Utility-aware Payment Channel Network Rebalance

Wangze Ni
HKUST
Hong Kong SAR, China
wniab@cse.ust.hk

Pengze Chen
HKUST
Hong Kong SAR, China
pchenax@cse.ust.hk

Lei Chen
HKUST (GZ) & HKUST
Guangzhou & Hong Kong SAR, China
leichen@cse.ust.hk

Peng Cheng
ECNU
Shanghai, China
pcheng@sei.ecnu.edu.cn

Chen Jason Zhang
PolyU
Hong Kong SAR, China
jason-c.zhang@polyu.edu.hk

Xuemin Lin
Shanghai Jiaotong Univeristy
Shanghai, China
xuemin.lin@gmail.com

ABSTRACT

The payment channel network (PCN) is a promising solution to increase the throughput of blockchains. However, unidirectional transactions can deplete a user's deposits in a payment channel (PC), reducing the success ratio of transactions (SRoT). To address this depletion issue, rebalance protocols are used to shift tokens from well-deposited PCs to under-deposited PCs. To improve SRoT, it is beneficial to increase the balance of a PC with a lower balance and a higher weight (i.e., more transaction executions rely on the PC). In this paper, we define the utility of a transaction and the utility-aware rebalance (UAR) problem. The utility of a transaction is proportional to the weight of the PC and the amount of the transaction, and inversely proportional to the balance of the receiver. To maximize the effect of improving SRoT, UAR aims to find a set of transactions with maximized utilities, satisfying the budget and conservation constraints. The budget constraint limits the number of tokens shifted in a PC. The conservation constraint requires that the number of tokens each user sends equals the number of tokens received. We prove that UAR is NP-hard and cannot be approximately solved with a constant ratio. Thus, we propose two heuristic algorithms, namely Circuit Greedy and UAR_DC. Extensive experiments show that our approaches outperform the existing approach by at least 3.16 times in terms of utilities.

PVLDB Reference Format:
Wangze Ni, Pengze Chen, Lei Chen, Peng Cheng, Chen Jason Zhang, and Xuemin Lin. Utility-aware Payment Channel Network Rebalance. PVLDB, 17(2): 184 - 196, 2023.
doi:10.14778/3626292.3626301

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HelloGreatWorld/UtilityRebalance.

1 INTRODUCTION

The payment channel network (PCN) is a leading approach to enhancing the throughput of blockchains [34]. A PCN consists of off-chain bidirectional payment channels (PCs), where the number of tokens in a PC is limited. However, depletion may happen, causing tokens to accumulate in higher flow directions and depleting some PCs [19]. For example, in the lightning network (LN), 63% of PCs lose 80% of the balance in one direction over time, which decreases the success ratio of transactions (SRoT) from 71% to 29% [4].

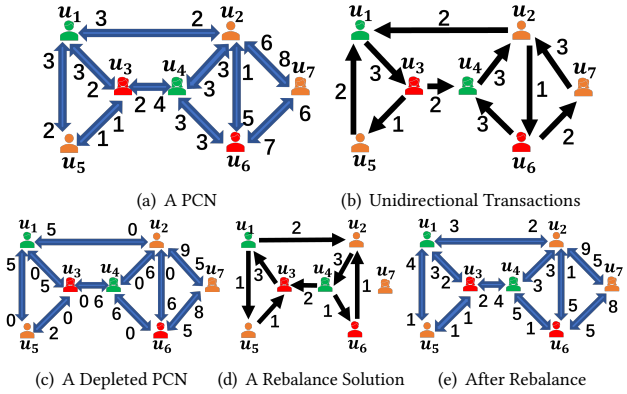
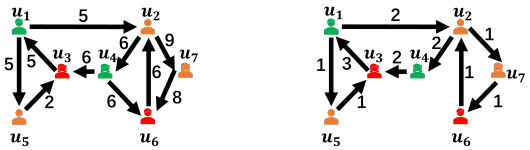


Figure 1: An Example of a PCN.

Figure 1(a) shows a PCN. For example, in the PC between u_1 and u_3, u_1 deposits 3 tokens, and u_3 deposits 2 tokens. Denote the direction from i to j in a PC as $p_{i,j}$. Although u_1 has no direct PC with u_6, u_1 can transfer at most 2 tokens to u_6 via the route $p_{1,3} \rightarrow p_{3,4} \rightarrow p_{4,6}$, as the minimum balance of a PC in the route is 2. Figure 1(b) shows some unidirectional transactions, and Figure 1(c) shows the PCN after executing these transactions. For example, after transferring 3 tokens from u_1 to u_3, the balance of u_1 in $p_{1,3}$ is 0. As most PCs have depleted balances in one direction, the SRoT of the PCN in Figure 1(c) decreases. For example, suppose tx wants to transfer 1 token from u_2 to u_5 via the shortest route (i.e., $p_{2,5} \rightarrow p_{1,5}$). Since u_2 has no token in $p_{2,1}$, tx fails.

Researchers have proposed some solutions to tackle the depletion issues. One promising way is to *rebalance* PCs by shifting tokens between PCs [4, 19]. A user sends tokens in the PC where she has abundant tokens and receives tokens in the PC where she lacks tokens. In a rebalance solution (RS), the sum of tokens received by a user across all PCs equals the sum of tokens she sends (i.e., the *conservation* constraint); otherwise, users who lose tokens will refuse to execute the RS. Moreover, the number of tokens shifted in a PC cannot exceed the *budget* proposed by the user (i.e., the *budget* constraint). Figure 1(d) shows an RS. For instance, u_3 receives 2 tokens in $p_{4,3}$ and 1 token in $p_{5,3}$, and sends 3 tokens in $p_{3,1}$.

Proceedings of the VLDB Endowment, Vol. 17, No. 2 ISSN 2150-8097.
doi:10.14778/3626292.3626301



(a) Solution rs_1 (b) Solution rs_2

Figure 2: Motivation Example.

Figure 1(e) shows the PCN after executing the RS in Figure 1(d). Now, since $p_{2,1}$ has 2 tokens, tx can be executed. In other words, rebalance can increase the SRoT.

Researchers propose an exact approach, *Revive*, to finding an RS shifting maximal tokens [19]. However, Revive ignores the differences between PCs. In a PCN, some PCs are frequently used to execute transactions, termed *"main" PCs*, while some PCs are rarely used, termed *"edge" PCs*. In other words, main PCs have higher weights than edge PCs in a PCN. In Figure 1(a), $p_{3,4}$ is a main PC used often, while $p_{6,7}$ is an edge PC used rarely. Thus, increasing tokens in main PCs will enable more transactions to be executed than increasing the same amount of tokens in edge PCs. Rebalancing main PCs has a greater effect on improving SRoT [25, 34]. Moreover, PCs with lower balances are more likely to fail in executing transactions due to insufficient balances. Thus, rebalancing PCs with lower balances benefits SRoT more. For example, in Figure 1(c), if we rebalance 1 token in $p_{6,4}$, then we can transfer 1 token from u_6 to u_4; otherwise, we cannot transfer tokens in $p_{6,4}$. However, even if we do not rebalance 1 token in $p_{6,7}$, we can still transfer 1 token from u_6 to u_7. In other words, rebalancing 1 token in $p_{6,4}$ is more beneficial to SRoT than rebalancing 1 token in $p_{6,7}$.

Thus, the transactions rebalancing tokens in different PCs have different utilities, which are defined as $\frac{w \cdot v}{b+1}$, where w is the weight of the PC, v is the number of shifted tokens, and b is the balance of the receiver. A transaction involving a PC with a higher weight and a receiver with a lower balance has a greater impact on improving SRoT and therefore has higher utility. However, it is intractable to find a set of transactions with maximal utilities, satisfying the budget and conservation constraints. Here is an example.

Example 1. *We want to find a valid RS with a maximal utility for the PCN shown in Figure 1(c). The utility $s_{i,j}$ of rebalancing $v_{i,j}$ tokens in $p_{i,j}$ is $s_{i,j} = w_{j,i} \cdot \frac{v_{i,j}}{b_{j,i}+1}$, where $b_{j,i}$ is the balance that the receiver j has in $p_{j,i}$, and $w_{j,i}$ is the weight of $p_{j,i}$. Assume the weight of the PC between u_3 and u_4 is 2, while the weights of the other PCs are 1. Additionally, the budget $d_{i,j}$ of j in $p_{i,j}$ is the difference between its balance in Figure 1(c) and the balance in Figure 1(a) (e.g., $d_{5,3} = 1$).*

The first approach makes a solution rs_1 where each user sends all tokens in each PC, shown in Figure 2(a). However, rs_1 violates the conservation constraint. For instance, A receives 5 tokens but sends 10 tokens. Moreover, rs_1 violates the budget constraint. For example, A transfers 5 tokens in $p_{1,2}$ while the budget is only 2 tokens. Thus, although rs_1 totally shifts 58 tokens, rs_1 is not valid.

Figure 2(b) shows an RS made by the second approach [19], which aims to maximize the number of shifted tokens. In this RS, $s_{4,3} = 2 \cdot \frac{2}{1} = 4$. Similarly, $s_{3,1} = 3$, $s_{1,5} = s_{5,3} = s_{6,2} = 1$, $s_{1,2} = s_{2,4} = 2$, and $s_{2,7} = s_{7,6} = \frac{1}{6}$. Thus, the utility of this RS is $\frac{43}{3}$.

Figure 1(d) shows an RS with a maximal utility. In this RS, $s_{4,3} = 4$, $s_{1,2} = 2$, $s_{2,4} = s_{3,1} = 3$, $s_{1,5} = s_{5,3} = s_{4,6} = s_{6,2} = 1$. Thus, the utility of this RS is 16, larger than the utility of the RS in Figure 2(b).

Example 1 shows that the existing method fails to find an RS with a maximal utility, and that there are an exponential number of solutions to consider. Thus, it is crucial to study how to efficiently seek an RS with a maximal utility. In this paper, we first formally define the utility-aware rebalance (UAR) problem. Given a PCN, UAR aims to find an RS with a maximal utility, satisfying the *conservation* and *budget* constraints. The conservation constraint requires that the number of tokens a user sends equals the number of tokens received. The budget constraint requires that the number of tokens a user sends in a PC does not exceed her budget. Like shown in Example 1, UAR aims to find a solution shown in Figure 1(d). We theoretically prove that UAR is NP-hard, and no constant factor approximation algorithm can solve UAR. Thus, we propose two heuristic algorithms, namely *Circuit Greedy* and *UAR_DC*. The RS obtained by Circuit Greedy has a higher utility, whereas UAR_DC runs much faster. Extensive experiments on real and synthetic datasets show that our approaches outperform the existing approach by 5 times in utilities. In summary, we have made the following contributions:

- In § 3, we formulate the utility-aware rebalance problem. We prove that UAR is NP-hard and that there is no constant factor approximation algorithm for UAR in general;
- In §4, we propose an effective algorithm, namely Circuit Greedy;
- In § 5, we propose an efficient algorithm, namely UAR_DC; and
- We conduct comprehensive experimental studies over real and synthetic datasets in § 6.

In addition, we introduce preliminaries in Section 2, discuss the related works in Section 7, and conclude our work in Section 8. Due to space limitations, we omit the proofs of some theorems in this paper, for more details please refer to our technical report [2].

2 PRELIMINARIES

In this section, we first introduce the preliminaries of PC and PCN in § 2.1 and § 2.2, respectively. Next, we introduce the preliminaries of rebalance and outline a famous protocol, Revive, in § 2.3.

2.1 Payment Channel

The development of blockchains suffers from notoriously low scalability [7, 41]. The root of the limited scalability is that each transaction requires a global consensus from a large number of users [8, 12]. Therefore, a promising way to improve scalability is that each transaction only needs to be locally confirmed by the relevant participants [34]. This idea inspires the design of the PC.

Two users first propose a transaction on the blockchain to open a PC with some deposited tokens. Then, the two users can transact off the blockchain to transfer deposited tokens between each other in the PC. A transaction is confirmed by the two users in the PC by a signed commitment. A commitment represents the updated allocation of the deposited tokens between the two users, namely a *state* of a PC. Either one of the two users can propose a transaction with the latest signed commitment on the blockchain to close the PC and redeem their tokens from the PC. Two users can execute arbitrary transactions in a PC while burdening the blockchain with

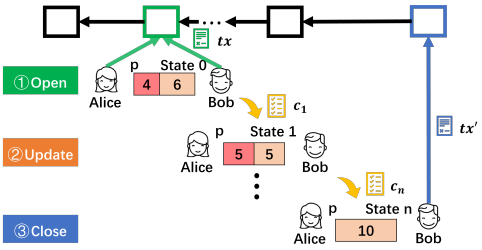


Figure 3: An Example of a Payment Channel.

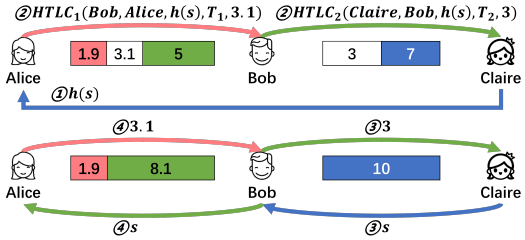


Figure 4: A 2-hop Transaction.

merely two transactions, one for opening and another for closing. Moreover, since users only can use the deposited tokens in PCs, the transactions in PCs are independent and can be executed in parallel. Thus, the scalability is enhanced significantly.

Figure 3 shows an example. Alice and Bob first propose an on-chain transaction tx on the blockchain to open a PC p. In tx, Alice deposits 4 tokens, and Bob deposits 6 tokens. Thus, the initial state of p is State 0, where Alice has 4 tokens, and Bob has 6 tokens. Then, Alice and Bob repeatedly transfer tokens in p without consensus on the blockchain. For example, Alice and Bob sign a commitment c_1 to update the state of p from State 0 to State 1 (i.e., Bob transfers 1 token to Alice). After several transactions, the balance of Alice in p is 0. Then, Alice can no longer transfer tokens to Bob in p. Thus, Bob proposes an on-chain transaction tx' with the latest commitment c_n on the blockchain to redeem his 10 tokens.

2.2 Payment Channel Network

A PCN is a collection of PCs. As introduced in § 2.1, opening a PC requires depositing some tokens. Thus, it is economically infeasible for a user to open a PC with every user, especially for users who transact infrequently. However, two users with no PC can transact by a route of linked PCs. The transaction from a sender to a receiver via a route with $x-1$ intermediates is named an *x-hop transaction*.

A sender us first finds a route τ of linked PCs to a receiver ur. Then, in τ, a user makes a transaction that would move tokens to its successor in a PC, conditioned on a secret s that only the receiver ur knows. Specifically, a transaction is executed by a hashed time-locked contract (HTLC) [34]. An HTLC $HTLC(\alpha, \beta, h, t, v)$ states that if a user α can propose a number whose hash value is h before the timestamp t, β transfers v tokens to α. Thus, the receiver ur can reveal the secret s to its predecessor in τ to obtain tokens. Then, sequentially, in τ, each user reveals s to its predecessor to obtain tokens. In other words, the secret propagates back through the route, which updates all PCs' balances and eventually achieves that the sender transacts the receiver. If an intermediate user u transfers tokens to her successor, u knows the secret and can obtain tokens from her predecessor. Thus, no intermediate user loses tokens. Moreover, to incentivize intermediate users to make hop transactions, existing designs allow intermediate users to charge transaction fees for forwarding a hop transaction [13, 34]. Usually, making a hop transaction is much cheaper than creating a PC between two users.

Figure 4 shows an example where Alice makes a 2-hop transaction to transfer 3 tokens to Claire. Claire first selects a secret s and informs Alice of s's hash value. Then, Alice finds a route τ, Alice $\rightarrow$ Bob $\rightarrow$ Claire. In τ, each user makes an HTLC with her subsequent user. Specifically, Claire obtains 3 tokens from Bob by revealing s, but Bob obtains 3.1 tokens from Alice by revealing s. The difference (i.e., 0.1 token) is the transaction fees that Bob charges.

Remark 2.1. Research has revealed that a small number of well-funded nodes with numerous PCs dominate the PCN, processing a significant proportion of transactions [21, 40]. The PCs that are used for vast transactions' execution are referred to as "main" PCs. The likelihood of a single node becoming the bottleneck in a PCN depends on various factors, like the topology of the PCN. For example, in a PCN with a decentralized topology, where transactions are spread out across multiple nodes, the likelihood of a single node becoming a bottleneck is low. This is because the transactions are distributed evenly across the network, reducing the load on any one node. Research has shown that removing 200 nodes with numerous channels from a real-world PCN with 2000 nodes decreases the SRoT from 67.5% to approximately 10% [32]. Thus, in real-world applications, the likelihood that a single node becoming the bottleneck in a PCN is low. When these well-funded nodes are non-malicious, a centralized topology decreases the delay of transactions and increases the SRoT of a PCN [15]. However, adversaries can deplete multiple "main" PCs simultaneously, decreasing the SRoT and increasing transaction delays and fees [26, 32]. These depletion attacks [26, 32] deplete the tokens of "main" PCs, leaving them starved of tokens to execute normal users' transactions. Thus, it is significant to let "main" PCs have sufficient tokens.

2.3 Rebalance

As introduced in § 2.1, if a user has depleted her tokens in a PC, she cannot make transactions in this PC. Then, the user has to close the depleted PC and open a new PC, which costs two expensive on-chain transactions. Instead of reopening a new PC, if a user has other PCs with tokens, she can "shift" the tokens in these PCs. For instance, if a user us has no tokens in a PC p with user ur but has tokens in another PC p' that is included in a route τ linking us and ur, then us can transfer v tokens to ur via τ and ask ur to transfer v tokens to us in p. In this way, a user can shift tokens in a PC p' to another PC p to rebalance the allocation of tokens in p.

Revive [19] is the first protocol to help multiple users to rebalance their PCs simultaneously. Briefly, users first propose requests to an elected leader, which include the PCs they want to rebalance and the budgets of rebalanced tokens. Then, the leader generates a rebalance solution (RS) that consists of a set of transactions where the sum of tokens each user sends equals the sum of tokens received, and the sum of tokens sent does not exceed the stated budget. Next, the leader informs users of the RS. Finally, users execute transactions to rebalance the tokens in their PCs according to the RS. Compared

with reopening PCs, Revive can save at least 50% of transaction fees [19]. Revive has been widely implemented and studied [10, 11, 16, 23, 34, 38]. This paper uses Revive as an example to study how to generate an RS, without modifying the protocol itself. Our approaches can be extended to other rebalance protocols [18].

Remark 2.2. Adversaries may participate in rebalance protocols and perform some malicious behaviors. In an RS, the sum of tokens a user sends equals the sum of tokens received. Thus, adversaries cannot steal other users' tokens. Moreover, transactions are executed by the users themselves. Thus, users can refuse to execute the transactions and select another leader if a malicious leader makes an invalid RS. In other words, *rebalance protocols are secure*. However, adversaries can halt protocols. For example, they can refuse to execute transactions in RSs. Existing protocols tackle malicious issues with some practical solutions. For example, decreasing adversaries' reputations [19], and locking adversaries' tokens [18]. After punishing adversaries, existing protocols will start from scratch to find a new RS for non-malicious users.

3 PROBLEM DEFINITION

In this section, we first define some basic concepts in § 3.1. Then, we define the utility-aware rebalance (UAR) problem in § 3.2. Finally, in § 3.3, we theoretically analyze the hardness of UAR. We prove that UAR is NP-hard, and there is no polynomial-time algorithm with a constant approximation ratio for UAR unless P = NP.

3.1 Basic Concepts

We first formally define the concept of a payment channel network.

Definition 1. A payment channel network is denoted by $G = \langle S, P \rangle$, where $S = \{u_1, u_2, \cdots, u_n\}$ is a set of users, and $P = \{p_{1,2}, p_{1,3}, \cdots, p_{n,n-1}\}$ is a set of payment channels. A payment channel is denoted by $p_{i,j} = \langle w_{i,j}, d_{i,j}, b_{i,j} \rangle$, where $w_{i,j}$ is the weight of $p_{i,j}$, budget $d_{i,j}$ is the maximum of tokens that u_i wants to rebalance in $p_{i,j}$, and $b_{i,j}$ is the balance that the sender u_i has in $p_{i,j}$.

In real-world applications, the trading demands of two users in a PC are usually different [22, 34]. Thus, the number of tokens a user needs may not be half of the tokens in a PC. Without loss of generality, in this paper, we allow users to set their budgets $d_{i,j}$. This is realistic and can be achieved in rebalance protocols [19]. Moreover, some tools [1, 22] can help users estimate their budgets. In a PCN, some *"main" PCs* are critical and are frequently used in hop transactions [24, 39]. However, some "edge" PCs are infrequently used. To increase the SRoT, it is more significant to have more tokens in main PCs than in edge PCs [25, 34]. Thus, we should give higher priority $w_{i,j}$ to critical PCs in rebalance. The weight of a PC is determined by the leader in the rebalance protocol (introduced in § 2.3). Some tools can help estimate the weight of an edge in a graph [6, 31]. In Example 1, $d_{1,2} = 2$, and $w_{3,4} = 2$. Users can state their balances $b_{i,j}$ when they propose their requests. Besides, we can probe the balances in PCs by some methods [17, 36]. Then, we formally define the concept of a rebalance solution.

Definition 2. A rebalance solution, denoted by $RS = \{tx_{1,2}, tx_{1,3}, \cdots, tx_{n,n-1}\}$, is a set of transactions. A transaction $tx_{i,j}$ transfers $v_{i,j}$ tokens from u_i to u_j. A valid rebalance solution satisfies

- **The budget constraint.** For any transaction $tx_{i,j}$ in RS, the amount $v_{i,j}$ cannot exceed the budget $d_{i,j}$ that u_i proposes in $p_{i,j}$, i.e., $v_{i,j} \leq d_{i,j}$; and
- **The conservation constraint.** For any user u_i, the sum of tokens she sends in RS equals the sum of tokens she receives in RS, i.e., $\forall u_i \in U$, $\sum_{tx_{i,j} \in RS} v_{i,j} = \sum_{tx_{j,i} \in RS} v_{j,i}$.

The conservation constraint guarantees no user will lose token in an RS. For example, in the RS shown in Figure 2(b), u_2 receives 2 tokens from u_1 and 1 token from u_6. Meanwhile, u_2 sends 2 tokens to u_4 and 1 token to u_7. The budget constraint guarantees that the rebalance solution will not over-rebalance PCs; otherwise, the SRoT may not increase (as shown in Figure 2(a)). In Figure 2(b), $RS = \{tx_{3,1}, tx_{1,5}, tx_{5,3}, tx_{1,2}, tx_{2,4}, tx_{4,3}, tx_{6,2}, tx_{2,7}, tx_{7,6}\}$, and $tx_{3,1}$ transfers $v_{3,1} = 3$ tokens from u_3 to u_1.

3.2 The Utility-aware Rebalance Problem

Based on the aforementioned concepts, we formulate the utility-aware rebalance (UAR) problem as follows:

Definition 3. Given a payment channel network $G = (U, P)$, the utility-aware rebalance problem aims to find a valid rebalance solution RS with a maximum utility $S = \sum_{tx_{i,j} \in RS} w_{j,i} \cdot \frac{v_{i,j}}{b_{j,i}+1}$.

Different PCs contribute differently to hop transactions [22, 34], and rebalancing PCs where the receivers have fewer balances is more important (as shown in Figure 1(d)). Thus, we need to find an RS with a maximal utility. Hence, UAR is realistic and significant.

Remark 3.1 (Privacy issues)**.** Rebalancing protocols in PCNs involve probing the balances in PCs to determine the amount of tokens transferred. Although in existing applications, we can probe the balances [38], probing reveals information about the balances in PCs, which compromises the privacy of the users. To address this privacy concern, researchers have proposed various techniques, such as revealing balances with noise [18, 35], to balance privacy and utility while maintaining the security and efficiency of the PCN. Our approaches are compatible with the techniques that reveal balances with noise [18, 35]. However, our algorithms calculate the utility of a rebalancing transaction using the balance of its receiver. If the receiver's balance is obscured with a random noise number, the utility of a rebalancing transaction may be affected, potentially reducing the improvement effects of our solutions on the SRoT.

Remark 3.2 (The feasibility in large networks without known topology)**.** As introduced in § 2.3, in a rebalance protocol (e.g., Revive), users propose their requests, which form a PCN pcn. The topology of pcn is known, and we conduct rebalancing transactions over pcn. Thus, we can still obtain rebalancing solutions, even if we do not know the topology of the whole PCN pcn^* (i.e., $pcn \subset pcn^*$). However, our methods assign each PC with a weight, which estimates the significance of a PC in the PCN. Thus, if we do not know the topology of pcn^*, we cannot accurately estimate the weight of a PC in pcn^*, although we can still estimate the weight of a PC according to pcn. In other words, we can obtain rebalancing solutions when we do not know the topology of the PCN, while their effectiveness in improvement on the SRoT may decrease. Moreover, we can use routing approaches (e.g., Flash [38]) to probe the topology.

Table 1: Symbols and Descriptions.

Symbol	Description
$G = \langle U, P \rangle$	A PCN
$U = \{u_1, \cdots, u_n\}$	A set of users
$P = \{p_{1,1}, \cdots, p_{n,n-1}\}$	A set of PCs
$b_{i,j}$	the balance that u_i has in $p_{i,j}$
$w_{i,j}$	the weight of a PC $p_{i,j}$
$d_{i,j}$	the budget of a PC $p_{i,j}$
$RS = \{tx_{1,2}, \cdots, tx_{n,n-1}\}$	a rebalance solution
$tx_{i,j}$	a rebalance transaction in a PC $p_{i,j}$
$v_{i,j}$	the amount of $tx_{i,j}$
$S = \sum_{tx_{i,j} \in RS} w_{j,i} \cdot \frac{v_{i,j}}{b_{j,i+1}}$	the utility of a rebalance solution RS

3.3 The Hardness of the UAR Problem

In this subsection, we first prove that UAR is NP-hard. Then, we prove that there is no polynomial algorithm with a constant approximation ratio for UAR. The approximation ratio is the ratio between the result obtained by an algorithm and the optimal solution [37]. We prove the NP-hardness of UAR by reducing it from the Hamilton decomposition problem [29].

Theorem 3.1. *The utility-aware rebalance problem is NP-hard.*

Proof. We first prove that the decision version of the UAR problem, namely D-UAR, is NP-complete by reducing from the Hamilton decomposition (HD) problem [29]. The D-UAR problem aims to find an RS with S_0 utility. The HD problem can be described as follows: Given a graph $HG = \langle E, V \rangle$, where E is a directed edge set, and V is a vertex set, the HD problem aims to partition the set E into a set of subsets, where each subset is a Hamilton circuit.

Given an HD problem instance, we construct a D-UAR problem instance as follows: For each edge in E from a vertex v_i to v_j, we generate a PC $p_{i,j}$, where $d_{i,j} = 1$, $b_{j,i} = 0$, and $w_{j,i} = 1$. Thus, each PC can execute at most one transaction transferring one token. We set S_0 as the number edges in E. Thus, if we can find an RS whose utility is S_0, we can partition E into a set of Hamilton circuits.

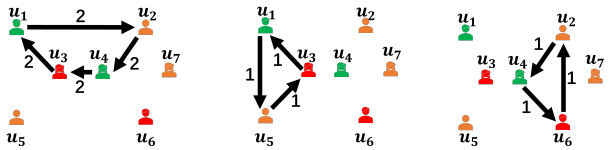
In other words, if we can solve the transformed D-UAR problem instance, we can solve the HD problem instance. Since the HD problem is NP-complete [29], D-UAR is also NP-complete. Since the UAR problem is the optimization version of the D-UAR problem, the UAR problem is NP-hard. Thus, the proof is completed. ■

Theorem 3.1 shows that no polynomial-time algorithm can exactly solve any UAR problem instance unless P = NP. A practical solution for solving NP-hard problems is to propose heuristic algorithms or approximation algorithms that can produce solutions within polynomial time [27, 28]. However, we proved that it is still hard to get an approximate solution to the UAR problem.

Theorem 3.2. *If $P \neq NP$, then for any constant $\rho \leq 1$, there is no polynomial time algorithm with approximation ratio ρ for the UAR problem.*

Proof sketch. We prove the theorem by contradiction. We show that if an algorithm $\mathcal{A}$ can solve any UAR instance with an approximation ratio of ρ, $\mathcal{A}$ can exactly solve HD [29]. ■

Thus, if P ≠ NP, UAR cannot be approximately solved with a constant ratio in polynomial time. Thus, we propose heuristic algorithms for UAR. Table 1 summarizes the commonly used symbols.



(a) 1st RC (b) 2nd RC (c) 3rd RC

Figure 5: The Rebalance Circuits in Figure 1(d).

4 THE CIRCUIT GREEDY ALGORITHM

As proven in Section 3.3, UAR is NP-hard, and no algorithm with a constant approximation ratio can solve UAR in polynomial time unless P = NP. Thus, in this section, we propose an effective heuristic algorithm, namely Circuit Greedy.

In this section, we first introduce the basic idea of Circuit Greedy in § 4.1. Next, we describe the details of Circuit Greedy and show a running example in § 4.2. Finally, we theoretically analyze the time complexity of Circuit Greedy in § 4.3.

4.1 The Basic Idea of Circuit Greedy

Briefly, Circuit Greedy repeatedly selects a *rebalance circuit* (RC) with the highest utility from the PCN until there is no RC in the PCN. An RC is a set of transactions with the same amount, where the PCs executing those transactions is a Hamilton circuit.

Definition 4. A rebalance circuit rc_i is a set of transactions, where the amounts of the transactions are the same, and the PCs executing the transactions form a Hamilton circuit.

For example, in Figure 2(b), $\{tx_{2,7}, tx_{7,6}, tx_{6,2}\}$ is an RC. We term a set of PCs that form a Hamilton circuit as a *PC circuit*. Circuit Greedy is inspired by Theorem 4.1. If an RS satisfies the conservation constraint, it can be decomposed into several RCs.

Theorem 4.1. *If a rebalance solution RS satisfies the conservation constraint, we can get a set of RCs $RC = \{rc_1, rc_2, \cdots, rc_n\}$, where the sum of the amounts of the transactions in each PC $p_{i,j}$ in RC equals the amount of the transaction in $p_{i,j}$ in RS.*

Proof sketch. We prove the theorem by contradiction. If *RS* cannot fully be decomposed by a set of RCs, a user receives more tokens than she sent, which violates the conservation constraint. Thus, the assumption is wrong, and the theorem is proved. ■

For example, the RS in Figure 1(d) can be decomposed into three RCs shown in Figure 5. In the RS shown in Figure 1(d), the amount of $tx_{3,1}$ is 3. The amount of $tx_{3,1}$ in Figure 5(a) is 2, and the amount of $tx_{3,1}$ in Figure 5(b) is 1. In other words, $tx_{3,1}$ in Figure 1(d) is the combination of the $tx_{3,1}$ in Figure 5(a) and Figure 5(b). Thus, we can find a set of RCs to generate an RS.

4.2 The Description of Circuit Greedy

Algorithm 1 shows the pseudo-code of the Circuit Greedy algorithm. Circuit Greedy first initializes the RC set *RC* as empty (line 1). Then, Circuit Greedy repeatedly finds an RC $rc^{\#}$ with the highest utility until no $rc^{\#}$ can be selected (lines 2-12). Specifically, for each PC $p_{i,j}$ in P, Circuit Greedy first uses a depth-first search procedure to find a PC circuit c_k (line 5). Then, Circuit Greedy finds the minimal remaining budget α_k of PCs in c_k (line 6). Next, Circuit Greedy

Algorithm 1: The Circuit Greedy algorithm.

Input: a PCN $G = \langle U, P \rangle$
Output: a rebalance solution RS

1 $RC = \emptyset$;
2 **repeat**
3 $rc^{\#} = \emptyset$;
4 **foreach** $p_{i,j} \in P$ **do**
5 Find a PC circuit c_k by a depth-first search procedure, where $p_{i,j}$ is included;
6 Get the minimal remaining budget α_k of the PCs in c_k;
7 Generate an RC rc_k of c_k where the amount of each transaction is α_k;
8 Calculate the utility of rc_k $\beta_k = \sum_{p_{h,g} \in c_k} w_{h,g} \cdot \frac{\alpha_k}{b_{g,h}+1}$;
9 $rc^{\#} \leftarrow$ the RC with the highest utility, $RC = RC \cup rc^{\#}$;
10 Update the remaining budget of PCs;
11 Delete PCs whose remaining budgets are 0;
12 **until** $rc^{\#} = \emptyset$
13 Combine the RCs in RC to generate an RS RS;
14 **return** RS;

generates an RC rc_k for c_k where the amount of each transaction is α_k (line 7). After that, we calculate the utility β_k of rc_k, i.e., $\beta_k = \sum_{p_{h,g} \in c_k} w_{h,g} \cdot \frac{\alpha_k}{b_{g,h}+1}$ (line 8). Then, Circuit Greedy selects the RC $rc^{\#}$ with the highest utility and adds $rc^{\#}$ into RC (line 9). After that, Circuit Greedy updates the remaining budgets of PCs (line 10) and deletes PCs whose remaining budgets are 0 (line 11). Finally, Circuit Greedy combines the RCs in RC to generate an RS RS (line 13) and returns RS (line 14). Here is a running example.

Example 2. *We use Circuit Greedy to get a solution to the UAR problem instance in Example 1. Table 2 shows the PC circuits obtained for each PC. For instance, although $p_{6,2}$ is involved in two PC circuits, $\{u_2 \rightarrow u_4 \rightarrow u_6\}$ and $\{u_2 \rightarrow u_7 \rightarrow u_2\}$, in the first round of the repeat-loop, the PC circuit obtained for $p_{6,2}$ is $\{u_2 \rightarrow u_4 \rightarrow u_6\}$. In the first round of the repeat-loop, since β_2 is the highest (i.e., 10), we generate an RC over c_2 where the amount of each transaction is 2, which is shown in Figure 5(a). Then, we update the remaining amount of each PC. After the update, $d_{3,1} = 3 - 2 = 1$, $d_{1,2} = 0$, $d_{2,4} = 1$, and $d_{4,3} = 0$. Thus, we remove $d_{1,2}$ and $d_{4,3}$ from P. Similarly, in the second round of the repeat-loop, $\alpha_5 = 1$, $\alpha_6 = 1$, $\alpha_7 = 1$, $\beta_5 = 3$, $\beta_6 = 3$, and $\beta_7 = \frac{4}{3}$. Thus, we generate an RC over c_5, which is shown in Figure 5(b). After the update, $d_{3,1} = 0$, and $d_{5,3} = 0$. Thus, we remove $d_{3,1}$ and $d_{5,3}$ from P. In the third round, since β_8 is larger than β_9, we generate an RC over c_8, which is shown in Figure 5(c). After the update, we remove $d_{2,4}$ and $d_{6,2}$ from P. Then, there is no PC circuit in P. After combining the RCs, we get an RS shown in Figure 1(d).*

4.3 The Analysis of Circuit Greedy

We first prove the time complexity of Circuit Greedy.

Theorem 4.2. *The time complexity of Circuit Greedy is $O(m^3)$, where m is the number of PCs in P.*

Table 2: The Running Example of Circuit Greedy.

Round	PC	PC circuit c_k	α_k	β_k
1^{st}	$\{p_{3,1}, p_{1,5}, p_{5,3}\}$	$c_1 = \{u_1 \rightarrow u_5 \rightarrow u_3\}$	1	3
	$\{p_{1,2}, p_{2,4}, p_{4,3}\}$	$c_2 = \{u_1 \rightarrow u_2 \rightarrow u_4 \rightarrow u_3\}$	2	10
	$\{p_{4,6}, p_{6,2}\}$	$c_3 = \{u_2 \rightarrow u_4 \rightarrow u_6\}$	1	3
	$\{p_{7,6}, p_{2,7}\}$	$c_4 = \{u_2 \rightarrow u_7 \rightarrow u_6\}$	1	$\frac{4}{3}$
2^{nd}	$\{p_{3,1}, p_{1,5}, p_{5,3}\}$	$c_5 = \{u_1 \rightarrow u_5 \rightarrow u_3\}$	1	3
	$\{p_{2,4}, p_{4,6}, p_{6,2}\}$	$c_6 = \{u_2 \rightarrow u_4 \rightarrow u_6\}$	1	3
	$\{p_{7,6}, p_{2,7}\}$	$c_7 = \{u_2 \rightarrow u_7 \rightarrow u_6\}$	1	$\frac{4}{3}$
3^{rd}	$\{p_{2,4}, p_{4,6}, p_{6,2}\}$	$c_8 = \{u_2 \rightarrow u_4 \rightarrow u_6\}$	1	3
	$\{p_{7,6}, p_{2,7}\}$	$c_9 = \{u_2 \rightarrow u_7 \rightarrow u_6\}$	1	$\frac{4}{3}$

Proof sketch. The repeat-loop runs $O(m)$ rounds. The time complexity of the depth-first search procedure is $O(m)$. There are at most m PCs in P. Thus, the total time complexity is $O(m^3)$. ■

In other words, the running time of Circuit Greedy grows linearly with the cube of the number of PCs.

5 THE UAR_DC ALGORITHM

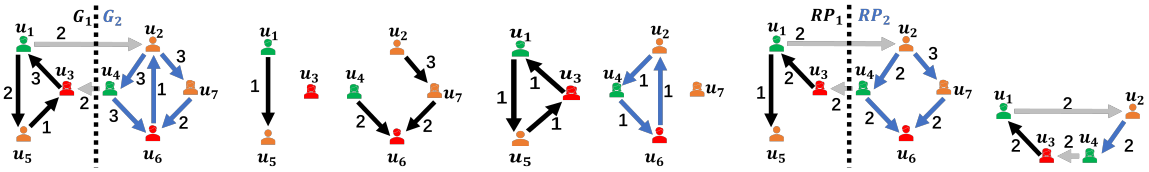
When m is large, the running time of Circuit Greedy can be high. To overcome the shortcoming of the Circuit Greedy algorithm (i.e., the high time complexity), in this section, we propose an efficient heuristic algorithm, namely UAR_DC. UAR_DC divides a large PCN into several small PCNs, obtains an RS over each small PCN, and combines RSs over small PCNs into an RS. Although the approximation ratio of UAR_DC to UAR is unbounded, we prove that UAR_DC obtains an RS over each small PCN with a constant approximation ratio of $(1 - \frac{1}{e})$, where e is Euler's number.

In this section, we first introduce the basic idea of UAR_DC in § 5.1. Next, we describe the details of UAR_DC and demonstrate a running example in § 5.2. Finally, we theoretically analyze the performance of UAR-DC in § 5.3.

5.1 The Basic Idea of UAR_DC

Divide-and-conquer is an efficient method to solve large problem instances [5]. However, it is not easy to design a divide-and-conquer approach for UAR. *Three challenges need to be solved: 1) How to effectively divide a large PCN? 2) How to obtain an RS with a high utility over a small PCN? 3) how to effectively combine the results from small PCNs?* When a PCN is divided into two PCNs, the PCs linking two users in two PCNs will be discarded. Due to the conservation constraint, some RCs may be missed. For example, if we divide the PCN in Figure 1(a) as shown in Figure 6(a), $p_{1,2}$ and $p_{4,3}$ will be discarded. Then, the RS shown in Figure 5(a) cannot be generated in the two small PCNs and may be missed in the final result.

Briefly, UAR_DC solves the first challenge by repeatedly moving a user with the minimized sum of linked PCs' budgets from a PCN until the difference between the numbers of the users in the two PCNs is at most 1. UAR_DC solves the second challenge by enumerating all PC circuits and repeatedly generating an RC with the highest utility over a PC. UAR_DC solves the third challenge by repeatedly generating an RC for a PC circuit involving the PC with the highest budget in the combination of the *residual PCNs* of the two small PCNs and the PCs between the two small PCNs. A *residual PCN* (RP) of a PCN is the set of PCs after generating an RS of the PCN. For example, Figure 6(b) shows the residual PCN of the



(a) A Division (b) A Residual PCN (c) An RS RS_1 (d) An RS RS_2 (e) Two Residual PCNs (f) An RC rc_4

Figure 6: Some Demonstration Figures in Section 5.

PCN in Example 1 after generating the RS in Figure 1(d), where the number next to each PC is the remaining budget of the PC.

The design of UAR_DC is inspired by a theorem and two observations. The theorem states an upper bound on the utility loss caused by dividing a PCN into two PCNs. The utility loss is defined as the difference between the maximal utility of the optimal solution of the large PCN and the sum of the maximal utility of the optimal solution of each small PCN.

Theorem 5.1. *If we divide a large PCN G into two PCNs G_1 and G_2, the utility loss is at most $w_{max} \cdot SD \cdot \ell$, where ℓ is the maximal number of PCs in a PC circuit in G, w_{max} is the maximal weight of a PC, and SD is the sum of the budgets of PCs linking two users in different PCNs, i.e., $SD = \sum_{u_i \in G_1} \sum_{u_j \in G_2} d_{i,j} + \sum_{u_i \in G_2} \sum_{u_j \in G_1} d_{i,j}$.*

Proof sketch. The upper bound of the utility loss is the utilities of RCs involving the PCs between two PCNs. The upper bound of the utility involving a PC $p_{i,j}$ is $w_{max} \cdot \frac{d_{i,j} \cdot \ell}{b_{min}+1}$. ■

Thus, to effectively divide a PCN, we should minimize the utility loss caused by the division. UAR_DC greedily moves a user from a large PCN to a small PCN, minimizing the increase in the sum of the budgets of the PCs between the small PCN and the large PCN. UAR_DC repeatedly moves users until the difference between the number of the users in the small PCN and the number of the users in the large PCN is less than 2. The first observation is that when the number of the users in a PCN is small, the time cost of enumerating all PC circuits in the PCN is acceptable.

Observsition 1. When the number λ of users in a PCN is a small constant, the time complexity of enumerating all PC circuits is a small constant $O(2^{\lambda^2})$.

Based on Observation 1, we design a method to obtain an RS over a small PCN. Specifically, we first enumerate all PC circuits in a small PCN. Then, we repeatedly select a PC circuit to generate an RC with the highest utility until no RC can be generated. The second observation is that the RP of a PCN may form an RC with the RP of another PCN and the PCs between the two PCNs.

Observsition 2. There may be some RCs in the combination of the residual PCNs of two PCNs and the PCs between the two PCNs.

For example, we divide the PCN in Example 1 into two PCNs G_1 and G_2, shown in Figure 6(a). The number next to a PC is the remaining budget of the PC. The PCs in G_1 are in black, the PCs in G_2 are in blue, and the PCs between G_1 and G_2 are in gray. We generate an RS rs_1 in G_1, shown in Figure 6(c), and an RS rs_2 in G_2, shown in Figure 6(d). The RPs of G_1 and G_2 are shown in Figure 6(e). There is an RC rc_4, shown in Figure 6(f), in the combination of RP_1, RP_2, and the gray PCs between G_1 and G_2. Thus, after obtaining RSs over two small PCNs, we combine their RPs and the PCs between the two PCNs and try to find an RS over the combined PCN.

5.2 The Description of UAR_DC

Algorithm 2 shows the pseudo-code of the UAR_DC algorithm. We use **the DC procedure** to recursively divide PCNs and combine the RSs (lines 24-29). If the number of users in G is at most λ, we do not divide G and use the *AllG* procedure to obtain an RS over G (lines 25-26); otherwise, we divide G into two PCNs G_1 and G_2 and obtain an RS over each divided PCN (lines 27-28). Specifically, we first use the *GD* procedure to divide G into G_1, G_2, and G_3, where G_1 and G_2 are two disjoint PCNs divided from G, and G_3 is the set of PCs between G_1 and G_2. Then, we use DC to obtain an RS and an RP over each divided PCN. We combine the RP of G_1, the RP of G_2, and the PCs between G_1 and G_2 as a new PCN G'. We use the *MC* procedure to obtain an RS over G'. We combine the RS obtained from G_1, the RS obtained from G_2, and the RS obtained from G' as RS. Finally, we return RS and RP (line 29).

The GD Procedure. We use GD to divide a large PCN G into two small PCNs, G_1 and G_2 (lines 16-23). We first initialize G_1 as $\emptyset$ and initialize G_2 as G (lien 17). Then, we repeatedly move a user from G_2 to G_1 until the difference between the numbers of the users in the two PCNs is at most 1 (lines 18-21). For each user u_i in G_2, we calculate $SD_i = \sum_{u_j \ in G_2} d_{i,j} + d_{j,i}$ (lines 19-20), which is the sum of the budgets of the PCs between u_i and G_2 after moving u_i. Then, we move the user with the smallest SD_i from G_2 into G_1 (line 21). Then, we move a PC p from G into G_1/G_2 if the two users of p are in G_1/G_2; otherwise, we move p into RP (line 22). Finally, we return G_1, G_2, and RP (line 23).

The AllG Procedure. We use AllG to obtain an RS over each small PCN (lines 10-15). We first enumerate all PC circuits C in G (line 11). Then, we repeatedly generate an RS over a PC circuit in C until no RS can be made (lines 12-13). In each round of the while loop, we generate an RC over a PC circuit in C with the highest utility (line 13). Specifically, for each PC circuit c_i in C, we find the minimal remaining budget α_i of the PCs in c_i. Then, we calculate the utility β_i of an RC over c_i, where the amount of each transaction is α_i. We generate an RC rc_i over c_i with the highest β_i, and the amount of each transaction in rc_i is α_i. Next, we update the remaining budget of each PC in c_i and remove PC circuits in C where some PCs' remaining budgets are zero (line 13). After that, we generate an RS RS and an RP RP according to the generated RCs (line 14). Finally, we return RS and RP (line 15).

The MC Procedure. We use MC to efficiently obtain an RS over an RP (lines 1-9). Since an RP may involve many users, we

Algorithm 2: The UAR_DC algorithm.

Input: a PCN $G = \langle U, P \rangle$, and an integer λ
Output: a rebalance solution RS

1 **Function** MC(G):
2 $P' = G.P$, $RC = \emptyset$;
3 **while** $P' \neq \emptyset$ **do**
4 $p_{g,h} \leftarrow$ the PC with the highest budget in P';
5 Find a PC circuit c involving $p_{g,h}$;
6 Generate an RC over c with the highest utility;
7 Update P';
8 Generate RS and RP by RC;
9 **return** RS and RP;
10 **Function** AllG(G):
11 $C \leftarrow$ enumerate all PC circuits in G;
12 **while** C *is not empty* **do**
13 Generate an RC rc over a PC circuit in C with the highest utility, update C;
14 Generate an RS RS and an RP RP by the made RCs;
15 **return** RS and RP;
16 **Function** GD(G):
17 $G_1 = \emptyset$, $G_2 = G$;
18 **while** $G_1.U.size < G_2.U.size - 1$ **do**
19 **foreach** u *in* $G_2.U$ **do**
20 Canculate SD;
21 Move the user with the smallest SD from G_2 into G_1;
22 Generate G_1, G_2, and RP;
23 **return** G_1, G_2, and RP;
24 **Function** DC(G):
25 **if** $G.U.size \leq \lambda$ **then**
26 RS, RP = AllG(G);
27 **else**
28 (G_1, G_2, RP) = GD(G), (RS_1, RP_1) = DC(G_1), (RS_2, RP_2) = DC(G_2), (RS_3, RP) = MC($RP_1 + RP_2 + RP$);
29 **return** $RS = RS_1 + RS_2 + RS_3$, RP;
30 **Function** Main():
31 RS, RP = DC(G);
32 **return** RS;

cannot use AllG to find an RS. We first initialize P' as G and RC as empty (line 2). Then, we repeatedly generate RCs in G until no RC can be generated (lines 3-8). We first find the PC $p_{g,h}$ with the highest remaining budget in P' (line 4) and find a PC circuit c involving $p_{g,h}$ using a depth-depth procedure (line 5). Next, we generate an RC over c where the amount of each transaction is the lowest remaining budget of the PCs in c (line 6). After that, we update the remaining budget of each PC and remove PCs with zero budget from P' (line 9).Finally, we generate RS and RP by RC and return RP and RS (line 9). Here is a running example.

Example 3. *We use UAR_DC to get a solution to the UAR problem instance in Example 1. We set λ as 4. We use GD to divide the PCN into two small PCNs. Specifically, in the first round of the while-loop at line 19, we get $SD_A = 7$, $SD_B = 9$, $SD_C = 6$, $SD_D = 8$, $SD_E = 3$, $SD_F = 6$, and $SD_G = 5$. Thus, we move E from G_2 to G_1. Then, in the second round of the while-loop at line 19, since E is in G_1, $SD_A = 5$, and $SD_C = 5$. Since SD_A is the smallest, we move A from G_2 to G_1. Similarly, in the third round, we move C from G_2 to G_1. Thus, the result of the division is shown in Figure 6(a).*

Then, we use AllG to obtain an RS over each divided PCN. Two PC circuits exist in G_2, $C = \{c_1, c_2\}$, where $c_1 = \{p_{B,D}, p_{D,F}, p_{F,B}\}$, and $c_2 = \{p_{B,G}, p_{G,F}, p_{F,B}\}$. Thus, $\alpha_1 = 1$, $\alpha_2 = 1$, $\beta_1 = 3$, and $\beta_2 = \frac{4}{3}$. Thus, we generate an RC over c_1 where the amount of each transaction is 1. Since the remaining demand of $p_{F,B}$ is 0, we remove c_1 and c_2 from C. Thus, C is empty, and the RS obtained over G_2 is shown in Figure 6(d). Similarly, we obtain an RS over G_1, shown in Figure 6(c).

Figure 6(e) shows the RPs of G_1 and G_2 and the PCs between G_1 and G_2. We use MC to obtain an RS over the PCN in Figure 6(e). Specifically, $p_{A,B}$ is the PC with the highest budget and is involved in a PC circuit $\{p_{A,B}, p_{B,D}, p_{D,C}, p_{C,A}\}$. Thus, we generate an RC rc_4, shown in Figure 6(f). Then, we remove $p_{A,B}$, $p_{B,D}$, $p_{D,C}$ and $p_{C,A}$ since their remaining budgets are 0. Then, $p_{B,D}$ is the PC with the highest budget. However, no PC circuit involves $p_{B,D}$. Thus, $p_{B,D}$ is removed. Similarly, we remove $p_{D,F}$, $p_{B,G}$, and $p_{G,F}$. Finally, we combine RS_1, RS_2, and rc_4 as the final RS, shown in Figure 1(d).

5.3 The Analysis of UAR_DC

In this subsection, we theoretically prove the approximation ratio of AllG and the time complexity of UAR_DC. Additionally, we discuss the pros and cons of UAR_DC. We first prove that AllG can obtain a constant-approximated solution over a small PCN.

Theorem 5.2. *The approximation ratio of AllG is $1 - \frac{1}{e}$, where e is Euler's number.*

Proof sketch. The utility function $U()$ is monotone and submodular. Moreover, AllG greedily selects a PC circuit from C to generate an RC with the highest utility. Thus, by [20], the approximation ratio of AllG is $1 - \frac{1}{e}$. ■

Although AllG is an algorithm with a constant approximation, its time complexity is not polynomial. Thus, we can only use AllG on small PCNs.

Theorem 5.3. *When λ is a small constant, the time complexity of UAR_DC is $O(n \cdot m^2)$, where n is the number of users in G, and m is the number of PCs in G.*

Proof sketch. The time complexity of AllG is $O(4^{\lambda^2})$. The time complexity of MC is $O(m^2)$. The time complexity of GD is $O(n \cdot m)$. We use each procedure $O(\frac{n}{\lambda})$ times. ■

In other words, when λ is a small constant, UAR_DC can return an RS within polynomial time, and the running time increases linearly with the number of users and the square of the number of PCs. Since $O(m) = O(n^2)$, the time complexity of UAR_DC is lower than that of Circuit Greedy.

6 EXPERIMENTAL STUDY

We seek to answer the following questions in our experimental studies: **Q1.** What is the discrepancy in utility between the RS obtained by our algorithms and the optimal RS with the highest

utility? **Q2.** What are the enhancements of our algorithms in comparison to the existing algorithms? **Q3.** What are our approaches' performances on executing transaction over a real-world PCN? **Q4.** Does the utility of an RS have an impact on the improvement effects on the SRoT? **Q5.** What is the time required for our algorithm to obtain RSs? and **Q6.** How do the relevant parameters affect the effectiveness of our algorithms?

In this section, we first introduce the experimental configuration in Section 6.1. Then, we conduct experiments over small-scale datasets and answer Q1 and Q2 in Section 6.2. Next, we present the experimental results of using our approaches to execute transactions over a real-world PCN and answer Q3 and Q4 in Section 6.3. After that, we conduct experiments over synthetic datasets and answer Q5 and Q6 in Section 6.4. In closing, we summarize our findings from the experiments in Section 6.5.

6.1 Experimental Configuration

Implementation. As introduced in Section 2.2, we need to find a route in a PCN for a transaction. In our experiments, we implement two famous routing methods, the Shortest Path method and the Flash method [38]. Specifically, the Shortest Path method identifies a route with the fewest intermediate users, while the Flash method partitions a transaction into sub-transactions and executes them along multiple routes. For each transaction tx, we first use a routing method to find a route or several routes to execute tx. If we transfer x tokens in a PC p to execute a transaction, we add p into a rebalance list and set its budget as x. We make a round of rebalance over the rebalance list every 50 transactions. If a user transfers x tokens in a PC p in a RS, p temporarily freezes x tokens until the next round of rebalance. We compare our rebalance scheme with another asynchronous rebalance scheme, Cycle [18]. Cycle maintains a set *PCS* of PC sets. Once a transaction is executed in a PC p, Cycle finds a PC set *pcs* in *PCS* and makes a rebalance transaction over p, if there are some PCs in *pcs* can form a PC circuit with p; otherwise, Cycle randomly adds p into a PC set or creates a new PC set.

Moreover, since UAR is NP-hard, we can only get optimal results in small problem instances. Additionally, the scalability of the existing algorithm *Revive* is limited. Thus, we compare Circuit Greedy (CG) and UAR_DC (DC) with an exact method *UE* and the existing method Revive in small-scale experiments. Moreover, we compare our algorithms with two baseline algorithms, *UR* and *UG*, in experiments. Specifically,

- **UE** enumerates all possible solutions and returns the one with the highest utility. Thus, the time complexity of UE is $O((d_{max}+1)^m)$, where d_{max} is the maximal budget of a PC, and m is the number of PCs;
- **Revive** first partitions a large PCN into small PCNs, and each small PCN contains at most ζ PCs. Then, Revive partitions each small PCN into a set of same-structure PCNs where each PC's budget is at most η. Then, Revive runs UE over each small PCN to obtain an RS. Then, Revive combines the obtained RS as the final results. Revive's time complexity is $O(\frac{m \cdot d_{max}}{\zeta \cdot \eta} \cdot (\eta+1)^{\zeta})$, where d_{max} is the maximal budget of a PC, and m is the number of PCs. *Thus, Revive cannot be used in real datasets where d_{max} is very large.* When $\zeta > 7$ and $\eta > 7$, UE's running time is unacceptable (shown in Section 6.2). Thus, we set $\zeta = \eta = 7$ in our experiments;

Table 3: Experimental Settings.

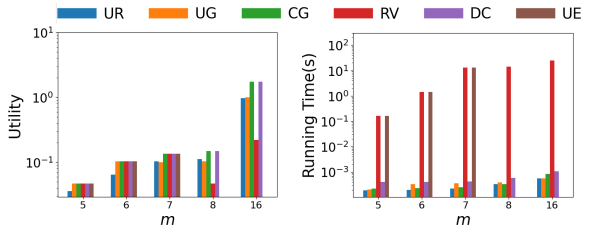
Parameters	Values
Number of users n	60, 70, **80**, 80, 100
Number of PCs m	600, 700, **800**, 900, 1000
Weight range $[w^-, w^+]$	**[1, 1]**, [1, 10], [1, 10^2], [1, 10^3], [1, 10^4]
Balance mean $\bar{b}$	10^3, 10^4, **10^5**, 10^6, 10^7
Balance variance δ	**0**, 10^2, 10^4, $5 * 10^4$, 10^5
Budget range d^+	10^2, 10^3, **10^4**, 10^5, 10^6

- **UR** repeatedly generates an RC in a random manner. In each round, UR first initializes a route τ as empty. Then, UR repeatedly adds a PC $p_{i,j}$ into τ in a random manner where u_i is the end of τ until τ becomes a Hamilton circuit or no PC can be added. If τ is a Hamilton circuit, UR generates an RC over τ where the amount of each transaction is the minimum budget of the PCs in τ; otherwise, UR randomly removes a PC in τ from the PCN; and
- **UG** repeatedly makes an RC in a greedy manner. In each round, UG first initializes τ as the PC $p_{g,h}$ with the highest budget. Then, UG repeatedly adds the PC $p_{i,j}$ with the highest budget into τ where u_i is the end of τ until there is a PC circuit in τ or no PC can be added. If τ includes a PC circuit c, UG makes an RC over c where the amount of each transaction is the minimum budget of the PCs in c; otherwise, UG removes the PC with the minimal budget in τ from the PCN.

Testing Parameters. As defined in § 3, UAR is associated with the following parameters: 1) the number n of users in a PCN, 2) the number m of PCs in a PCN, 3) the budget d of each PC, 4) the weight w of each PC, and 5) the balance b of each PC. Thus, in our experiments, we test the effects of the number of users, the number of PCs, the budget distribution, the weight distribution, and the balance distribution. Moreover, by convention [18, 38], we also test the effects of the number of executed transactions.

Real-world datasets. We use the datasets [33] of LN [30], including 35378 PCs between 2463 users. The maximal balance of a PC is 16777216 SAT, the minimal balance is 1100 SAT, the mean balance is 2802245 SAT, and the balance variance is 4939860.

Synthetic datasets. To comprehensively test the effects of parameters, we generate synthetic datasets. In synthetic datasets, we generate n users and m PCs, where each PC randomly selects two users as the sender and the receiver. As the settings of the experiments of balance protocols [18], we vary n from 60 to 100 and vary m from 600 to 1000. Compared with real datasets, the *PC probability* that any two users have a PC soars from 3‰to 28%. In other words, compared with the PCN in the real datasets, the PCN in the synthetic datasets has a denser PC set. In the real datasets, the value of each PC's balance is large. The reason is that the monetary value of 1 SAT is low (546 SAT ≈ \$0.1 [9]). However, in some PCNs, the monetary value of each transaction unit is higher, and the value of each PC's balance is smaller. Thus, to test the effect of the value of budgets, in synthetic datasets, we randomly set the budget of a PC with a Uniform distribution within $[1, d^+]$, where d^+ varies from 10^2 to 10^6. Moreover, to test the effect of weight distribution, different from real datasets, we uniformly set the weight of each PC within $[w^-, w^+]$, where $[w^-, w^+]$ varies from $[1, 1]$ to $[1, 10^4]$.



(a) Utility of Varying m (b) Running Time of Varying m

Figure 7: Results of Varying m (Small).

Additionally, we set the balance of each PC by a Gaussian distribution. According to the statistic of the real datasets, we vary mean value $\bar{b}$ from 10^3 to 10^7 and vary variance δ from 0 to 10^5.

Comparison metric. For each experiment, we sample 10 problem instances. We report the average SRoT, the average number of shifted tokens, the average amount of the algorithms' running time and the utility of the RSs returned by the approaches:

- The **SRoT** is the ratio of the successfully executed transactions;
- The **number of shifted tokens** of an approach is calculated as the sum of rebalance transactions' amounts;
- The **running time** of an approach is estimated as the time that the approach used to obtain a valid RS. The shorter the running time is, the more efficient the approach is; and
- The **utility** of an RS is calculated as Definition 3. The higher the utility is, the more effective the approach is.

Table 3 illustrates settings on the synthetic datasets, where we mark the default values of parameters in bold font. In each group of experiments, we vary the value of one parameter while setting other parameters' values to their default values. All experiments were run on an Intel CPU@2.8 GHz with 24 GB RAM in Python.

6.2 Experiments over Small Datasets

Since UAR is NP-hard and the time complexity of Revive is high when d_{max} is large, we can only run UE and Revive on small datasets. Thus, in this subsection, we conduct experiments over small datasets to compare our approaches with UE and Revive and answer Q1 and Q2. In the small datasets, we generate 9 users and m PCs. The weight of each PC is randomly set within $(0, 1]$, the balance of each PC is randomly set within $[0, 10]$, and the budget of each PC is set as d^+. Besides, m varies from 5 to 16, and d^+ varies from 5 to 40. The default value for m and d^+ is 7. Due to space limitations, we only present the results of varying m. For similar results of varying d^+, please refer to our technical report [2].

The effect of the number of PCs. As shown in Figure 7(a), when m increases, the utilities of the RSs obtained by all approaches except Revive increase. The reason is that when m gets larger, we can rebalance more PCs, which increases the total utility. Specifically, *the utilities of the RSs obtained by our approaches are close to those of the RSs obtained by UE.* Moreover, the utilities of the RSs obtained by our approaches are higher than the utilities of the RSs obtained by the baselines. When $m \leq 7$, the utilities of the RSs obtained by Revive equal those obtained by UE. The reason is that when $m \leq 7$, Revive does not partition the PCN and directly invokes UE to obtain the RS. However, the utility of the RS obtained

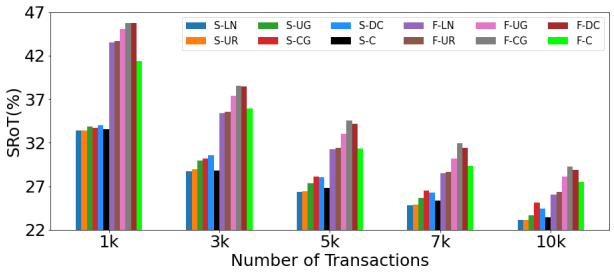


Figure 8: The SRoTs.

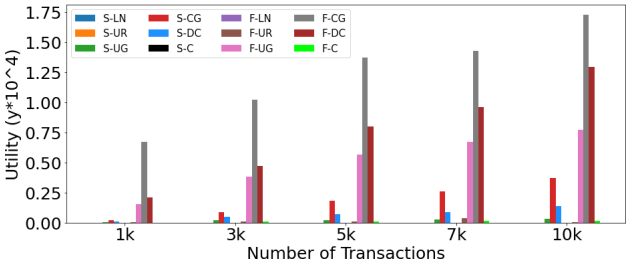


Figure 9: The Utilities.

by Revive when $m = 8$ is lower than that of the RS obtained by Revive when $m = 7$. The reason is that when $m > 7$, Revive randomly partitions a large PCN into small PCNs. The PCs in small PCNs may not form PC circuits. Moreover, unlike DC, Revive does not try to find RCs in the residual PCNs. Thus, the utilities of the RSs obtained by Revive decrease. When $m > 7$, *our approaches outperform Revive at least 3.16 times in utilities.* As shown in Figure 7(b), when m becomes larger, the running time of all approaches increases. Specifically, when m is larger than 7, the running time of UE is extremely high. Thus, in Revive, we set $\zeta = 7$. Moreover, since the number of users is similar to the number of PCs, due to the overhead of the division, DC spends more running time than CG.

6.3 Experiments over Real Datasets

In this subsection, we conduct experiments to execute transactions over a real-world PCN to answer Q3 and Q4. Specifically, the method X-Y uses X as the routing method and Y as rebalance method. For example, $S-LN$ uses Shortest Path as the routing method and does not use a rebalance method, and $F-DC$ uses Flash as the routing method and DC as the method to obtain RSs.

Figure 8 shows the SRoT of each approach. When we execute more transactions, the SRoT decreases. The reason is that when we execute more transactions, the distribution of tokens in PCs is more imbalanced, and the depletion issue is more serious, fitting our introduction in Section 1. Our approaches achieve higher SRoTs than the baselines. Specifically, when using Shortest Path as the routing method, compared to LN, our methods improve the SRoT by 9%. Flash is the state-of-art method to increase the SRoT. However, when using Flash as the routing method, our methods can further improve the SRoT by 12%. Our methods can improve the SRoT even more when using Flash because when using Flash, more PCs need to be rebalanced and it is easier to satisfy the conservation constraint. Compared with the original LN using Shortest Path, combining Flash, our methods can improve the SRoT by 37%. Moreover, our approaches outperform Cycle. The reason is that Cycle makes RSs

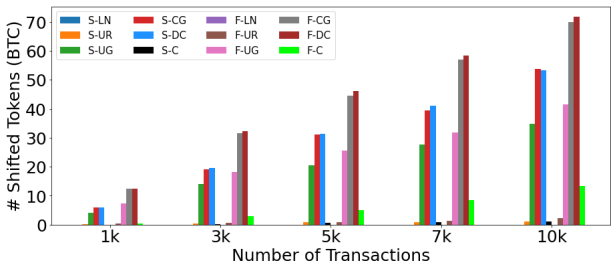
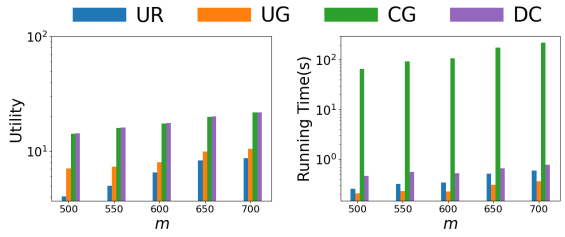


Figure 10: The Number of Shifted Tokens.



(a) Utility of Varying m (b) Running Time of Varying m

Figure 11: Results of Varying m (Synthetic).

locally and asynchronously. Thus, it is hard for Cycle to satisfy the conservation constraint, and the RSs obtained by Cycle are not effective in improving the SRoT.

Figure 9 shows the utilities of the RSs obtained by each approach. With the number of transactions increases, the utility obtained by each approach increases. The reason is that when we execute more transactions, we make more rounds of rebalance and the sum of RSs' utilities is higher. Specifically, the utilities obtained by our approaches are higher than those obtained by the baselines. Meanwhile, the SRoTs achieved by our approaches are also higher than those of the baselines, shown in Figure 8. Particularly, when the number of transactions is smaller than 3k, the distribution of PCs' balances is not highly imbalanced, and the depletion issue is not very serious. In this case, an RS with a higher utility may over-rebalance, leading to a negative effect on SRoT due to freezing tokens. This is why, although F-DC obtains a higher utility than F-CG, the SRoT of F-DC is slightly lower than that of F-CG. In other words, F-DC over-rebalances when the number of transactions is smaller than 3k. Nonetheless, F-DC still outperforms the baselines. When the number of transactions is large (such as in real-world applications), the approach obtaining a higher utility achieves a higher SRoT. Thus, the results in Figure 8 and Figure 9 prove that it is beneficial for increasing SRoT to find an RS with a higher utility. Thus, our UAR problem is realistic, and our approaches are significant.

Figure 10 shows the number of tokens shifted by each approach. As the number of transactions increases, the number of shifted tokens increases. The reason is that when we execute more transactions, we make more rounds of rebalance and rebalance more tokens. Specifically, since LN does not rebalance, the number of shifted tokens of F-LN and S-LN are both zero. Besides, since Cycle makes RSs locally, the number of shifted tokens in the RSs is small. The number of shifted tokens in our algorithms is much larger than that of Cycle and the baselines. In other words, our algorithms generate RSs more effectively.

6.4 Experiments over Synthetic Datasets

In the experiments over real datasets, we cannot test the effect of the weight range, the budget distribution, the balance distribution, the number of users, and the number of PCs. Thus, to comprehensively test the effects of the related parameters on our approaches' performances and answer Q5, in this subsection, we generate synthetic datasets and conduct experiments over them to test the effects of the parameters. Due to space limitations, we only present the results of varying m, δ, and d^+. For similar results of varying n, $[w^-, w^+]$, and $\bar{b}$, please refer to our technical report [2].

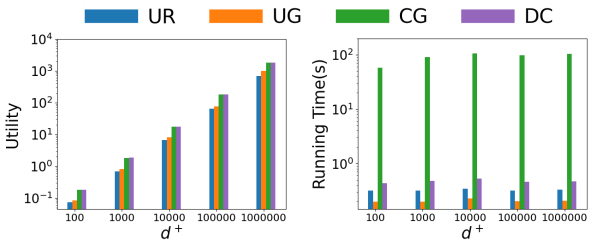
Effect of the number of PCs. As shown in Figure 11(a), when m increases, the utilities of the RSs obtained by all approaches increase, fitting the results in Figure 7(a). The utilities of the RSs obtained by CG are slightly lower (about 1%) than those of the RSs obtained by DC. The reason is that the PCN in synthetic datasets is denser than that in real datasets. Thus, the division in DC causes less utility loss. As shown in Figure 11(b), when there are more PCs, the running time of all approaches increases, fitting the results in Figure 7(b). Specifically, *DC saves at least 99% of CG's running time.* The reason is that in synthetic datasets, the number of PCs is much larger than the number of users.

Effect of the balance variance. As shown in Figure 13(a), when δ gets larger, the utilities of the RSs obtained by all approaches increase. The reason is that when δ gets larger, we can find PCs with lower balances to rebalance, which increases the total utility. Specifically, CG is more sensitive to the change of δ than DC. The reason is that DC finds RCs within small PCNs, but CG finds an RC with the highest utility among the RCs over all PCs. Thus, CG is easier to find RCs over PCs with lower balances, which increases the utility of the RS. As shown in Figure 13(b), when δ gets larger, the running time of all approaches remains the same. The reason is that the balance of each PC does not affect the time complexity of all approaches.

Effect of the budget range. As shown in Figure 12(a), when d^+ gets larger, the utilities of the RSs obtained by all approaches increase. This is because larger budgets allow for more tokens to be rebalanced in PCs, resulting in a higher total utility. As shown in Figure 12(b), in the beginning, the increase in d^+ increases the running time of all approaches. When d^+ is large, the running time of all approaches almost keeps stable as d^+ increases. The reason is that when d^+ is small, the budget constraint limits the number of RCs that can be generated. Thus, when d^+ is small, as d^+ is larger, the budget constraint is more relaxed, and all approaches can generate more RCs, increasing the running time. However, when d^+ is large enough, the budget constraint is sufficiently relaxed that further increases in d^+ have little effect on the running time.

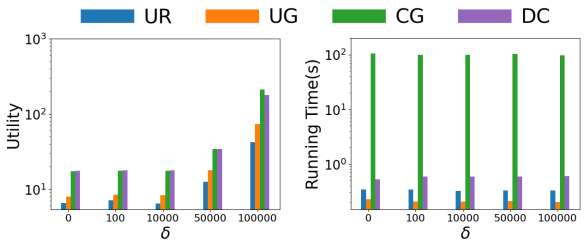
6.5 Experiment Summary

In closing, we summarize our findings as follows: (1) When the scale of the UAR problem instance is small, the utilities of the RSs obtained by our approaches are close to those of the optimal RS with the highest utility. (2) The scalability of Revive is limited. Revive costs huge running time on some real applications where each PC's budget is high, e.g., Lightning Network. When the UAR problem instance contains more than ζ PCs, our approaches outperform the existing approach Revive by at least 3.16 times in terms of utility. (3)



(a) Utility of Varying d^+ (b) Running Time of Varying d^+

Figure 12: Results of Varying d^+ (Synthetic).



(a) Utility of Varying δ (b) Running Time of Varying δ

Figure 13: Results of Varying δ (Synthetic).

Compared to the original LN using Shortest Path, when combined with Flash, our methods can improve SRoT by 37%. (4) The utility of an RS does have an impact on the improvement effects on the SRoT. (5) The running time of our approaches increases as our analysis in Theorem 4.2 and Theorem 5.3. [18] shows that the average running time of a round of rebalancing using Revive is over 30 seconds. Our experimental results demonstrate that our algorithms take less than 1 second to run. Thus, our algorithms' running time is acceptable for practical applications. (6) UAR_DC is suitable for the UAR problem instances where the PCN is dense (e.g., the PC probability is 28%). In dense scenarios, UAR_DC saves at least 99% of the running time of Circuit Greedy.

7 RELATED WORKS

In this section, we discuss the methods to refund a depleted PC, which can be classified into three categories. The methods [30] in the first class are trivial, which propose two transactions on the blockchain to close and reopen a depleted PC. The on-chain transactions are expensive, and the overhead (i.e., proposing two on-chain transactions) of reopening decreases the throughput [19].

The methods [34] in the second class route hop-transactions to alleviate depletion issue. For example, in Figure 1(c), suppose u_7 wants to transfer 3 tokens to u_2. Instead of transferring via a route $\tau_1 = \{u_7 \rightarrow u_2\}$, the methods in the second class will transfer via another route $\tau_2 = \{u_7 \rightarrow u_6 \rightarrow u_2\}$. The reason is that τ_2 can decrease the imbalance of the tokens in $p_{7,6}$ and $p_{6,2}$. However, since the transaction fees and the latency of a hop transaction are proportional to the number of intermediates, τ_2 incurs more transaction fees and latency for users. *Users are unwilling to bear higher transaction fees and delays to help PC owners balance their PCs.* Moreover, since most transactions are unidirectional, the methods in the second class can prolong the time when PCs' balances are depleted, but they *cannot entirely avoid the depletion issue* [4].

The third class of methods for tackling depletion issues is a rebalance protocol, which shifts tokens among PCs. With these protocols, users can rebalance their PCs in place without incurring transaction fees [4]. Participants can customize their rebalance requests, and all transactions are conducted off-chain, avoiding any impact on the blockchain's throughput. Compared to the methods in the other two classes, rebalance protocols are more effective and result in lower transaction fees [14]. Many rebalance protocols have been proposed, including Cycle [18], HIDE&SEEK [3], and Shaduf++ [14]. Cycle [18] rebalances PCs asynchronously without freezing users' tokens, keeping SRoT during rebalance. However, Cycle partitions a PCN into cycles and rebalances each cycle independently, resulting in a sub-optimal solution. HIDE&SEEK [3] obtains rebalancing solutions by multi-party computation. Thus, in HIDE&SEEK, users do not need to tell their balances to other users. However, HIDE&SEEK costs much longer time to obtain rebalancing solutions. Shaduf++ [14] allows a user to shift tokens among her PCs. In other words, when users' PCs do not form a circuit, they still can rebalance their PCs. However, shaduf++ requires an online transaction for each pair of PCs, incurring additional transaction fees. Compared with the algorithms used to obtain rebalancing solutions in the existing protocols, *our approaches can generate RSs with higher utilities.* Moreover, *our approaches can be used in some existing protocols.* For example, a user in Cycle can use our algorithms to decompose her request into sub-requests and assign sub-requests to cycles.

8 CONCLUSION

This paper aims to propose an efficient and effective utility-aware rebalance solution. We define the utility-aware rebalance (UAR) problem. We theoretically prove that UAR is NP-hard, and no deterministic algorithms can solve UAR with a constant approximation ratio. Thus, we propose two heuristic algorithms, namely Circuit Greedy and UAR_DC. Experiments conducted on real and synthetic datasets demonstrate that our approaches outperform the existing approach by achieving a factor of 3.16 improvement in utilities.

ACKNOWLEDGMENTS

Lei Chen's work is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, CRF Project C2004-21GF, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants. Chen Jason Zhang's work acknowledges partial support from the following funding sources: ITF (PRP/009/22FX), PolyU-MinshangCT Generative AI Laboratory (Fund No: P0046453), Research Matching Grant Scheme (Fund No: P0048191), and Research Matching Grant Scheme (Fund No: P0048183), PolyU Start-up Fund by (Fund No: P0046703). Peng Cheng's work is supported by the National Natural Science Foundation of China under Grant No. 62102149. Xuemin Lin's work is supported by NSFC U2241211 and U20B2046. Corresponding author: Peng Cheng.

REFERENCES

[1] [n.d.]. [Online] Lndmanage. https://github.com/bitromortac/lndmanage, Last accessed on 2023-10-18.

[2] [n.d.]. [Online] Technical Reports. https://github.com/HelloGreatWorld/UtilityRebalance, Last accessed on 2023-10-18.

[3] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. 2022. Hide & seek: Privacy-preserving rebalancing on payment channel networks. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 358–373.

[4] Nitin Awathare, Vinay Joseph Ribeiro, Umesh Bellur, et al. 2021. REBAL: Channel balancing for payment channel networks. In *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 1–8.

[5] Peng Cheng, Xiang Lian, Lei Chen, Jinsong Han, and Jizhong Zhao. 2016. Task assignment on multi-skill oriented spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2201–2215.

[6] Xue-Qi Cheng, Fu-Xin Ren, Hua-Wei Shen, Zi-Ke Zhang, and Tao Zhou. 2010. Bridgeness: a local index on edge significance in maintaining global connectivity. *Journal of Statistical Mechanics: Theory and Experiment* 2010, 10 (2010), P10011.

[7] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*. 123–140.

[8] Maya Dotan, Saar Tochner, Aviv Zohar, and Yossi Gilad. 2022. Twilight: A differentially private payment channel network. In *31st USENIX Security Symposium (USENIX Security 22)*. 555–570.

[9] Quinn DuPont. 2019. *Cryptocurrencies and blockchains*. John Wiley & Sons.

[10] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 106–123.

[11] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 801–815.

[12] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. 2019. BlockchainDB: A shared database on blockchains. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1597–1609.

[13] Oğuzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. 2020. How to profit from payments channels. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*. Springer, 284–303.

[14] Zhonghui Ge, Yi Zhang, Yu Long, and Dawu Gu. 2022. Shaduf++: Non-Cycle and Privacy-Preserving Payment Channel Rebalancing. *Cryptology ePrint Archive* (2022).

[15] Jianan Guo, Zhaojie Wang, Hai Liang, Minghao Zhao, Hui An, and Yilei Wang. 2022. Improving transaction succeed ratio in payment channel networks via enhanced node connectivity and balanced channel capacity. *International Journal of Intelligent Systems* 37, 11 (2022), 9013–9036.

[16] Maurice Herlihy. 2018. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*. 245–254.

[17] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. 2019. On the difficulty of hiding the balance of lightning network channels. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 602–612.

[18] Zicong Hong, Song Guo, Rui Zhang, Peng Li, Yufen Zhan, and Wuhui Chen. 2022. Cycle: Sustainable Off-Chain Payment Channel Network with Asynchronous Rebalancing. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 41–53.

[19] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*. 439–453.

[20] Samir Khuller, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Information processing letters* 70, 1 (1999), 39–45.

[21] Kimberly Lange, Elias Rohrer, and Florian Tschorsch. 2021. On the impact of attachment strategies for payment channel networks. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.

[22] Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. 2020. Secure balance planning of off-blockchain payment channel networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1728–1737.

[23] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. 2019. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 63–79.

[24] Xiaofei Luo and Peng Li. 2022. Learning-Based Off-Chain Transaction Scheduling in Prioritized Payment Channel Networks. *IEEE Journal on Selected Areas in Communications* 40, 12 (2022), 3589–3599.

[25] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 455–471.

[26] Ayelet Mizrahi and Aviv Zohar. 2021. Congestion attacks in payment channel networks. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II*. Springer, 170–188.

[27] Wangze Ni, Peng Cheng, and Lei Chen. 2022. Mixing transactions with arbitrary values on blockchains. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2602–2614.

[28] Wangze Ni, Peng Cheng, Lei Chen, and Xuemin Lin. 2021. When the recursive diversity anonymity meets the ring signature. In *Proceedings of the 2021 International Conference on Management of Data*. 1359–1371.

[29] Bernard Péroche. 1984. NP-completeness of some problems of partitioning and covering in graphs. *Discrete applied mathematics* 8, 2 (1984), 195–208.

[30] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments.

[31] Yuhua Qian, Yebin Li, Min Zhang, Guoshuai Ma, and Furong Lu. 2017. Quantifying edge significance on maintaining global connectivity. *Scientific reports* 7, 1 (2017), 1–13.

[32] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 2019. Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In *2019 ieee european symposium on security and privacy workshops (euros&PW)*. IEEE, 347–356.

[33] Stafanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2018. Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions. In *Proceedings 2018 Network and Distributed System Security Symposium*.

[34] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. 2020. High throughput cryptocurrency routing in payment channel networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[35] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. 2020. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–39.

[36] Gijs Van Dam, Rabiah Abdul Kadir, Puteri NE Nohuddin, and Halimah Badioze Zaman. 2020. Improvements of the balance discovery attack on lightning network payment channels. In *ICT Systems Security and Privacy Protection: 35th IFIP TC 11 International Conference, SEC 2020, Maribor, Slovenia, September 21–23, 2020, Proceedings 35*. Springer, 313–323.

[37] Vijay V Vazirani. 2001. *Approximation algorithms*. Vol. 1. Springer.

[38] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 370–381.

[39] Han Xue, Qun Huang, and Yungang Bao. 2021. EPA-Route: Routing payment channel network with high success rate and low payment fees. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 227–237.

[40] Philipp Zabka, Klaus-T Foerster, Christian Decker, and Stefan Schmid. 2022. Short paper: A centrality analysis of the lightning network. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 374–385.

[41] Peilin Zheng, Quanqing Xu, Zibin Zheng, Zhiyuan Zhou, Ying Yan, and Hui Zhang. 2021. Meepo: Sharded consortium blockchain. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1847–1852.