Fair Distribution of Digital Payments: Balancing Transaction Flows for Regulatory Compliance

Ashlesha Hota*
IIT Kharagpur
India

Shashwat Kumar*
IIT Kharagpur
India

Daman Deep Singh
IIT Delhi
India

Abolfazl Asudeh
University of Illinois Chicago
United States

Palash Dey
IIT Kharagpur
India

Abhijnan Chakraborty
IIT Kharagpur
India

Abstract

The concentration of digital payment transactions in just two UPI apps - PhonePe and Google Pay - has raised concerns of duopoly in India's digital financial ecosystem. To address this, the National Payments Corporation of India (NPCI) has mandated that no single UPI app should exceed 30% of total transaction volume. Enforcing this cap, however, poses a significant computational challenge: how to redistribute user transactions across apps without causing widespread user inconvenience while maintaining capacity limits?

In this paper, we formalize this problem as the Minimum Edge Activation Flow (MEAF) problem on a bipartite network of users and apps, where activating an edge corresponds to a new app installation. The objective is to ensure a feasible flow respecting app capacities while minimizing additional activations. We further prove that Minimum Edge Activation Flow is NP-complete. To address the computational challenge, we propose scalable heuristics, named Decoupled Two-Stage Allocation Strategy (DTAS), that exploit flow structure and capacity reuse. Experiments on large semi-synthetic transaction network data show that DTAS finds solutions close to the optimal ILP within seconds, offering a fast and practical way to enforce transaction caps fairly and efficiently.

Keywords

UPI, Transaction caps, Flow Optimization, Fairness,Greedy heuristics

1 Introduction

Digital payments in India have witnessed explosive growth through the Unified Payments Interface (UPI), an interoperable platform that allows seamless money transfers between bank accounts. Despite the presence of over 15 UPI applications, the ecosystem is highly concentrated: only a few apps like PhonePe, Google Pay, and Paytm account for over 97% of all UPI transactions [4, 6]. This duopoly-like concentration poses serious concerns for competition, resilience, and systemic reliability. If a dominant app experiences downtime, a substantial portion of India's digital payments network can become temporarily non-functional, threatening both financial stability and public trust.

To address these risks, the National Payments Corporation of India (NPCI) introduced a market share cap stipulating that no single UPI app may process more than 30% of total transactions [11]. This policy aims to ensure that the transaction load is distributed more evenly across the network, reducing dependence on a few dominant players and encouraging innovation among smaller competitors. However, enforcing such a limit in practice is nontrivial because users naturally gravitate toward familiar apps, and transaction behavior is user-driven rather than centrally controlled. The key technical challenge is: How can transaction flow be regulated across apps in real time while minimizing user disruption and maintaining regulatory compliance?

The simplest enforcement method is a tail-drop mechanism, where once an app exceeds its transaction share, subsequent transactions through it are blocked. This method guarantees compliance but at the cost of transaction failures and poor user experience, as users receive no advance warning. A more practical and user-friendly alternative is an alert-based system, where users are notified when an app approaches its transaction cap and are encouraged to switch to another app. This approach can be implemented through a Random Early Detection (RED) like strategy, where the regulator (e.g., NPCI) alerts an app before it actually exceeds its limit, prompting a smoother redistribution of transactions. Such early signaling can maintain compliance while preserving user satisfaction and avoiding abrupt transaction failures.

However, implementing this strategy effectively requires addressing real-world behavioral and systemic challenges. Users are often reluctant to install and maintain multiple UPI apps, and simply providing alerts may not guarantee sufficient switching. While smaller apps or the government could offer targeted incentives to promote redistribution, these efforts are limited by fixed incentive budgets. Therefore, the key challenge is to design a minimal-intervention strategy that leverages alerts and selective incentives to achieve a balanced transaction distribution with minimum additional app installations and limited incentive overhead.

We formulate this problem as a Minimum Edge Activation Flow problem on a flow network $G = (V = \{s\} \cup U \cup A \cup \{t\}, E = E_{\text{solid}} \cup E_{\text{dashed}})$, where users U and apps A form the two partitions. Solid edges correspond to pre-installed apps, and dashed edges represent potential installations. Users are connected to a source s, with capacities equal to their transaction volumes, and apps are connected to a sink t, with capacities corresponding to their allowed transaction share. The objective is to activate the minimal set of dashed edges that allows a feasible integral flow from source to sink without exceeding app capacities. Despite the intuitive structure of the formulation, the underlying optimization problem is computationally challenging. Determining the smallest set of additional app installations or equivalently, the minimum number of dashed edges that must be activated to ensure a feasible routing of all user transactions within app capacity limits—is NP-complete. The hardness holds even under

*Both authors contributed equally to this research.

restrictive settings, such as when the number of apps is limited to three. This result rules out the existence of a polynomial-time exact algorithm unless P =NP. Motivated by this, we develop efficient greedy heuristics that approximate the optimal solution while maintaining practical feasibility in large-scale transaction networks. Our key contributions are the following:

(1) **Computational intractability.** We establish that the decision version of the problem is NP-complete via a polynomial-time reduction from 3-Partition.
(2) **Greedy Heuristics.** To address computational intractability, we propose a linear programming (LP) relaxation of the integral flow constraints. The relaxation provides a tractable method to compute lower bounds and fractional flow solutions, which can guide edge activation decisions and inform practical redistribution strategies.
   - ▷ CARL (**Capacity-Aware Reuse-first Layered Allocation**): The CARL heuristic prioritizes efficient capacity utilization and reuse of already deployed applications. It operates by processing users in ascending order of their transaction-to-capacity ratio, thereby giving precedence to users with limited available capacity relative to their transactional load. For each user, CARL performs allocation in three structured layers: (i) preinstalled apps, (ii) existing extra apps, and (iii) new extra apps if additional capacity is required. Within each layer, apps are sorted by their remaining capacity in descending order to ensure balanced load distribution. This layered reuse-first design minimizes redundant app installations, enhances overall system stability, and improves scalability. By emphasizing high-capacity reuse and adaptive layering, CARL achieves a judicious trade-off between transaction coverage and capacity fairness across users and applications.
   - ▷ DTAS (**Decoupled Two-Stage Allocation Strategy**): The DTAS heuristic employs a two-phase greedy framework that significantly improves upon CARL by enforcing stricter separation between allocation tiers. In *Phase 1*, all transactions are allocated exclusively through preinstalled apps, ensuring that every user fully utilizes their zero-cost connections before accessing additional capacity. This prevents the premature allocation of existing extra apps to a single user, which in CARL could lead to early saturation of such apps and consequently restrict access for other users who have them preinstalled. In *Phase 2*, DTAS allocates the remaining unmet demand by first reusing existing extra apps (installed for other users) and then introducing new extra apps only when necessary. By prioritizing preinstalled capacity globally before activating new edges, DTAS achieves superior capacity preservation, fairness, and reduced installation overhead. Empirical evaluation shows that this disciplined two-phase structure allows DTAS to outperform CARL in both total feasible allocation and capacity balance across the network.
(3) **Experimental Evaluation.** To empirically validate our proposed allocation strategies, we conducted extensive experiments across configurations ranging from 10,000 to 100 million transactions and up to 1.2 million users. Each setup was tested under consistent capacity constraints with 20 varying transaction-to-user ratios. Both heuristics—CARL (Capacity-Aware Reuse-first Layered Allocation) and DTAS (Dual-Tier Allocation Strategy)—were evaluated against the optimal Integer Linear Program (ILP) and its Linear Programming (LP) relaxation. Across all configurations, both algorithms achieved 100% feasibility and stability, demonstrating their robustness at scale. Notably, DTAS consistently matched the ILP-optimal objective, differing by at most one or two app installations in rare instances. This near-optimal performance was accompanied by impressive computational efficiency, DTAS achieved near-optimal allocations in few seconds, over 99% faster than the LP baseline, while maintaining up to 30% runtime improvement over CARL at scale. These results emphasize the practical scalability and precision of our approach, positioning DTAS as an effective balance between optimality and efficiency in large-scale transaction allocation environments.
(4) **Dataset Generation.** We have used a semi-synthetic dataset of transactions. Rabobank, a Dutch bank, provides data in the form of cumulative transactions, and based on the reported statistics of C2C (customer-to-customer), C2B (customer-to-business), and B2B (business-to-business) transactions, this dataset has been generated [9]. The data is presumed to be from June 2023, and six years of transaction data have been uniformly divided across 30 days of the month to simulate daily activity. The dataset consists of four columns: *start id* (the ID of the user initiating the transaction), *value* (the amount of money transferred), *day* (the day of the month when the transaction occurred), and *end id* (the ID of the user receiving the money).
   Additionally, the distribution of UPI applications among users has been generated using official NPCI statistics, which provide the monthly transaction volumes handled by each UPI app [7]. Each user in the dataset is assigned a set of preinstalled UPI apps based on this real-world transaction volume distribution. Specifically, higher-volume apps such as PhonePe and GPay are more likely to be allocated to a larger portion of users, while smaller-volume apps (e.g., Axis Bank, Airtel Payments Bank) are assigned less frequently. This probabilistic allocation reflects the real-world penetration and usage patterns of UPI applications across the user base.

2 Background and Related Work

The Unified Payments Interface (UPI), developed by the National Payments Corporation of India (NPCI) under the guidance of the Reserve Bank of India (RBI), has emerged as a cornerstone of India's digital economy. It serves as an interoperable architecture framework equipped with standardized Application Programming Interfaces (APIs) that enable seamless, real-time, and secure fund transfers between individuals and merchants. UPI's open and inclusive design allows multiple banks and fintechs to integrate easily, promoting innovation and financial inclusion at scale [8].

As of 2024, UPI has over 350 million active users, connects more than 550 banks, and powers 77 payment apps such as Google Pay, PhonePe, BHIM, and WhatsApp Pay [1]. It has enabled the deployment of over 340 million merchant QR codes across India. In 2023, UPI processed 117 billion transactions worth USD 2.19 trillion, with Person-to-Merchant (P2M) payments accounting for around 62% of total transactions [1]. UPI's phenomenal growth reflects India's effective public–private partnership model, combining government vision, RBI's progressive regulatory framework, and collaboration among banks, fintechs, and merchants. However, rising transaction volumes have simultaneously increased the risk of fraud and anomaly. Several scholarly works, such as [2, 3, 5], have emphasized AI-driven approaches to strengthen fraud detection mechanisms in digital payment systems.

Beyond fraud detection, another critical concern in the financial domain is fairness in decision-making. Unfair decision-making in financial services occurs when certain groups or individuals face biased treatment in areas such as loan approvals, credit scoring, or mortgage access. Such bias can arise from historical discrimination or from machine learning models that inadvertently learn unfair patterns from data. Consequently, minority groups or equally qualified applicants may be denied fair financial opportunities, reinforcing existing social and economic inequalities. Song et al. [10] address this issue by proposing a Temporal Fair Graph Neural Network (TF-GNN) framework that models financial transactions as dynamic networks and enforces individual fairness over time. They introduce two new fairness notions specific to temporal graphs, provide a theoretical analysis of their fairness regret, and demonstrate through real-world experiments that their approach improves both prediction accuracy and fairness compared to prior methods.

There is another growing problem — the UPI ecosystem has become too concentrated, with over 80% of the 19.63 billion transactions (worth Rs. 24.90 lakh crore) processed in September 2025 being handled by just two third-party apps [6]. This duopoly poses serious systemic and security risks, as any disruption in these platforms could cripple a major share of India's digital payments. Moreover, such extreme concentration stifles innovation, reduces competition, and threatens fair market access within the country's largest payment infrastructure. To the best of our knowledge, this is the first work that formally studies and models this concentration issue, proposing mechanisms to enhance resilience and promote a more balanced and inclusive UPI ecosystem.

3 Problem Formulation

We model the UPI transaction balancing problem as a bipartite flow network. Let U denote the set of users and A denote the set of UPI apps. Each user $u \in U$ generates t_u transactions, while each app $a \in A$ can process at most c transactions, typically defined as a fraction of the total transaction volume. Users may already have certain apps installed, represented by solid edges $E_{\text{solid}} \subseteq U \times A$, while potential additional installations are represented by dashed edges $E_{\text{dashed}} \subseteq U \times A$. A source node s is connected to all users with edges of capacity t_u, and a sink node t is connected to all apps with edges of capacity c. The goal is to identify the minimal subset of dashed edges $E' \subseteq E_{\text{dashed}}$ that need to be activated such that all transactions can be routed from s to t without exceeding app capacities or dropping transactions. Figure 1 illustrates our construction.

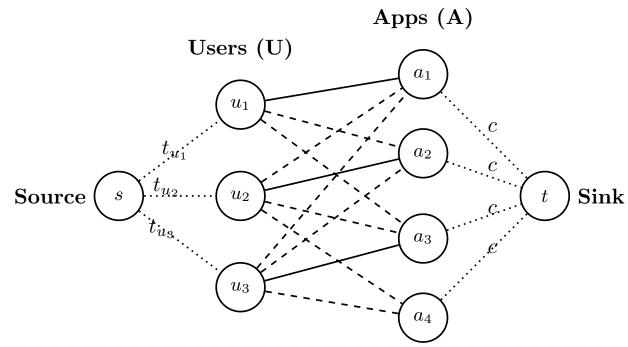

Figure 1: Illustration of the bipartite UPI transaction flow network. Solid edges represent pre-installed apps (E_{solid}), dashed edges denote potential installations (E_{dashed}), and dotted edges indicate user transactions (t_{u_i}) and app capacities (c).

> Minimum Edge Activation Flow (MEAF)
>
> **Input.** Given a bipartite graph $G = (U \cup A, E_{\text{solid}} \cup E_{\text{dashed}})$, a source node s connects to each $u \in U$ by an edge of capacity t_u, and a sink node t connects to each $a \in A$ by an edge of capacity c. Edges in E_{solid} represent existing connections, while E_{dashed} denote optional edges that can be activated.
>
> **Output:** Find the smallest subset $E' \subseteq E_{\text{dashed}}$ such that all demands t_u can be routed from s to t through $E_{\text{solid}} \cup E'$ without violating any capacity c.

Formally, let $f(u, a)$ denote the number of transactions routed from user u to app a. The flow must satisfy the following conditions: every user's transactions are fully routed, i.e., $\sum_{a \in A} f(u, a) = t_u$, and the total flow into any app respects its capacity, $\sum_{u \in U} f(u, a) \leqslant c$. Flow is allowed only on solid edges or activated dashed edges, and transaction units are indivisible, so $f(u, a) \in \mathbb{Z}_{\geqslant 0}$. Finally, the total flow in the network equals the total number of transactions, ensuring no transaction is dropped. The objective is to minimize $|E'|$, the number of additional app installations required to achieve a feasible integral flow. We now formally show Minimum Edge Activation Flow is NP-complete.

Theorem 3.1. *Minimum Edge Activation Flow is NP-complete.*

Proof: It is easy to see that the problem is in NP: given a set of activated dashed edges and an integral flow assignment, we can verify in polynomial time that (i) flow conservation holds, (ii) capacities on (s, u) and (a, t) are respected, (iii) flow is sent only on activated dashed edges, and (iv) the number of activated dashed edges is at most k.

We prove NP-hardness by a polynomial-time reduction from 3-Partition. Let an instance of 3-Partition be given by $S = \{s_1, \ldots, s_{3m}\}$ and B with $\sum_i s_i = mB$ and $B/4 < s_i < B/2$ for all i. We construct an instance of UPI flow problem as follows.

Construction. Create $3m$ users $U = \{u_1, \ldots, u_{3m}\}$ with $t_{u_i} = s_i$, $\forall i \in [3m]$. Create m apps $A = \{a_1, \ldots, a_m\}$ with capacities $c_{a_j} = B$

for all j. Add source s and sink t with edges (s, u_i) of capacity t_{u_i} and edges (a_j, t) of capacity B. Let $E_{\text{solid}} = \emptyset$ and $E_{\text{dashed}} = U \times A$ (i.e., every user can connect to every app via a dashed edge). All (u, a) edges have sufficiently large capacity (e.g., capacity t_u) so that the app capacities are the binding constraints. Set the activation budget $k := 3m$.

This construction is clearly polynomial in the input size.

($\Rightarrow$) If the 3-Partition instance is a YES-instance, then the constructed flow instance admits a feasible integral flow using at most $k = 3m$ activated dashed edges. Assume s can be partitioned into m disjoint triples $T_1, \ldots, T_m$ with $\sum_{s \in T_j} s = B$ for each j. For each triple T_j, choose an app a_j and for each user u_i with $s_i \in T_j$ activate exactly the dashed edge (u_i, a_j) and send the full amount $t_{u_i} = s_i$ on that edge. For each app a_j, the incoming flow is $\sum_{u_i \in T_j} t_{u_i} = B$, so the capacity (a_j, t) is respected. Each user uses exactly one dashed edge, so the number of activated dashed edges is exactly $3m = k$. All source capacities (s, u_i) are saturated, hence the total flow value is $\sum_i s_i = mB$. Thus there exists a feasible integral flow using at most k activations.

($\Leftarrow$) If the constructed flow instance admits a feasible integral flow using at most $k = 3m$ activated dashed edges, then the 3-Partition instance is a YES-instance. Suppose there is a feasible integral s–t flow of value $\sum_{i=1}^{3m} s_i = mB$ using at most $3m$ activated dashed edges. Since $E_{\text{solid}} = \varnothing$ and every user u_i has positive demand $t_{u_i} = s_i > 0$, each user must have at least one activated outgoing dashed edge in order to send any flow. Therefore any feasible solution uses at least $3m$ activated dashed edges. By the budget bound, the solution uses exactly $3m$ activations, hence each user activates *exactly one* dashed edge and sends all of its (integral) demand through that edge.

Let $S_j \subseteq U$ be the set of users assigned to app a_j (i.e., those for which (u, a_j) is activated and carries the full t_u). Flow feasibility and capacity imply for every j that

$$\sum_{u \in S_j} t_u \; \leqslant \; c_{a_j} \; = \; B.$$

Summing over all apps and using that the total flow equals mB gives

$$\sum_{j=1}^{m} \sum_{u \in S_j} t_u \; = \; \sum_{u \in U} t_u \; = \; mB \; = \; \sum_{j=1}^{m} B,$$

which forces equality in each app separately: $\sum_{u \in S_j} t_u = B$ for all j. Finally, by the 3-Partition bounds $B/4 < t_u < B/2$, no app can receive 1 item (any $t_u < B/2$) or $\geqslant 4$ items (each $t_u > B/4$ would exceed B). Therefore each S_j has exactly three users and their demands sum to B. The family $\{S_1, \ldots, S_m\}$ thus yields a partition of s into m triples each summing to B, i.e., a YES-solution for 3-Partition. □

Remark 3.2. The Minimum Edge Activation Flow problem generalizes the classical Set Cover problem. Consequently, unless P = NP, it admits no polynomial-time approximation algorithm with a factor better than $(1 - o(1)) \log n$.

This problem can be formulated as an Integer Linear programming (ILP). We introduce variables $f(u, a) \in \mathbb{Z}_{\geqslant 0}$ representing the number of transactions routed from u to a, and binary variables $x(u, a)$ for $(u, a) \in E_{\text{dashed}}$ indicating whether a potential edge is activated.

The objective is to minimize the number of additional edges activated:

$$\min \sum_{(u,a) \in E_{\text{dashed}}} x(u, a). \tag{1}$$

Each user's transactions must be fully routed, which is captured by

$$\sum_{a \in A} f(u, a) = t_u, \quad \forall u \in U, \tag{2}$$

while the capacities of apps must not be exceeded:

$$\sum_{u \in U} f(u, a) \leqslant c_a, \quad \forall a \in A. \tag{3}$$

Flow through a dashed edge is permitted only if the edge is activated:

$$f(u, a) \leqslant t_u \cdot x(u, a), \quad \forall (u, a) \in E_{\text{dashed}}. \tag{4}$$

Flow on solid edges is unconstrained by activation, i.e., $f(u, a) \geqslant 0$ for $(u, a) \in E_{\text{solid}}$. All flows are integral:

$$f(u, a) \in \mathbb{Z}_{\geqslant 0}, \quad x(u, a) \in \{0, 1\}. \tag{5}$$

4 Heuristic Algorithms for Minimum Edge Activation Flow

Given the computational hardness of the Minimum Edge Activation Flow problem, we first analyze a relaxed version of the Integer Linear Programming (ILP) formulation to assess the strength of the model and the impact of relaxation on fairness and runtime. Specifically, the edge activation constraint in the ILP was relaxed such that activation variables corresponding to dashed edges were allowed to take continuous values in the range $[0, 1]$, while retaining integrality for other decision variables. This relaxation is consistent with the modeling assumption that flow on solid edges is unconstrained by activation, i.e.,

$$0 \leqslant x(u, a) \leqslant 1 \quad \forall (u, a) \in E_{\text{dashed}}. \tag{6}$$

The relaxed ILP significantly reduced runtime, enabling faster convergence for larger data instances. However, despite this computational advantage, the integrality gap between the relaxed and original ILP solutions was observed to be nearly **0%**, indicating that the ILP formulation is structurally tight and highly expressive. Furthermore, the relaxation did not yield notable improvements in fairness: the skewness in transaction distribution across applications remained largely unchanged.

These findings highlight that while the ILP formulation is strong, fairness and scalability improvements require algorithmic intervention beyond relaxation. To this end, we develop a series of **capacity-aware heuristic algorithms** that progressively enhance allocation efficiency, minimize redundant app installations, and maintain user trust stability. We begin with a baseline version of CARL, refine it based on empirical insights from transaction data, and finally introduce DTAS, which employs a two-stage allocation structure with an improved sorting logic to further enhance fairness and scalability.

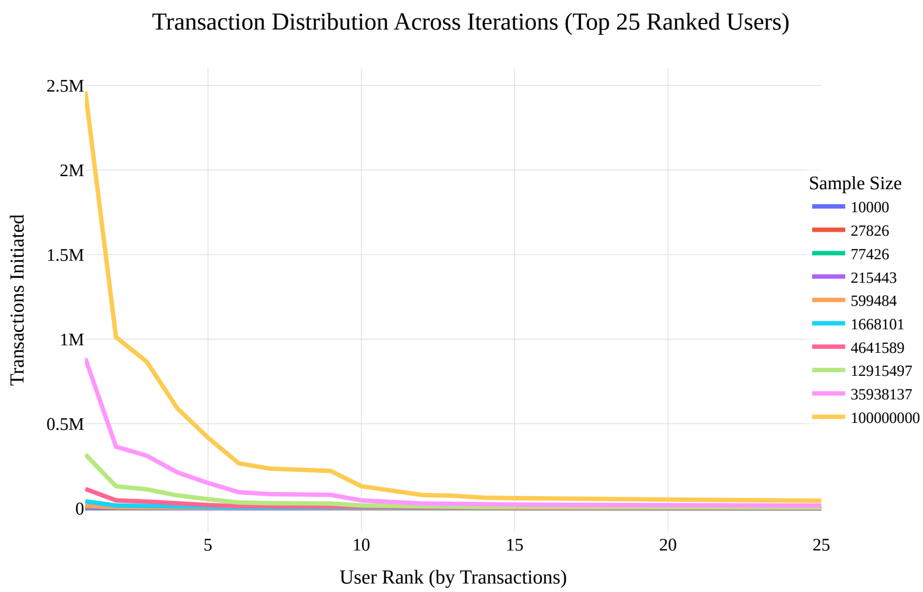

Figure 2: A few users generate most transactions, while the majority contribute very little.

4.1 Preliminary Capacity-Aware Reuse-first Layered Allocation

The initial version of Capacity-Aware Reuse-first Layered Allocation (CARL) was developed to allocate transactions in a structured, layered manner with the objective of minimizing redundant app installations while ensuring full transaction coverage. In this preliminary design, users were processed in **descending order of their transaction-to-capacity ratio**, based on the intuition that heavy users should be prioritized to prevent app capacity overflow in later stages.

However, when tested on large-scale transaction datasets, this strategy exhibited significant inefficiencies due to the **highly skewed transaction distribution**. As illustrated in Figure 2, a small fraction of users generated the majority of transactions, while most users contributed only minimally. Processing heavy users first led to the early saturation of high-capacity applications, leaving lightweight users with limited or no access to their preinstalled apps. This forced many small users to install additional apps even when aggregate capacity was sufficient.

These effects manifested in two major shortcomings:

(1) **Redundant installations:** Several lightweight users were compelled to install extra apps simply because their preferred high-capacity apps were prematurely exhausted.
(2) **Poor utilization balance:** Heavy users dominated the capacity of major apps, while smaller apps remained underutilized.

Although Preliminary CARL achieved full transaction routing, it did so with suboptimal fairness, higher installation overhead, and reduced long-term scalability.

4.2 Refined Capacity-Aware Reuse-first Layered Allocation

To address these shortcomings, the refined version of CARL reorders users in **ascending order** of their transaction-to-capacity ratio, effectively prioritizing small-volume users. This adjustment ensures that lightweight users benefit from available capacity early, resulting in better reuse of existing app capacity and a substantial reduction in redundant installations.

The refined algorithm assigns transactions through the following three allocation layers:

(1) **Preinstalled Apps:** Users first utilize the apps they already have installed, incurring no additional installation cost.
(2) **Existing Extra Apps:** If preinstalled apps lack sufficient capacity, the algorithm uses extra apps previously installed by any user.
(3) **New Extra Apps:** Only as a last resort does the algorithm install new apps for a user, minimizing overall installation count.

This refined ordering yielded significant performance improvements. Under the experimental setup, the runtime dropped by nearly **99%**—from approximately 10,000 seconds to just **15 seconds**—while still producing results **very close to the optimal ILP** in terms of installation count and fairness.

Moreover, this *reuse-first layered allocation* not only enhances capacity utilization but also aligns with real-world behavioral constraints—users are reluctant to install multiple UPI apps, and incentive programs are often limited by budget. By recommending already adopted apps before introducing new ones, CARL fosters user trust, reduces friction, and achieves a balanced transaction distribution with minimal intervention.

Algorithm 1 Capacity-Aware Reuse-first Layered Allocation

Initialize $apps \leftarrow [0 \ldots |initialApps| - 1]$, $remainingCapacity[app] \leftarrow c$
$transactionsHandled[app] \leftarrow 0$, $extraApps \leftarrow \emptyset$, $userApps[u] \leftarrow mp[u]$
for $(u, apps)$ in mp **do** $extraApps \leftarrow extraApps \cup apps$
end for
$users \leftarrow [(id, mp[id], userTransactions[h])]$ for valid (id, h); Sort by $\frac{|transactions|}{|preinstalled| \cdot c}$ desc
$totalRemaining \leftarrow 0$
for $(user, preinstalled, trans)$ in $users$ **do**
  $remaining \leftarrow |trans|$; sort $preinstalled$ by $remainingCapacity$ desc
  for a in $preinstalled$ **do**
    if $remaining \leqslant 0$ **then break**
    end if
    $alloc \leftarrow \min(remaining, remainingCapacity[a])$
    if $alloc > 0$ **then** $remainingCapacity[a] - = alloc$; $transactionsHandled[a] + = alloc$; $remaining - = alloc$
    end if
  end for
  if $remaining > 0$ **then**
    $extras \leftarrow \{a \in extraApps \mid a \notin preinstalled\}$ sorted by $remainingCapacity$
    for a in $extras$ **do**
      if $remaining \leqslant 0$ **then break**
      end if
      $alloc \leftarrow \min(remaining, remainingCapacity[a])$
      if $alloc > 0$ **then** $remainingCapacity[a] - = alloc$; $transactionsHandled[a] + = alloc$; $remaining - = alloc$; $userApps[user] \cup = \{a\}$
      end if
    end for
  end if
  if $remaining > 0$ **then**
    $newApps \leftarrow \{a \in apps \mid a \notin (preinstalled \cup extraApps)\}$ sorted by $remainingCapacity$
    for a in $newApps$ **do**
      if $remaining \leqslant 0$ **then break**
      end if
      $alloc \leftarrow \min(remaining, remainingCapacity[a])$
      if $alloc > 0$ **then** $remainingCapacity[a] - = alloc$; $transactionsHandled[a] + = alloc$; $remaining - = alloc$; $extraApps \cup = \{a\}$; $userApps[user] \cup = \{a\}$
      end if
    end for
  end if
  $totalRemaining + = remaining$
end for
$newMP \leftarrow \{u : list(userApps[u])\}$; $extraAppsInstalled \leftarrow \sum_u |newMP[u]| - totalInitial$
return $(newMP, transactionsHandled, totalRemaining)$

4.3 Decoupled Two-Stage Allocation Strategy (DTAS)

Building on the insights gained from the refined CARL algorithm, we introduce the
DTASf(DTAS), a significantly improved heuristic designed to minimize inter-user interference, prevent premature app saturation, and further reduce redundant installations. While CARL operates in a per-user, layer-by-layer fashion, DTAS fundamentally restructures the allocation workflow by **decoupling the allocation into two global phases**. This structural redesign eliminates several systemic inefficiencies observed in CARL and produces allocations that are consistently closer to the ILP-optimal solution.

Two-Stage Allocation Framework. Unlike CARL, which attempts to satisfy each user sequentially across multiple layers, DTAS processes all users collectively in two distinct stages:

(1) **Stage 1: Preinstalled App Allocation.** In this stage, all users are allocated as many transactions as possible using only their preinstalled apps. By prioritizing preinstalled capacity upfront, DTAS ensures that every user's inherent, zero-cost connections are fully leveraged before the system begins introducing or reusing extra apps. This prevents later phases from unintentionally consuming the preinstalled capacity of users who rely on a limited number of apps.
(2) **Stage 2: Extra App Allocation.** Once all preinstalled allocations are complete, the algorithm addresses the remaining unmet demand for each user. It first attempts to route additional transactions through **existing extra apps** that were previously installed for any user, promoting global reuse. Only if necessary—i.e., when no sufficient existing capacity is available—does DTAS install a **new extra app** for that user, thereby minimizing installation overhead.

This global decoupling is crucial. In CARL, later users could inadvertently saturate an app that served as the only preinstalled option for some earlier user, forcing the latter to install a new app unnecessarily. With DTAS, such conflicts are eliminated because all preinstalled utilizations are resolved before any extra-app allocations begin.

Refined User Ordering. To further improve fairness and capacity preservation, DTAS introduces a refined user-sorting mechanism. Users are processed in **ascending order of their transaction counts**. This prioritizes smaller users, ensuring they receive early access to available capacity.

For users with equal transaction volume, DTAS applies a second-level tie-breaker based on the **number of preinstalled apps**. Users with fewer preinstalled apps are routed earlier because they have fewer natural allocation options. This ordering prevents scenarios in which:

- ▷ a user with only one or two preinstalled apps loses access to those apps due to saturation by a more flexible user, or
- ▷ capacity is consumed by users who could have used alternate apps, leaving constrained users underserved.

This refined sorting ensures a more equitable distribution of limited capacity and preserves user trust by reducing unnecessary app installations.

Advantages of DTAS Over CARL. Through its two-stage separation and refined ordering, DTAS resolves key limitations observed in CARL:

- ▷ **No premature saturation of preinstalled apps.** Users retain access to their inherent app connections without being overridden by others.
- ▷ **Lower redundant installations.** DTAS consistently installs fewer extra apps across all datasets compared to both preliminary and refined CARL.
- ▷ **Improved stability and fairness.** Transaction loads are distributed more evenly across applications, preventing over-reliance on a small set of popular apps.
- ▷ **Closer to ILP-optimal results.** Empirical evaluations show that DTAS nearly matches the ILP's installation count, often differing by only one or two installations in large-scale experiments.

Overall Impact. By explicitly decoupling preinstalled and extra-app allocations and employing an ordering strategy aligned with fairness and capacity constraints, DTAS achieves a more stable, trust-preserving, and capacity-efficient transaction allocation. It builds on the strengths of CARL while eliminating its key limitations, resulting in a heuristic that is highly scalable, near-optimal in performance, and well suited for large-scale UPI transaction networks.

5 Experimental Evaluations

5.1 Computational Environment

The experiments were conducted on a Linux-based system with the following specifications: *Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60 GHz*, featuring *12 physical cores and 24 threads* across two sockets, with a total of 30 MB L3 cache and two NUMA nodes. The system was equipped with *125 GiB of RAM* and *122 GiB of swap space*, and did not employ a dedicated GPU. It ran Ubuntu 22.04 LTS (64-bit). All algorithms were implemented in `Python 3.12.3` and executed using the Gurobi Optimizer (version 12.0.3) as the solver for ILP formulations. Each configuration represents a unique combination of users, their respective transaction loads, and the available UPI applications with varying capacity constraints. The ILP-based solution provides the optimal allocation and serves as the baseline for comparison. Heuristic approaches namely Capacity-Aware Reuse-first Layered Allocation, and Decoupled Two-Stage Allocation Strategy are evaluated in terms of their efficiency, scalability, and proximity to the optimal ILP solution.

To assess the effectiveness and scalability of the proposed heuristics, we conducted a series of experiments comparing **ILP**, **LP Relaxation**, CARL, and DTAS. All algorithms were evaluated on synthetic configurations of increasing transaction volumes, ranging from 10,000 to 100 million transactions. Each configuration was tested under identical conditions to ensure fairness in comparison.

5.2 Performance Metrics

We evaluate each algorithm using the following metrics:

- ▷ **Number of App Installations:** Total number of app installations required to satisfy all transactions. Lower values indicate better efficiency.

Algorithm 2 Decoupled Two-Stage Allocation Strategy (DTAS)

Initialize *apps*, set $remainingCapacity[a] \leftarrow c$, $handled[a] \leftarrow 0$
$extraApps \leftarrow \bigcup_{u \in U} preinstalled[u]$ ▷ Apps preinstalled by some user
$userApps[u] \leftarrow preinstalled[u]$
Build $users \leftarrow (u, preinstalled[u], transactions[u])$ and sort by $(|transactions|, |preinstalled|)$
Phase 1: Preinstalled Allocation
for $(u, pre, txns)$ in $users$ **do**
  $remain \leftarrow |txns|$
  for a in pre sorted by $remainingCapacity[a]$ desc **do**
    $alloc \leftarrow \min(remain, remainingCapacity[a])$
    $remainingCapacity[a] -= alloc$; $handled[a] += alloc$; $remain -= alloc$
    if $remain \leqslant 0$ **then break**
    end if
  end for
  $userRemain[u] \leftarrow remain$
end for
Phase 2: Extra App Allocation
for $(u, pre, txns)$ in $users$ **do**
  $remain \leftarrow userRemain[u]$
  if $remain \leqslant 0$ **then continue**
  end if
  (a) Reuse existing extra apps
  for a in sort$(extraApps \setminus pre$, by $remainingCapacity[a]$ desc) **do**
    $alloc \leftarrow \min(remain, remainingCapacity[a])$
    Update $remainingCapacity[a]$, $handled[a]$, $remain$, $userApps[u]$
    if $remain \leqslant 0$ **then break**
    end if
  end for
  (b) Install new extra app if needed
  if $remain > 0$ **then**
    $a^* \leftarrow \arg\max_{a \notin pre \cup extraApps} remainingCapacity[a]$
    Allocate to a^*; update $remainingCapacity[a^*]$, $handled[a^*]$, $remain$, $userApps[u]$, $extraApps$
  end if
end for
Compute total extra installations and unallocated transactions
Return: final allocations and handled transactions

- ▷ **Execution Time:** Total computational time taken to reach a feasible allocation.
- ▷ **Fairness Index:** A measure of how evenly transactions are distributed across apps (*Inverse Gini coefficient*).

Comparison of App Installations: Table 1 summarizes the total number of app installations required by each algorithm for increasing transaction sizes. The ILP provides the optimal lower bound, while the LP Relaxation closely approximates it. Among the heuristics, both CARL and DTAS achieve results close to the LP solution. Interestingly, CARL slightly outperforms DTAS for smaller instances due to its compact reuse-first ordering; however, as the dataset grows, DTAS **becomes the superior choice**—exhibiting better scalability, reduced redundant installations, and improved stability across large workloads.

Table 1: Comparison of Installations Across Different Algorithms

Transactions	ILP	LP	CARL	DTAS
10,000	5	3.78	5	5
69,519	27	26.03	29	28
483,293	90	88.99	95	91
1,274,274	261	260.32	273	261
8,858,667	1,901	1,900.40	1,967	1,902
23,357,214	4,697	4,696.06	4,855	4,697
100,000,000	***	***	6,863	6,599

Note: '***' indicates that the runtime for ILP and LP was prohibitively high, and results could not be computed within reasonable time limits.

Execution Time Comparison: Figure 3 presents the execution time of different algorithms across configurations. While ILP and LP approaches exhibit exponential growth in runtime with increasing data size, the heuristics (particularly DTAS) maintain near-linear scalability, making them viable for real-world deployment.

Fairness Index: As illustrated in Figure 4, the Gini index results were transformed into a fairness-oriented metric using the Inverse Gini Score (IGS = 1-G). DTAS achieves the highest fairness (IGS = 0.53), outperforming both ILP (0.34) and LP (0.30). This demonstrates DTAS 's superior balance in capacity utilization and more equitable transaction distribution across UPI apps.

Quantitative Results: The experiments were conducted on 1.21 million users and 100 million transactions to assess how varying application capacity limits influence allocation outcomes. As shown in Table 2, increasing app capacity sharply reduces the number of users requiring new installations—from about 2.25% at a 10% capacity limit to nearly zero at 40%. This implies that a vast majority of users can be fully served even under moderate capacity levels, and only a very small subset (less than 2.3%) faces unmet demand.

However, this increased capacity also accentuates duopoly tendencies, where a few dominant applications absorb a disproportionate share of the total transaction load. Such concentration, while improving feasibility, may negatively affect competition and resilience. Conversely, lowering app capacity reduces dominance but results in unallocated transactions—over 65 million at a 10% limit—indicating potential service degradation if strict caps are imposed. If regulatory constraints, such as NPCI's transaction thresholds per app, were enforced, the corresponding remaining transactions directly represent the volume that would go unserved, potentially leading to user friction and reduced system efficiency.

6 Conclusion

- ▷ Both CARL and DTAS perform close to the LP relaxation in terms of total installations, ensuring high-quality feasible allocations.
- ▷ DTAS consistently reduces redundant installations by decoupling preinstalled and new allocations, leading to improved fairness and capacity preservation.
- ▷ The execution time of DTAS remains nearly linear with data size, maintaining scalability and efficiency even at large transaction volumes.

Table 2: Impact of App Capacity on Remaining Demand and Unallocated Transactions

App Cap.	Users w/ Remaining Demand (%)	Unallocated Transactions
10	2.25%	65,982,913
15	1.00%	50,982,913
20	0.54%	40,418,301
25	0.29%	30,418,301
30	0.13%	20,418,301
35	0.02%	10,418,301
40	≈0%	1,525,685

- ▷ Lower app capacities limit dominance but increase unallocated transactions, underscoring the need for balanced capacity rules. Strict NPCI-style thresholds could leave millions of transactions unserved.
- ▷ Overall, DTAS offers a strong trade-off between optimality, fairness, and efficiency, making it effective for large-scale transaction allocation under realistic capacity limits.

References

[1] European Payments Council AISBL. 2024. UPI: revolutionising real-time digital payments in India. https://www.europeanpaymentscouncil.eu/news-insights/insight/upi-revolutionising-real-time-digital-payments-india. Accessed: 2024-06-26.

[2] Deepa Devassy, Divya K S, Gripsy Paul Mannickathan, Alen Biju, Aswathi T, and Devarsh R. 2025. FedUPI: Federated Learning Empowered Detection of Fraudulent UPI Transactions. In *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*. 1007–1015. doi:10.1109/ICSCSA66339.2025.11170880

[3] Muhammad Liman Gambo, Anazida Zainal, and Mohamad Nizam Kassim. 2022. A convolutional neural network model for credit card fraud detection. In *2022 International Conference on Data Science and Its Applications (ICoDSA)*. IEEE, 198–202.

[4] Jaspreet Kalra. 2024. India delays UPI market share cap in relief for Walmart backed PhonePe, Google Pay. www.reuters.com/technology/india-delays-upi-payments-market-share-cap-relief-walmart-backed-phonepe-google-2024-12-31/.

[5] N Lingareddy, Devadutta Indoria, S Deepika, Geogen George, M Keerthi Priya, et al. 2025. Enhancing Digital Payment Security: UPI Fraud Detection with Advanced Machine Learning Algorithms. In *2025 Global Conference in Emerging Technology (GINOTECH)*. IEEE, 1–7.

[6] Meghna Mittal. 2025. 80% of UPI volumes concentrated with two app providers, fintech body writes to FinMin. https://shorturl.at/LFhTN.

[7] National Payments Corporation of India (NPCI). 2025. Unified Payments Interface (UPI) Ecosystem Statistics. https://www.npci.org.in/what-we-do/upi/upi-ecosystem-statistics.

[8] Shailesh Rastogi, Chetan Panse, Arpita Sharma, and Venkata Mrudula Bhimavarapu. 2021. Unified Payment Interface (UPI): A digital innovation and its impact on financial inclusion and economic development. *Universal Journal of Accounting and Finance* 9, 3 (2021), 518–530.

[9] Akrati Saxena. 2023. RaboBank Transaction Dataset. https://github.com/akratiiet/RaboBank_Dataset. Accessed: 2025-04-12.

[10] Zixing Song, Yuji Zhang, and Irwin King. 2023. Towards fair financial services for all: A temporal GNN approach for individual fairness on transaction networks. In *Proceedings of the 32nd ACM international conference on information and knowledge management*. 2331–2341.

[11] Financial Times. 2024. India's next digital payments boom. https://www.ft.com/content/978a7a66-0850-498b-add5-2c9333e7c9a3. Accessed: 2025-04-12.

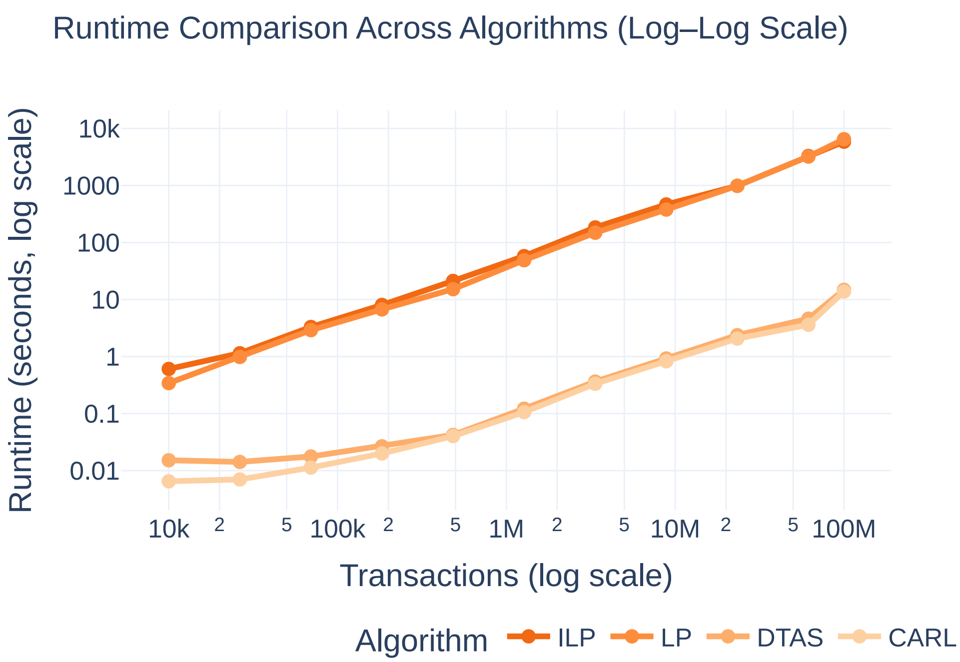

Figure 3: Execution time comparison of ILP, LP, CARL, and DTAS across increasing transaction volumes.

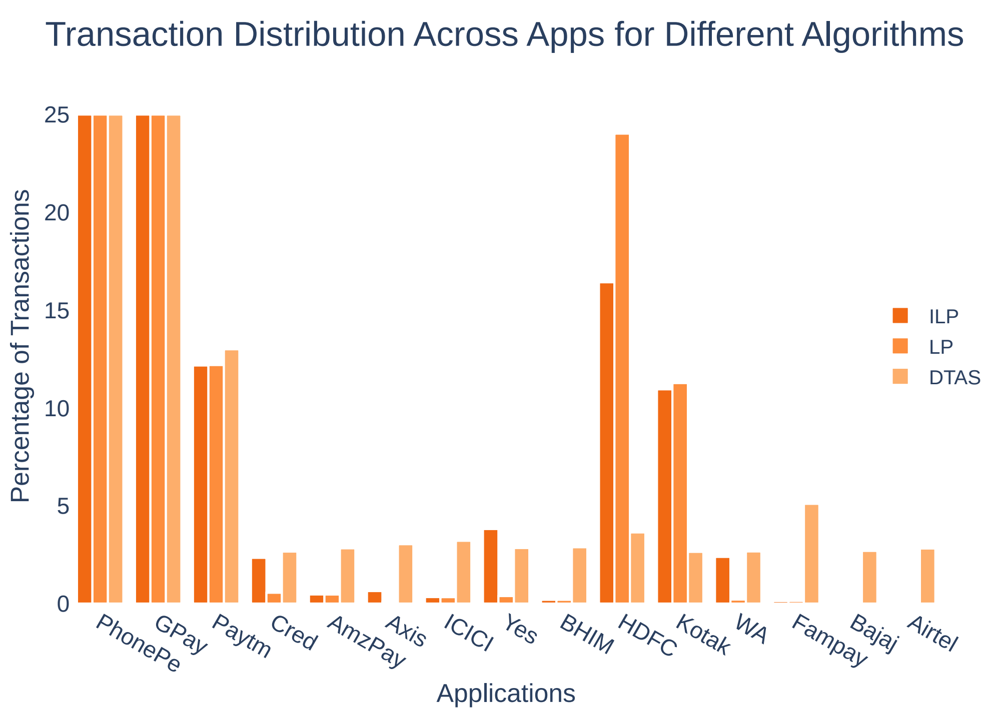

Figure 4: Transaction Allocation Across UPI Apps Using the Different Approaches on 25M Transactions.