcal{W} \leftarrow t_c - \mathcal{S}$
2: Sort $\mathcal{X}$ with respect to $\mathcal{W}$ in non-increasing order
3: for $m \leftarrow 1$ to $\infty$ do
4:     if $k < n$ then $id \leftarrow$SUM-INDEX($\mathcal{W}, n, k, m$).MAIN()
5:     else $id \leftarrow$ MIN-HEAP-OP($\mathcal{W}, n, k, m$).MAIN()
6:     end if
7:     if $id =$ NIL return NIL end if
8:     $\mathcal{U} \leftarrow x_{id}$
9:     if IS-VALID($\mathcal{U}$) return $\mathcal{U}$ end if
10: end for
```

IV. SUM-INDEX: A HEURISTIC SOLUTION TO SM-SUM

In this section, we focus on solving SM-SUM. In particular, we propose to use directed acyclic graph (DAG) $\mathcal{G}$ to represent all the $\Sigma_{i=0}^{k}\binom{n}{i}$ subsets. In $\mathcal{G}$, we prove a partial order among the subsets with respect to the subset sum, which leads to the heuristic to enumerate the subsets according to the index sum. That is, the subset sum is related to the number of elements and index sum. Such a heuristic is leveraged in algorithm SUM-INDEX to solve SM-SUM. To begin with, we define the terminologies of *set sum* and *index sum* as follows.

Definition 8. *Given a set of n real numbers $\mathcal{W}$, the set sum of $\mathcal{W}$ is defined to be the sum of all the elements in $\mathcal{W}$, i.e., $\mathcal{E}(\mathcal{W}) = \Sigma_{i=1}^{n}w_i$.*

Definition 9. *Given a set of n positive real numbers $\mathcal{W} = \{w_1, \cdots, w_n\}$ and a subset $\mathcal{I} = \{w_{\sigma_1}, \cdots, w_{\sigma_p}\}$ of $\mathcal{W}$, the index sum of $\mathcal{I}$ is defined to be the sum of the indexes of its corresponding elements in $\mathcal{W}$, i.e., $\mathcal{D}(\mathcal{I}) = \Sigma_{i=1}^{p}\sigma_p$.*

There are $\Sigma_{i=0}^{k}\binom{n}{i}$ nodes in $\mathcal{G}$, in which each node represents a subset. Moreover, $\mathcal{G}$ consists of $k+1$ connected components, in which the p-th component is the collection of subsets of size $p-1$. The number of subsets of size i, i.e., $\binom{n}{p}$, is exactly the size of the p-th component. Furthermore, the subset in the i-th component tends to have a smaller subset sum than the subset in the j-th component if $i < j$ because of the essential difference in subset sizes. This also indicates a partial order between the components in terms of subset sum. Next, we introduce the directed edges in each component.

In the p-th component of $\mathcal{G}$, all the subsets of size p are listed level by level according to the index sum as shown in Fig. 2. In the figure, the "greater" operator between two subsets represents the relationship of subset sum between them. We find that the subset with a smaller index sum is likely to have a larger set sum. In particular, given a subset $\mathcal{I}$ whose index sum is not the smallest, there must be another subset $\mathcal{I}'$ with smaller index sum and larger subset sum, which is proved in Thm. 2. Moreover, we add a directed edge from the node representing $\mathcal{I}'$ to the node representing $\mathcal{I}$. In this way, all the nodes are (weakly) connected in the p-th components. Because the edges are always from the upper level, i.e., smaller index sum, to the lower level, i.e., larger index sum, the p-th components is a directed and acyclic.

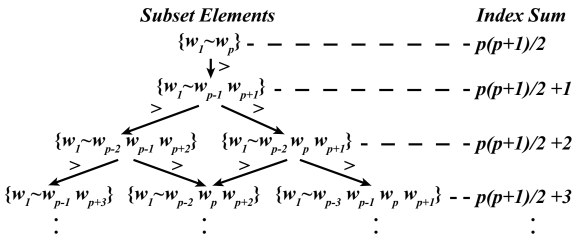


Fig. 2. Subsets of size p listed level by level according to index sum

Theorem 2. *Given a set of n positive real numbers $\mathcal{W} = \{w_1, \cdots, w_n\}$ where $w_1 \geq \cdots \geq w_n$ and a subset $\mathcal{I}$ of $\mathcal{W}$ where $|\mathcal{I}| = p$, $\mathcal{D}(\mathcal{I}) = d$, and $d \neq p(p+1)/2$, there exists at least one subset $\mathcal{I}'$ of $\mathcal{W}$ such that $|\mathcal{I}'| = p$, $\mathcal{D}(\mathcal{I}') = d - 1$, and $\mathcal{E}(\mathcal{I}') \geq \mathcal{E}(\mathcal{I})$.*

Proof. Let $\mathcal{I} = \{w_{\sigma_1}, \cdots, w_{\sigma_p}\}$ where $\Sigma_{i=1}^{p} \sigma_p = d$ and $\sigma_1 < \cdots < \sigma_p$. Assume, for the sake of contradiction, that for all i, $w_{\sigma_i - 1} \in \mathcal{I}$ if $\sigma_i > 1$ (C1). Assume, for the sake of contradiction, that $\sigma_1 > 1$ (C2). We have $w_{\sigma_1 - 1} \in \mathcal{I}$ according to assumption C1. However, w_{σ_1} is the largest element in $\mathcal{I}$. Therefore, C2 does not hold and $\sigma_1 = 1$. Similarly, we can get $\sigma_i = i$ for every $1 \leq i \leq p$. Therefore, $d = \Sigma_{i=1}^{p} \sigma_i = p(p+1)/2$, which contradicts the condition that $d \neq p(p+1)/2$. Hence, C1 does not hold and there exists at least one i such that $\sigma_i > 1$ and $w_{\sigma_i - 1} \notin \mathcal{I}$.

Let $\sigma_t > 1$ and $w_{\sigma_t - 1} \notin \mathcal{I}$. We construct $\mathcal{I}' = \mathcal{I} \setminus \{w_{\sigma_t}\} \cup \{w_{\sigma_t - 1}\}$ satisfying the conditions in the theorem as follow.

- $\mathcal{I}' \subset \mathcal{W}$ since $\mathcal{I} \subset \mathcal{W}$ and $\{w_{\sigma_t - 1}\} \in \mathcal{W}$.
- $|\mathcal{I}'| = |\mathcal{I}| - 1 + 1 = p$.
- $\mathcal{D}(\mathcal{I}') = \mathcal{D}(\mathcal{I}) - \sigma_t + (\sigma_t - 1) = d - 1$.
- $\mathcal{E}(\mathcal{I}') \geq \mathcal{E}(\mathcal{I})$ since $\mathcal{E}(\mathcal{I}') - \mathcal{E}(\mathcal{I}) = w_{\sigma_t - 1} - w_{\sigma_t} \geq 0$.

□

We apply the above construction to all the $k+1$ components in $\mathcal{G}$, resulting in $\mathcal{G}$ to be a DAG. The partial orders among the subsets is reveal in $\mathcal{G}$, which gives birth to a heuristic algorithm SUM-INDEX to solving SM-SUM as shown in Algo. 2.

In SUM-INDEX, we do not order all the subsets to find the subset with exactly the m-th largest subset sum since the time complexity, i.e., $O(\Sigma_{i=0}^{k}\binom{n}{i} \cdot \log(\Sigma_{i=0}^{k}\binom{n}{i}))$, is too high. Instead, we enumerate the subset size in decreasing order, the index sum of the subsets in increasing order, and the subset in lexicographical order sequentially. The algorithm SUM-INDEX begins with the first subset of size k and index sum $k(k+1)/2$. Such a subset contains exactly the k largest elements in $\mathcal{W}$. To find the next subset, procedure NEXT-SUBSET first tries to invoke the procedure NEXT-PD to find a subset with the same subset size and index sum. In detail, NEXT-PD takes a subset

Algorithm 2 SUM-INDEX: a heuristic algorithm for SM-SUM

Input: $\mathcal{W}$: a set of n positive real numbers; k: a positive integer that $k < n$; m: a positive integer that $m \leq \Sigma_{i=0}^{k}\binom{n}{i}$

Output: the indexes of the subset of $\mathcal{W}$ with approximately the m-th largest subset sum among all the $\Sigma_{i=0}^{k}\binom{n}{i}$ subsets

```
procedure MAIN( )
    if $m = 1$ return FIRST-SUBSET( ) end if
    return NEXT-SUBSET( )
end procedure
procedure FIRST-SUBSET( )
    global $gp \leftarrow k$ ▷ "global" variable for all procedures
    global $gd \leftarrow \frac{gp(gp+1)}{2}$
    return FIRST-PD($gp, gd$)
end procedure
procedure NEXT-SUBSET( )
    return NEXT-PD($gp, gd$) if not NIL
    $gd \leftarrow gd + 1$
    return FIRST-PD($gp, gd$) if not NIL
    $(gp, gd) \leftarrow (gp - 1, \frac{gp(gp+1)}{2})$
    if $gp = 0$ return NIL end if
    return FIRST-PD(gp, gd)
end procedure
procedure FIRST-PD($p, d$)
    if $d < \frac{(p+1)p}{2}$ or $d > \frac{(n-p+1+n)p}{2}$ then return NIL
    end if
    $r \leftarrow$ an array of size $p$ in which all elements are $0$
    for $i \leftarrow 1$ to $p$ do
        $r_i \leftarrow \max(r_{i-1} + 1, d - \frac{(n-p+i+1+n)(p-i)}{2})$
        $d \leftarrow d - r_i$
    end for
    return $r$
end procedure
procedure NEXT-PD($p, d, r$)
    $s \leftarrow$ an array of size $p$ in which all elements are $0$
    for $i \leftarrow 1$ to $p$ do $s_i \leftarrow s_{i-1} + r_i$ end for
    for $i \leftarrow p - 1$ to $1$ do
        if $r_i+1 < r_{i+1}$ and $d-s_i-1 \geq \frac{(r_i+r_i+p-i+3)(p-i)}{2}$
and $d - s_i - 1 \leq \frac{(n+i+1-p+n)(p-i)}{2}$ then
            $(r_i, d) \leftarrow (r_i + 1, d - s_i - 1)$
            for $j \leftarrow i + 1$ to $p$ do
                $r_j \leftarrow \max(r_{j-1}+1, d-\frac{(n+j+1-p+n)(p-j)}{2})$
                $d \leftarrow d - r_j$
            end for
            return $r$
        end if
    end for
    return NIL
end procedure
```

r as input and outputs the next subset of r in lexicographical order. If such a subset is not found, NEXT-SUBSET increases the desired index sum and invokes procedure FIRST-PD to find the first subset in lexicographical order. Finally, if the index sum exceeds the limit, NEXT-SUBSET will increase the subset size and set the index sum to the smallest one.

V. MIN-HEAP-OP: AN OPTIMAL SOLUTION TO LM-SUM

In this section, we propose algorithm MIN-HEAP-OP to solve LM-SUM. We build a min-heap to maintain the relationship in terms of subset sums among the 2^n subsets, and the subsets are generated by manipulating the min-heap.

To begin with, we construct a binary tree $\mathcal{H}(n)$ as follows:

- $\mathcal{H}(n).root = \{n\}$
- For each element $e \in \mathcal{H}(n)$ in which $\min(e) \neq 1$, $e.lc = e \setminus \{\min(e)\} \cup \{\min(e) - 1\}$
- For each element $e \in \mathcal{H}(n)$ in which $\min(e) \neq 1$, $e.rc = e \cup \{\min(e) - 1\}$

An example of $\mathcal{H}(4)$ is shown in Fig. 3(a). We see that $\mathcal{H}(4)$ is a complete binary tree which contains and only contains all the subsets of $\{1, 2, 3, 4\}$. Next, we prove it in case of n through Thm. 3 and Thm. 4.

Theorem 3. *The tree $\mathcal{H}(n)$ contains all the non-empty subsets of $U = \{1, 2, \cdots, n\}$.*

Proof. Consider an arbitrary subset I of U. We prove the lemma by induction on the minimum value of I.

Base case. When $\min\{I\} = n$, we can infer that $I = \{n\}$ because $I \subseteq U$ and $\max\{U\} = n$. Therefore, I is an element, in particular, the root of $\mathcal{H}(n)$.

Induction step. Let $1 < k \leq n$ and assume I is an element of $\mathcal{H}(n)$ as long as $\min(I) \geq k$. We aim to prove that I is an element of $\mathcal{H}(n)$ as long as $\min(I) = k - 1$. We consider three circumstances as follows.

- Case 1: $|I| = 1$. We can infer that $I = \{k - 1\}$ as $\min(I) = k - 1$. Consider another subset $I' = \{k\}$. Because $I' \subseteq U$ and $\min(I') = k$, we know I' is an element of $\mathcal{H}(n)$. $I'.lc = I' \setminus \{k\} \cup \{k - 1\} = I$. As a result, I is an element of $\mathcal{H}(n)$ as well.
- Case 2: $|I| \neq 1$ and $\min(I \setminus \{k - 1\}) = k$. Consider another subset $I' = I \setminus \{k - 1\}$. As $I' \subseteq I \subseteq U$ and $\min(I') = k$, I' is an element of $\mathcal{H}(n)$. $I'.rc = I' \cup \{k - 1\} = I$. Therefore, I is also an element of $\mathcal{H}(n)$.
- Case 3: $|I| \neq 1$ and $\min(I \setminus \{k - 1\}) \neq k$. Consider another subset $I' = I \setminus \{k - 1\} \cup \{k\}$. Because $I \subseteq U$ and $k \in U$, we can infer that $I' \subseteq U$. Furthermore, I' is an element of $\mathcal{H}(n)$ as $\min(I') = k$. $I'.lc = I' \setminus \{k\} \cup \{k - 1\} = I$. Hence, I is an element of $\mathcal{H}(n)$ as well.

The above three cases cover all the subsets whose minimum value equals $k - 1$. Meanwhile, we show the subsets are elements of $\mathcal{H}(n)$ for all the three cases. Therefore, I is an element of $\mathcal{H}(n)$ as long as $\min(I) = k - 1$.

Based on the base case and induction step, we conclude that $I \subseteq U$ is an element of $\mathcal{H}(n)$ as long as $\min(I) \geq 1$, which completes the proof. □

Theorem 4. *The tree $\mathcal{H}(n)$ is a complete binary tree, which contains and only contains all the non-empty subsets of $U = \{1, 2, \cdots, n\}$.*

Proof. Notate the set of elements at level k and its size as $\mathcal{H}(n, k)$ and $|\mathcal{H}(n, k)|$, respectively. In the following, we prove that $|\mathcal{H}(n, k)| \leq 2^{k-1}$ and $\min(e) = n + 1 - k$ for any $1 \leq k \leq n - 1$ and $e \in \mathcal{H}(n, k)$ by induction on the value of k.

Base Case. When $k = 1$, there is only one element $\{n\}$ in the first level of the tree $\mathcal{H}(n)$, i.e., $\mathcal{H}(n, k) = \{\{n\}\}$. We can get that $\mathcal{H}(n, k) = 1 \leq 2^{k-1}$ and $\min(\{n\}) = n = n + 1 - k$.

Induction Step. Let k be an integer that $1 \leq k < n - 1$ and assume that $|\mathcal{H}(n, k)| \leq 2^{k-1}$ and $\min(e) = n + 1 - k$ for any $e \in \mathcal{H}(n, k)$. We aim to prove that $|\mathcal{H}(n, k+1)| \leq 2^k$ and $\min(e) = n - k$ for any $e \in \mathcal{H}(n, k + 1)$. We prove the two statements separately as follows.

- There are exactly two children for each element in $\mathcal{H}(n, k)$ according to the definition of the tree $\mathcal{H}(n)$. If there is no overlapping element among all the children of all the elements in $\mathcal{H}(n, k)$, $|\mathcal{H}(n, k + 1)|$ will be exactly twice the value of $|\mathcal{H}(n, k)|$, i.e., $|\mathcal{H}(n, k+1)| = 2 \cdot |\mathcal{H}(n, k)| \leq 2 \cdot 2^{k-1} = 2^k$. If any overlapping element, the value of $|\mathcal{H}(n, k + 1)|$ will be smaller, i.e., $|\mathcal{H}(n, k + 1)| \leq 2^k$. As a result, $|\mathcal{H}(n, k + 1)| \leq 2^k$.
- Considering an arbitrary element $e \in \mathcal{H}(n, k + 1)$, e must be a child of some element $e' \in \mathcal{H}(n, k)$. We can get $\min(e') = n + 1 - k$ according to the assumption. If $e'.lc = e$, then $\min(e) = \min(e' \setminus \{\min(e')\} \cup \{\min(e') - 1\}) = n - k$. Otherwise, $e'.rc = e$. Under this circumstance, $\min(e) = \min(e' \cup \{\min(e') - 1\}) = n - k$. As a result, $\min(e) = n - k$ for any $e \in \mathcal{H}(n, k + 1)$.

Based on the base case and the induction step, we draw the conclusion that $|\mathcal{H}(n, k)| \leq 2^{k-1}$ and $\min(e) = n + 1 - k$ for any $1 \leq k \leq n - 1$ and any $e \in \mathcal{H}(n, k)$. Consider $k = n$, we can infer that $\min(e) = 1$ for any $e \in \mathcal{H}(n, n)$. Therefore, all the elements in $\mathcal{H}(n, n)$ have no child, i.e., $\mathcal{H}(n)$ consists of exactly n levels. Therefore, $|\mathcal{H}(n)| = \Sigma_{i=1}^{n} |\mathcal{H}(n, i)| \leq \Sigma_{i=1}^{n} 2^{i-1} = 2^n - 1$. In another word, the size of the tree $\mathcal{H}(n)$ is no more than $2^n - 1$.

The number of non-empty subsets of U is $2^n - 1$. According to Lem. 3, the size of $\mathcal{H}(n)$ is no less than $2^n - 1$, which is the number of non-empty subsets of U. Therefore, the size of $\mathcal{H}(n)$ is exactly $2^n - 1$. Meanwhile, $\mathcal{H}(n)$ contains and only contains all the non-empty subsets of U. Moreover, $\mathcal{H}(n)$ consists of exactly n levels. As a result, $\mathcal{H}(n)$ is a complete binary tree, which completes the proof. □

Based on the binary tree $\mathcal{H}(n)$, we construct a minimum heap $\mathcal{W}_{\mathcal{H}(n)}$ as follows:

- $\mathcal{W}_{\mathcal{H}(n)}.root.value = \mathcal{H}(n).root$
- For each element $e \in \mathcal{W}_{\mathcal{H}(n)}$ in which $e.value.lc \neq$ NIL, $e.lc.value = e.value.lc$
- For each element $e \in \mathcal{W}_{\mathcal{H}(n)}$ in which $e.value.rc \neq$ NIL, $e.rc.value = e.value.rc$
- For each element $e \in \mathcal{W}_{\mathcal{H}(n)}$, $e.key = \mathcal{E}(\mathcal{W}_{e.value})$

Each element in $\mathcal{H}(n)$ represents the indexes of a selected subset of $\mathcal{W}$. For example, Fig. 3(b) shows the subsets generated according to $\mathcal{H}(n)$ when $n = 4$. In Fig. 3(b), we can see the subset sum of a parent node is always no less than the one of a child node, which implies $\mathcal{H}(4)$ to be a min-heap. In the following, we formally prove that $\mathcal{H}(n)$ is a min-heap based on the subset sum as shown in Thm. 5.

Theorem 5. *$\mathcal{W}_{\mathcal{H}(n)}$ is a binary min-heap.*

Proof. On one hand, $\mathcal{W}_{\mathcal{H}(n)}$ is a complete binary tree since the value field in $\mathcal{W}_{\mathcal{H}(n)}$ is exactly the same with $\mathcal{H}(n)$ while

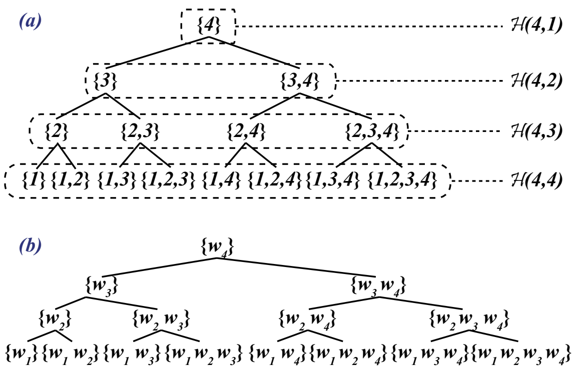


Fig. 3. (a) $\mathcal{H}(4)$ and (b) $\mathcal{W}_{\mathcal{H}(4)}$

$\mathcal{H}(n)$ is a complete binary tree as proven in Thm. 4. On the other hand, $\mathcal{W}_{\mathcal{H}(n)}$ satisfies the min-heap property, which is demonstrated as follows.

- For each e in $\mathcal{W}_{\mathcal{H}(n)}$ with left child, i.e., $e.lc \neq$ NIL, the key of $e.lc$ must be no smaller than the key of e.

$$\begin{aligned} & e.lc.key \\ &= \mathcal{E}(\mathcal{W}_{e.lc.value}) = \mathcal{E}(\mathcal{W}_{e.value.lc}) \\ &= \mathcal{E}(\mathcal{W}_{e.value \setminus \{\min(e.value)\} \cup \{\min(e.value)-1\}}) \\ &= \mathcal{E}(\mathcal{W}_{e.value}) - w_{\min(e.value)} + w_{\min(e.value)-1} \\ &= e.key - (w_{\min(e.value)} - w_{min(e.value)-1}) \\ &\geq e.key \end{aligned}$$

- For each e in $\mathcal{W}_{\mathcal{H}(n)}$ with right child, i.e., $e.rc \neq$ NIL, the key of $e.rc$ must be no smaller than the key of e.

$$\begin{aligned} e.rc.key &= \mathcal{E}(\mathcal{W}_{e.value.rc}) \\ &= \mathcal{E}(\mathcal{W}_{e.value \cup \{\min(e.value)-1\}}) \\ &= \mathcal{E}(\mathcal{W}_{e.value}) + w_{\min(e.value)-1} \\ &= e.key + w_{min(e.value)-1} \\ &\geq e.key \end{aligned}$$

To summarize, $\mathcal{W}_{\mathcal{H}(n)}$ is a binary min-heap as it is a complete binary tree and satisfies the min-heap property, which completes the proof. □

Note that min-heap is an efficient data structure to find the k-th minimum element. We leverage the min-heap property to solve the problem LM-SUM as shown in Algo. 3.

In Algo. 3, we build a min-heap $\mathcal{W}_{\mathcal{H}(n)}$ to represent all the subsets of $\mathcal{W}$. In case that m equals 1, Algo. 3 directly returns the whole set $\mathcal{W}$ because $\mathcal{W}$ owns the largest subset sum essentially. Otherwise, we applies the DELETE-MIN operation to $\mathcal{W}_{\mathcal{H}(n)}$ to get its root r. The key of r is the smallest subset sum according to the min-heap property. Therefore, we exclude the elements in $r.value$ from the whole set and get the transaction indexes to be selected. DELETE-MIN will also deletes the minimum element, i.e., the root, from the min-heap and maintains the min-heap property. In this way, different subsets can be generated continuously.

Algorithm 3 MIN-HEAP-OP: a min-heap-based algorithm to solve LM-SUM

Input: $\mathcal{W}$: a set of n positive real numbers; k: a positive integer that $k > n$; m: a positive integer that $m \leq 2^n$
Output: the indexes of the subset of $\mathcal{W}$ with the m-th largest subset sum among all the 2^n possible subsets

```
1: procedure MAIN( )
2:     if $m = 1$ return FIRST-SUBSET( ) end if
3:     return NEXT-SUBSET( )
4: end procedure
5: procedure FIRST-SUBSET( )
6:     global $\mathcal{W}_{\mathcal{H}(n)} \leftarrow$ a min-heap built as stated
7:     return $\{1, 2, \cdots, n\}$
8: end procedure
9: procedure NEXT-SUBSET( )
10:     $r \leftarrow$ DELETE-MIN($\mathcal{W}_{\mathcal{H}(n)}$)
11:     if $r \neq \emptyset$ then return $\{1, 2, \cdots, n\} \setminus r.value$ end if
12:     return NIL
13: end procedure
```

VI. TIME COMPLEXITY ANALYSIS

In this section, we analyze the time complexity of the algorithms SUM-INDEX, MIN-HEAP-OP, and FAIR-PACK.

Theorem 6. *The time complexity of* SUM-INDEX *is $O(n)$.*

Proof. As shown in Algo. 2, the main procedure of SUM-INDEX finally calls the procedure FIRST-PD on line 13 or 16, or the procedure NEXT-PD on line 11. Note that the time complexity of SUM-INDEX is irrelevant to the value of m since m is only an indicator for whether the first subset is to be generated.

In terms of FIRST-PD, it contains a for-loop from 1 to p. Because p, as the number of elements in the subset, is of $O(n)$ size, FIRST-PD takes $O(n)$ time.

There are three for-loops in procedure NEXT-PD on line 30, 31, and 34. The first for-loop on line 30 takes $O(n)$ time. The third for-loop on line 34 is inside the second for-loop on line 31 but will be invoked no more than once because there is a RETURN statement right after it. As a result, the second and third for-loops takes $O(n)$ time in total. Overall speaking, NEXT-PD takes $O(n)$ time.

Finally, we conclude that the time complexity of the algorithm SUM-INDEX is $O(n)$. □

Theorem 7. *The time complexity of* MIN-HEAP-OP *is $O(n)$.*

Proof. The major time overhead of MIN-HEAP-OP lies in the construction of the min-heap $\mathcal{W}_{\mathcal{H}(n)}$ on line 6 and the operation DELETE-MIN on line 10. In the construction of $\mathcal{W}_{\mathcal{H}(n)}$, we only generate its root and store how the other elements are generated instead of generating all the elements in memory. As a result, it only takes $O(1)$ for the min-heap construction. In terms of DELETE-MIN, the time complexity should be logarithmic to the size of the heap. As a result, each DELETE-MIN takes $O(n)$ because the size of $\mathcal{W}_{\mathcal{H}(n)}$ is $2^n - 1$. Note that there are also $O(n)$ key comparisons between any two elements of $\mathcal{W}_{\mathcal{H}(n)}$, in which the subset sums are to be calculated. However, the calculation of subset sum of a child node can be derived from the subset sum of its parent because the difference between them is only one or two elements. To this end, The subset sums of the $O(n)$ subsets can be calculated in $O(n)$ time. In conclusion, the time complexity of the algorithm MIN-HEAP-OP is $O(n)$. ☐

Finally, it comes to the time complexity of the algorithm FAIR-PACK. As shown in Algo. 1, FAIR-PACK iterates the variable m from 1 to infinity until a valid subset (block) is found. Hence, the running time of FAIR-PACK heavily depends on the block validity ratio defined as follows.

Definition 10. *Block Validity Ratio: the possibility for a block to be valid (%).*

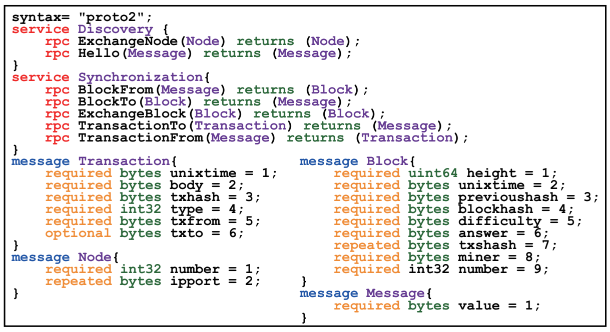
Theorem 8. *Supposing the block validity ratio to be α, the algorithm* FAIR-PACK *terminates in $\frac{\log(1-\beta)}{\log(1-\alpha)} \cdot O(n)$ with a possibility no less than β.*

Proof. The possibility that FAIR-PACK terminates in k rounds is $1-(1-\alpha)^k$, in which each round is a call of SUM-INDEX or MIN-HEAP-OP. Hence, we have $(1-(1-\alpha)^k) \geq \beta$, which leads to $k \geq \frac{\log(1-\beta)}{\log(1-\alpha)}$. Each round of FAIR-PACK takes $O(n)$ time according to Thm. 6 and Thm. 7. Finally, FAIR-PACK terminates in $\frac{\log(1-\beta)}{\log(1-\alpha)} \cdot O(n)$ with a possibility no less than β, which completes the proof. ☐

For example, if the block validity ratio is 0.5%, FAIR-PACK will terminate in around $460 \cdot O(n)$, $597 \cdot O(n)$, and $919 \cdot O(n)$ with possibilities of 90%, 95%, and 99%, respectively.

VII. PERFORMANCE EVALUATION

In this section, we conduct extensive experiments to evaluate the performance of FAIR-PACK.



```
syntax= "proto2";
service Discovery {
    rpc ExchangeNode(Node) returns (Node);
    rpc Hello(Message) returns (Message);
}
service Synchronization{
    rpc BlockFrom(Message) returns (Block);
    rpc BlockTo(Block) returns (Message);
    rpc ExchangeBlock(Block) returns (Block);
    rpc TransactionTo(Transaction) returns (Message);
    rpc TransactionFrom(Message) returns (Transaction);
}
message Transaction{                    message Block{
    required bytes unixtime = 1;            required uint64 height = 1;
    required bytes body = 2;                required bytes unixtime = 2;
    required bytes txhash = 3;              required bytes previoushash = 3;
    required int32 type = 4;                required bytes blockhash = 4;
    required bytes txfrom = 5;              required bytes difficulty = 5;
    optional bytes txto = 6;                required bytes answer = 6;
}                                           repeated bytes txshash = 7;
message Node{                               required bytes miner = 8;
    required int32 number = 1;              required int32 number = 9;
    repeated bytes ipport = 2;          }
}                                       message Message{
                                            required bytes value = 1;
                                        }
```

Fig. 4. Protocol buffers of the blockchain prototype

First, we developed a proof-of-concept blockchain prototype using around 1070-line python code based on gRPC. Fig. 4 shows the communication interfaces among blockchain nodes. In the prototype, two services are implemented to support the blockchain runtime, i.e., peer discovery ("Discovery") and data synchronization ("Synchronization"). The "Discovery" service is used for discovering the nodes inside the blockchain network. When a node is started, it will greet several static nodes (the same as bootnodes in Ethereum) and exchange the connectivity information. The block and transaction synchronization is achieved by the "Synchronization" service, which consists of five remote procedure calls. One thing in particular is that Proof of Work (PoW) serves as the consensus protocol of the blockchain prototype.

Furthermore, three transaction packing algorithms, i.e., FAIR-PACK, FAIR-FIRST [16], and RANDOM-PACK are implemented with around 510-line C++ code. The three packing algorithms are integrated into the blockchain prototype with the help of *ctypes*, using which the packing algorithms are compiled as dynamic link libraries and can be called in python programs.

Finally, we deploy the blockchain prototype together with the three packing algorithms on Amazon Web Services with up to 60 Elastic Compute Cloud (EC2) instances. The 60 C4.LARGE EC2 instances constitute 120 nodes, in which each instance with 2 vCPUs and 3.75GB RAM is shared by two nodes.

The Bitcoin data in year 2012, whose size is around 804 megabytes containing around 1.9 million transactions, is used as the input for the permissioned blockchain.

The performance metrics are the fairness and average response time as discussed in problem definition. The performances of the packing algorithms can be affected by the transaction incoming rate, block generation time, block size, and block validity ratio. We study how the three factors influence the performance of the three packing algorithms. In particular, transaction incoming rate, block size, and block validity ratio are tuned by direct parameter setting, while block generation time is tuned by varying the PoW difficulty. The experiment runs for 5 minutes and 100 times for each parameter setting, e.g., transaction incoming rate as $600tx/s$, blockchain generation time as $5.0s$, block size as $3000tx/bk$, and block validity ratio to be 0.5%. Fig. 5 presents the results.

A. Influence of Transaction Incoming Rate

As the transaction incoming rate increases, there will be more transactions in memory pool when generating a block. Moreover, the backlog of memory pool will increase if transaction incoming rate is larger than transaction processing speed. We study the influence of the transaction incoming rate with results shown in Fig. 5 (a) and (b). Particularly, we vary the transaction incoming rate from $100tx/s$ to $1000tx/s$ with a step of $50tx/s$ and fix the block generation time, block size, and block validity ratio to be $5.0s$, $3000tx/bk$, and 0.5%, respectively.

The transaction incoming rate, if less than $600tx/s$, will be less than or equal to the transaction processing speed, which is calculated to be $\frac{3000tx/bk}{5.0s/bk} = 600tx/s$. On this circumstance, the average response time only slightly increases as the transaction incoming rate increases for all the three transaction packing algorithms. This is because there is nearly no backlog of the memory pool. In terms of fairness, all the three algorithms perform better with the increase of the transaction incoming rate. The reason behind it is that the increasing number of transactions decreases response time differences among the transactions.

When the transaction incoming rate is over $600tx/s$, the backlog of the memory pool will increase as time passes

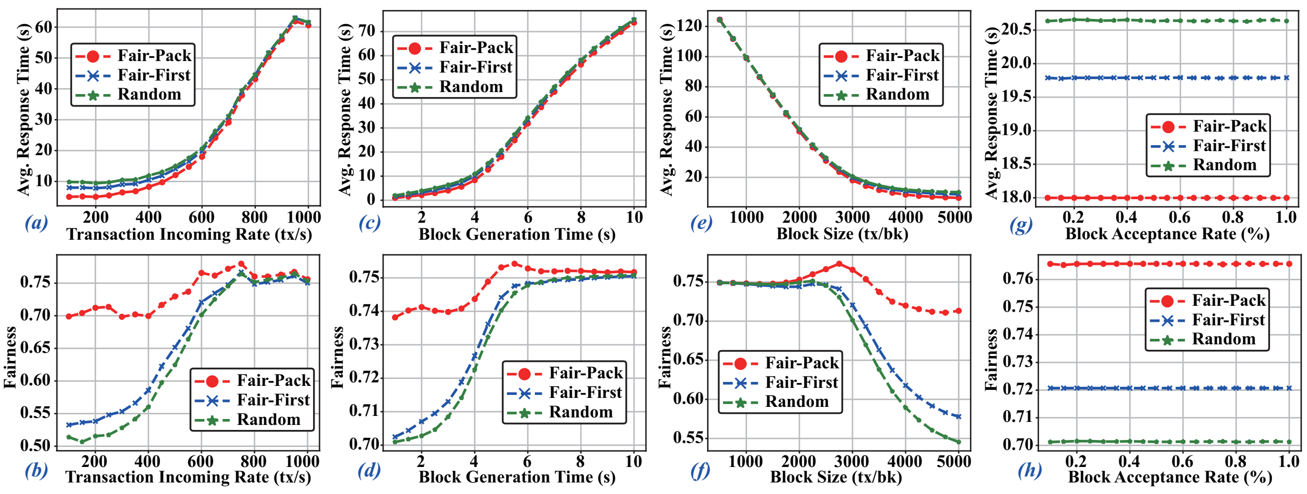


Fig. 5. Experimental result

because the transaction processing speed is less than the transaction incoming rate. In this case, more and more transactions remain unpacked in the memory pool, which increases the average response time regardless of the packing algorithms. However, the fairness only fluctuates and even increases. This is because all the transactions in the blockchain incur long response times and the deviation among the response times of the transactions will be smaller.

Overall speaking, all three transaction packing algorithms are influenced by the transaction incoming rate. With different transaction incoming rates, the response time using FAIR-PACK is slightly better than one using the other two algorithms. Moreover, FAIR-PACK can achieve fairness of 0.70 when the transaction incoming rate is no more than $1000tx/s$ while the fairness using FAIR-FIRST and RANDOM are unsatisfactory, i.e., up to 0.54 and 0.52, respectively.

B. *Influence of Block Generation Time*

In this subsection, we study how the performances of the three algorithms are affected by the block generation time. The results in terms of the average transaction response time and the fairness are shown in Fig. 5 (c) and (d), respectively. We vary the block generation time from $1.0s$ to $10.0s$ with a step of $0.5s$. Nonetheless, the transaction incoming rate, the block size, and the block validity ratio are fixed to be $600tx/s$, $3000tx/bk$, and 0.5%, respectively.

The average response time increases with the increase of the block generation time whatever the transaction packing algorithm is. A short block generation time decreases the waiting times of the transactions and increases the possibility for the transactions to be packed. Moreover, Fig. 1 (c) indicates that the average response time increases remarkably when the block generation time is over $5.0s$, which is the time when the transaction incoming rate is higher than the transaction processing speed. On such circumstances, transactions will be stacked in the memory pool and remain unpacked for a long time. In general, the three algorithms achieve similar average response time regardless of the block generation time.

In terms of fairness, our algorithm outperforms the other two algorithms remarkably when the block generation time is no more than $5.0s$. The number of transactions in the memory pool will be smaller than the block size when the block generation time is less than $5.0s$. In this case, our algorithm FAIR-PACK will employ MIN-HEAP-OP as the underlying transaction selection algorithm, which achieves larger fairness. When the block generation time is over $5.0s$, FAIR-PACK still outperforms two other algorithms although with degraded advantages. The reason is that the heuristic algorithm SUM-INDEX is employed for most of the time on this circumstance.

C. *Influence of Block Size*

Block size is another significant factor influencing the performance of the transaction packing algorithms. In the setting, we vary the block size from $500tx/bk$ to $5000tx/bk$ with a step of $250tx/bk$ and set the transaction incoming rate, block generation time, and block validity ratio to be $600tx/s$, $5.0s$, and 0.5%, respectively. The results of average response time and fairness are shown in Fig. 5 (e) and (f), respectively.

At first glance, Fig. 5 (e) and (f) are nearly symmetric with Fig. 5 (c) and (d), respectively. Indeed, the influence of large block size is similar to the effect of a short block generation time. The distinct difference lies in the scale of the y-axis. For example, the average response time can be as least as $1.5s$ when the block generation time is $1s$. However, the best average response time is up to $9s$ when the block size is $5000tx/bk$. The reason is that the average response time depends much on the block generation time, which is fixed to be $6s$ in this subsection.

In Fig. 5 (f), the fairness among transactions using FAIR-FIRST and RANDOM can be as least as 0.58 and 0.55 when the block size is $5000tx/bk$. This is because the deviation of waiting times of the transactions can be vast with large block size. However, our algorithm FAIR-PACK remains effective on this circumstance, which results from the theoretically optimal algorithm MIN-HEAP-OP.

D. Influence of Block Validity Ratio

Finally, we study the influence of the block validity ratio. In terms of block validity ratio, it naturally comes to our minds that a low block validity ratio can lead to the invalidity of all the blocks containing transactions with large waiting times. Then, a small number of transactions with long waiting times result in long response times of a small set of transactions, substantial deviation of the response times in terms of the full transaction set, and finally poor fairness.

To verify the idea, we conduct experiments in which the transaction incoming rate, the block generation time, and the block size are fixed to be $600tx/s$, $5.0s$, and $3000tx/bk$, respectively and the block validity ratio varies from 0.1% to 1.0% with a step of 0.05%. However, neither the fairness nor the average response time is distinctly affected by the block validity ratio as shown in Fig. 5. The reason is that the validity of a block is random and can not be set deliberately. As a result, we can pack a transaction with long waiting time as long as a block containing it is valid.

In terms of fairness, FAIR-PACK, FAIR-FIRST, and RANDOM achieves fairness of 0.765, 0.720, and 0.701, respectively. Note that, an improvement to 0.765 from 0.701 or 0.720 is significant since the value of fairness only varies from 0 (exclusive) to 1 (inclusive), and a fairness of 0.700 is trivial to achieve by random packing. The average response times using FAIR-PACK, FAIR-FIRST, and RANDOM are around $18.00s$, $19.75s$, and $20.65s$, respectively. That is, FAIR-PACK reduces the average response time of FAIR-FIRST and RANDOM by 8.9% and 12.8%, respectively.

VIII. Related Work

The existing works related to blockchain fairness can be classified into three categories, i.e., fairness among service providers, between service providers and requesters, and among service requesters.

In terms of fairness among service providers, a service provider contributing a certain proportion of resources is supposed to gain the same portion of rewards in fair blockchains. The research communities have studied such fairness in permissionless blockchains, in which the resource is computational resource and the rewards refer to monetary rewards. It is shown that the Bitcoin mining protocol is not incentive compatible and an attack can make the miners' revenue larger than their fair share [17]. In [15], the authors only consider the rewards of the miner contributing the largest computational resource and propose Bitcoin-NG, which improves such fairness. In [18], the authors consider approximate fairness among all the miners and propose FruitChains with theoretical analysis.

A service provider is supposed to receive some rewards if the service requester enjoys its service, which is fairness between the service providers and requesters. In traditional systems, the rewards are transferred with the help of a trustworthy third-party. The smart contract in blockchains provides great potential to enhance such fairness since it removes the third party and the transactions are automatically executed. In [19], the authors solve the problem that malicious contractual parties may prematurely abort from a protocol to avoid financial payment. In [20], the authors explore the solution space for enabling the fair exchange of a cryptocurrency payment for a receipt. The fairness between cloud service providers and requesters are investigated in [21] and [22].

The fairness among the service requesters is insufficiently explored in blockchain. In permissionless blockchains, the service requesters are supposed to pay transaction fees in order to make their transactions confirmed [23]. As a result, the transactions with high transaction fees are more likely to be confirmed earlier, which achieves general fairness although not quantified. There is no native cryptocurrency to be paid as transaction fee in permissioned blockchains. Hence, fairness is not defined or investigated. In [16], the fairness problem in permissioned blockchains was first studied and FAIR-FIRST was proposed. However, it lacks theoretical analysis, and the performance is not satisfactory.

IX. Conclusion

This paper presents FAIR-PACK, the first fairness-based transaction packing algorithm for permissioned blockchain empowered IIoT systems. In particular, we formally define the fairness problem and transform it into the problem of subset sum through a proof of the correlation between the fairness and the subset sum of the transaction waiting times. Then, a heuristic algorithm and a min-heap-based optimal algorithm are proposed to solve the subset sum problem for different parameter settings. The proof and the two algorithms contribute to FAIR-PACK, a fairness-based transaction packing algorithm for permissioned blockchain empowered IIoT systems. Extensive experimental results have articulated the advantages of FAIR-PACK over prior packing algorithms in terms of both fairness and average response time.

Acknowledgment

This research is supported by Hong Kong RGC Research Impact Fund (RIF) with project number R5034-18.

References

[1] H. Subramanian, "Decentralized blockchain-based electronic marketplaces." *Commun. ACM*, 2018.

[2] C. Chen, T. Xiao, T. Qiu, N. Lv, and Q. Pei, "Smart-contract-based economical platooning in blockchain-enabled urban internet of vehicles," *IEEE Trans. on Ind. Informatics*, 2019.

[3] Y. Wang, Z. Su, and N. Zhang, "Bsis: Blockchain-based secure incentive scheme for energy delivery in vehicular energy network," *IEEE Trans. on Ind. Informatics*, 2019.

[4] S. Zou, J. Xi, H. Wang, and G. Xu, "Crowdblps: A blockchain-based location-privacy-preserving mobile crowdsensing system," *IEEE Trans. on Ind. Informatics*, 2019.

[5] J. Xu, S. Wang, B. K. Bhargava, and F. Yang, "A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing," *IEEE Trans. on Ind. Informatics*, 2019.

[6] M. Herlihy, "Blockchains from a distributed computing perspective," *Commun. ACM*, 2019.

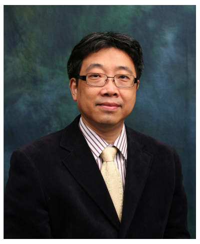
[7] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017.

[8] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *IEEE Trans. on Computers*, 2019.

[9] L. Xu, W. Yin, X. Zhang, and Y. Yang, "Fairness-aware throughput maximization over cognitive heterogeneous NOMA networks for industrial cognitive iot," *IEEE Trans. on Commun.*, 2020.

[10] M. Li, D. Hu, C. Lal, M. Conti, and Z. Zhang, “Blockchain-enabled secure energy trading with verifiable fairness in industrial internet of things,” *IEEE Trans. on Ind. Informatics*, 2020.
[11] J. Leng, D. Yan, Q. Liu, K. Xu, J. L. Zhao, R. Shi, L. Wei, D. Zhang, and X. Chen, “Manuchain: Combining permissioned blockchain with a holistic optimization model as bi-level intelligence for smart manufacturing,” *IEEE Trans. on Syst. Man Cybern. Syst.*, 2020.
[12] U. Schwiegelshohn and R. Yahyapour, “Analysis of first-come-first-serve parallel job scheduling,” in *ACM SODA*, 1998.
[13] R. Jain, D.-M. Chiu, and W. R. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer system.* Eastern Research Lab, 1984.
[14] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A provably secure proof-of-stake blockchain protocol,” in *Annual International Cryptology Conference*. Springer, 2017.
[15] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, “Bitcoin-ng: A scalable blockchain protocol,” in *USENIX NSDI*, 2016.
[16] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, and J. He, “Blochie: a blockchain-based platform for healthcare information exchange,” in *IEEE SMARTCOMP*, 2018.
[17] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” *Commun. ACM*, 2018.
[18] R. Pass and E. Shi, “Fruitchains: A fair blockchain,” in *ACM PODC*, 2017.
[19] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *IEEE Symposium on Security and Privacy*, 2016.
[20] J. Liu, W. Li, G. O. Karame, and N. Asokan, “Toward fairness of cryptocurrency payments,” *IEEE Secur. Priv.*, 2018.
[21] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, “Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization,” in *IEEE INFOCOM*, 2018.
[22] Y. Zhang, R. Deng, X. Liu, and D. Zheng, “Outsourcing service fair payment based on blockchain and its applications in cloud computing,” *IEEE Trans. on Serv. Comput.*, 2018.
[23] Ö. Gürcan, A. Del Pozzo, and S. Tucci-Piergiovanni, “On the bitcoin limitations to deliver fairness to users,” in *Springer OTM*, 2017.

Shan Jiang received the B.Sc. degree in computer science and technology from Sun Yat-sen University, Guangzhou, China in 2015. He is currently a Ph.D. student with Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. He was a research assistant with The Hong Kong Polytechnic University from April 2015 to June 2016 and a visiting student with Imperial College London from November 2018 to March 2019. His research interests include distributed systems, blockchain, and multi-robot systems.



Jiannong Cao (M’93-SM’05-F’15) received the B.Sc. degree in computer science from Nanjing University, Nanjing, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, WA, USA, in 1986 and 1990, respectively. He is currently the Otto Poon Charitable Foundation Professor in Data Science and the Chair Professor of Distributed and Mobile Computing in the Department of Computing at The Hong Kong Polytechnic University, Hong Kong. He is also the director of the Internet and Mobile Computing Lab (IMCL) in the department and the associate director of University’s Research Facility in Big Data Analytics (UBDA). His research interests include parallel and distributed computing, wireless networking and mobile computing, big data and machine learning, and cloud and edge computing.

Hanqing Wu is currently a Ph.D. candidate with Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. He received the B.Sc. degree in software engineering from Tongji University in 2010. His research interests include distributed computing, blockchain, and big data.

Yanni Yang is currently a Ph.D. candidate in the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China. Before that, she received the B.E. and M.Sc. degrees from the Ocean University of China in Qingdao, in 2014 and 2017, respectively. She was a visiting student in the Media Lab of MIT in 2019. Her research interests include wireless human sensing, pervasive and mobile computing, and Internet of things. She has published paper in many conferences and journals, e.g., Ubicomp, SECON, IEEE Internet of Things Journal.