Approximation and Hardness Results for the Parallel-Block Construction Problem

Arivarasan Karmegam
IMDEA Networks Institute, Madrid, Spain
Universidad Carlos III de Madrid, Madrid, Spain

Alexandru Popa
University of Bucharest, Bucharest, Romania.

Lucianna Kiffer
IMDEA Networks Institute, Madrid, Spain

Antonio Fernández Anta
IMDEA Software Institute, Madrid, Spain
IMDEA Networks Institute, Madrid, Spain

Abstract

Several high-throughput blockchains, including Solana, Sui, and Aptos, execute transactions in parallel across multiple cores to improve throughput and reduce latency. However, transaction conflicts induced by shared state access fundamentally couple block construction with parallel scheduling. We formalize and study the *Parallel-Block Construction* problem: given transactions with execution times, rewards, and conflict relations, select and schedule a subset of transactions on p parallel cores within a runtime (gas) budget to maximize total reward.

We provide a comprehensive analysis of the complexity and approximation landscape for this problem. When the number of cores p is part of the input, we prove strong inapproximability via a reduction from Maximum Clique: unless NP = ZPP, no polynomial-time algorithm achieves an $n^{1-\epsilon}$-approximation for any $\epsilon > 0$. For constant p and uniform processing times, we give a greedy algorithm achieving a tight $(1 - 1/e)$-approximation ratio. We further establish NP-completeness even for fixed $p = 4$ and unit-length transactions, and show that for $p = 2$ the problem admits an exact polynomial-time algorithm via a reduction to maximum-weight matching (with a cardinality constraint). In contrast, allowing heterogeneous processing times restores hardness: the problem becomes NP-complete for $p = 2$ with processing times in $\{1, 3\}$ (even with unit rewards). For $p = 2$ and processing times in $\{1, 2\}$, we design a polynomial-time $(2/3 - \delta)$-approximation algorithm (for any $\delta > 0$) via a structural decomposition and a reduction to a budgeted matching problem.

We validate our theoretical results with experiments on Ethereum mainnet execution traces. In the homogeneous setting, our $(1 - 1/e)$-approximation algorithm almost always achieves rewards above 99% of the MILP-based optimal baseline, far exceeding the guaranteed factor of $1 - 1/e = 0.63$. In the heterogeneous setting ($p = 2$, processing times in $\{1, 2\}$), the non-overlap variant achieves above 99% of the optimal across all tested configurations, confirming that the theoretical 2/3 bound is a pessimistic worst-case guarantee that does not reflect typical performance on real workloads.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Problems, reductions and completeness; Theory of computation → Parallel algorithms

Keywords and phrases Blockchain, Parallel Execution, NP-Completeness, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.AFT.2026.12

Funding This work is funded in part by MICIU/AEI /10.13039/501100011033/, ERDF, and the ESF+ under predoctoral training grant PREP2022-000373 and grant DRONAC (PID2022-140560OB-I00) and by MICIU/AEI/10.13039/501100011033 under grant CEX2024-001471-M.

Acknowledgements We would like to thank Lioba Heimbach and the DISCO group at ETH Zurich for providing the transaction traces used in our experiments section.

1 Introduction

Block builders run on multi-core CPUs, yet many platforms still execute transactions largely sequentially to preserve deterministic, order-equivalent semantics. This conservative execution model leaves substantial performance on the table: a large fraction of transactions touch disjoint parts of the state and could be processed concurrently, but any pair that overlaps on a state key with at least one write must be serialized. As a result, parallelism is constrained by *conflict structure* rather than only by available hardware. Chains that execute concurrently, such as Solana's Sealevel runtime [19, 22], Block-STM-based systems [9, 1], and object-centric designs such as Sui [5], confront exactly this constraint.

This tension is especially acute during *block construction*. A block builder assembling a block from a mempool is not only deciding how to execute transactions fast but also making an economic choice about which transactions to include under a hard runtime (or gas) budget. When execution is done in parallel, these decisions become tightly coupled: selecting a transaction can reduce feasible parallelism by introducing conflicts, while a more parallel-friendly subset can increase total reward by fitting more transactions into the same budget. A recent systems work [13] has formalized and empirically evaluated this viewpoint via the *Parallel-Block Construction* (PBC) problem, proposing MILP baselines and fast deterministic heuristics and showing that MILPs quickly become intractable while heuristics remain practical at scale.

1.1 Problem Overview

In this paper, we complement the systems perspective of Karmegam et al. [13] with a *complexity and approximation-theoretic* study of the underlying optimization problem. Understanding the formal computational boundaries of this problem is practically important; it tells block builders which regimes admit efficient, optimal, or near-optimal scheduling, and which regimes are provably intractable regardless of the algorithm used.

We assume a mempool in which each transaction τ is characterized by its execution time $\tau.t$, reward $\tau.reward$, and read/write sets $(\tau.R, \tau.W)$, which induce conflicts as defined in Section 2. The PBC problem is defined as follows: given a set *Mpool* of transactions, p identical cores, conflict constraints induced by read/write sets, and a runtime budget B, select a subset of *Mpool* and schedule it non-preemptively so that (i) conflicting transactions never overlap, (ii) the makespan is at most B, and (iii) the total reward is maximized. PBC captures the algorithmic core of parallel-block construction and exposes sharp trade-offs between reward and parallelizability. This coupling of selection and scheduling connects PBC to classical packing and scheduling-with-conflicts problems, and it exposes clique- and matching-like structure in the induced compatibility graph that we exploit in several regimes.

While our model is deliberately simplified, real mempools contain transactions with highly varied execution times, complex smart-contract interactions, and richer conflict structures than we capture here. Our goal is not to model Ethereum in full generality, but to establish the complexity-theoretic foundations of parallel-block construction by analyzing the cleanest non-trivial cases and progressively relaxing assumptions. Even in these simplified regimes, the picture is already rich: sharp phase transitions emerge between tractable and intractable cases as p or the allowed processing-time set changes by the smallest possible amount. Understanding where these boundaries lie is a necessary step toward a complete theory of conflict-aware block construction.

1.2 Related Work

Our work sits at the intersection of (i) deterministic parallel execution in blockchains, and (ii) classical scheduling and packing under conflict constraints. On the systems side, several designs demonstrate that substantial speedups are possible when parallelism is exploited; on the theory side, conflict constraints naturally lead to hard combinatorial optimization problems, motivating a careful complexity and approximation analysis.

Parallel execution in blockchain systems. A large line of systems work aims to exploit multi-core hardware while preserving deterministic semantics. Speculative execution engines such as Block-STM execute transactions optimistically in parallel and resolve conflicts via deterministic validation and re-execution against a fixed transaction order [9]. In contrast, Solana's Sealevel runtime relies on *declared access lists* (read/write accounts) to schedule non-conflicting transactions concurrently, while serializing conflicts [19, 22]. Object-centric designs adopt related ideas by using explicit access metadata to expose parallelism (e.g., at the level of objects) while preserving deterministic outcomes [5]. These works focus primarily on accelerating execution of an already determined block (typically with a fixed order), whereas our focus is on the *block construction* side: selecting transactions from the mempool and scheduling them under a hard runtime budget to maximize reward.

Validator-side optimization for block construction. Closest to our setting is recent validator-side work that formalizes and empirically studies (i) ordered-block scheduling and (ii) parallel-block construction under conflict constraints, proposing MILP formulations and fast heuristics and evaluating them on Ethereum traces [13]. Their results show that exact MILP optimization becomes quickly intractable as instance size, heterogeneity, or core count grows, while lightweight heuristics remain practical and achieve near-optimal solutions on many workloads [13]. In contrast to this empirical/systems emphasis, our contribution is to establish a detailed *complexity- and approximation-theoretic* landscape for the PBC problem, including strong inapproximability for general p, exact solvability in special regimes, and tight approximation guarantees in others.

List scheduling for precedence constraints. Deterministic execution under a fixed total order naturally induces precedence constraints, relating ordered-block execution to precedence-constrained makespan scheduling on identical machines ($P|\text{prec}|C_{\max}$). This classical problem is NP-hard in general [20], and it is a standard motivation for approximation via list scheduling, for which the celebrated $(2 - 1/p)$ performance guarantee is due to Graham [10, 11]. While our paper centers on block *construction* (where selection is also a decision variable), these scheduling foundations help explain why even executing a fixed ordered block optimally is computationally challenging, and why approximation and structural restrictions are essential.

Scheduling with conflict graphs. Our model relates to scheduling under incompatibilities represented by a *conflict graph* $G = (V, E)$. In the classical parallel-machine setting with conflicts (often written $P|G|C_{\max}$), each edge indicates a pair of jobs that cannot be assigned to the same machine, and feasibility is equivalent to m-colorability of G [14]. For unit-time jobs in slotted time, the same conflict-graph abstraction yields a direct correspondence to graph coloring: colors represent time slots, so feasible schedules correspond one-to-one with proper colorings and the chromatic number equals the minimum number of slots [15]. With heterogeneous processing times, this clean equivalence breaks, and one naturally encounters richer coloring generalizations (e.g., multicoloring variants for jobs requiring

Block length	p	Tx Lengths	Result	Reference
One transaction	Arbitrary	Homogeneous	$(n^{1-\epsilon})$-inapproximable	Corollary 8
Arbitrary	Constant	Homogeneous	$(1-1/e)$-approximation	Theorem 9
Constant	Constant	Homogeneous	Exact polynomial time	Remark 11
Arbitrary	4	Homogeneous	NP-complete	Theorem 14
Arbitrary	2	Homogeneous	Exact polynomial time	Theorem 15
Arbitrary	2	$\{1,3\}$	NP-complete	Theorem 17
Arbitrary	2	$\{1,2\}$	$\left(\frac{2}{3}-\delta\right)$-approximation	Theorem 19

Table 1 Summary of complexity and approximation results for the Parallel-Block Construction (PBC) Problem.

multiple slots) [15]. In blockchain execution, this perspective has been used to contrast *block-order-preserving* schedulers with *coloring-based* orderings derived from (approximate) colorings of the transaction conflict graph; the latter can offer substantial latency improvements depending on the conflict ratio and workload [17].

Packing/knapsack under conflicts. On the selection side, PBC generalizes several packing problems. Multiple knapsack (MKP) is a canonical model for selecting profitable items subject to multiple capacity constraints and admits a PTAS in important regimes [6]. Adding conflicts between items yields the knapsack problem with conflict graphs, which is strongly NP-hard in general and has been studied via exact methods and pseudo-polynomial/FPTAS results on restricted graph classes [16]. Our setting further couples such conflict-aware selection with parallel-machine scheduling under a makespan budget, which goes beyond classical knapsack variants and helps explain the need for new complexity classifications and approximation techniques.

Clique hardness and matching-based optimization. Our hardness results leverage the well-known inapproximability of Maximum Clique established by Håstad, under the standard assumption NP $\neq$ ZPP [12]. On the algorithmic side, our polynomial-time solvability results for restricted regimes exploit matching structure; maximum-weight matching is solvable in polynomial time via Edmonds' algorithm, and parametric/budgeted variants are a classical tool for enforcing cardinality-type constraints [7, 18]. These techniques become relevant in our $p = 2$ regime, where schedules decompose into pairings and singletons, enabling exact optimization.

1.3 Our Contributions

We provide a comprehensive complexity and approximation landscape for the Parallel-Block Construction problem.

Homogeneous Transactions (Uniform Processing Times)

Strong Inapproximability for Arbitrary p. When the number of cores p is part of the input, we prove that the problem is extremely hard to approximate. Via a reduction from Maximum Clique, we show that, unless NP = ZPP, no polynomial-time algorithm achieves an $n^{1-\varepsilon}$-approximation for any $\varepsilon > 0$.

$(1-1/e)$-Approximation for Constant p. When p is constant, and all transactions have identical processing time, we design a greedy round-based algorithm that achieves a tight

$(1 - 1/e)$-approximation ratio. The algorithm repeatedly selects a maximum-reward clique of size at most p in the compatibility graph.

NP-Completeness for $p = 4$. Even when $p = 4$ and all transactions have unit processing time, we prove that the decision version of the PBC problem is NP-complete via a reduction from 3D-Matching. Thus, fixing the number of cores does not eliminate computational hardness.

Exact Polynomial-Time Algorithm for $p = 2$. For $p = 2$ and uniform processing times, we show that the problem admits an exact polynomial-time solution. The key idea is a reduction to maximum-weight matching with a cardinality constraint, solvable using parametric matching techniques.

Heterogeneous Transactions

Hardness with Heterogeneous Processing Times. Allowing heterogeneous processing times immediately restores hardness: for $p = 2$ and processing times in $\{1, 3\}$ (with unit rewards), the problem becomes NP-complete.

$(2/3 - \delta)$-Approximation for $p = 2$, Lengths $\{1, 2\}$. For $p = 2$ and processing times in $\{1, 2\}$, we design a polynomial-time $(2/3 - \delta)$-approximation algorithm for any $\delta > 0$. The algorithm reduces a structured non-overlap variant to a budgeted maximum-profit matching problem and uses a structural decomposition of optimal schedules to bound the loss.

Table 1 summarizes the complexity results described. Taken together, our results reveal sharp phase transitions: small changes in the number of cores or in the transaction lengths dramatically alter computational complexity.

1.4 Organization of the Paper

Section 2 introduces notation and formal definitions. Section 3 describes the proofs and approximations for cases which handle only homogeneous transactions, i.e., transactions with the same execution time. Section 3.1 proves strong inapproximability when p is part of the input. Section 3.2 presents the $(1 - 1/e)$-approximation algorithm for constant p. Section 3.3 establishes NP-completeness for $p = 4$. Section 3.4 gives the exact polynomial-time algorithm for $p = 2$ in the uniform case. Section 4 describes the proofs and approximations for heterogeneous transactions. Section 4.1 proves NP-completeness for $p = 2$ with lengths in $\{1, 3\}$. Section 4.2 presents the $(2/3 - \delta)$-approximation algorithm for $p = 2$ with lengths in $\{1, 2\}$. In Section 5, Subsection 5.1 discusses implementation aspects and practical considerations, and Subsection 5.2 discusses the results. Section 6 concludes the paper.

2 Preliminaries

Transactions. We assume that a block builder has an execution environment that can process up to p transactions in parallel, where p is the number of cores of the block builder. Each transaction τ has a *Priority Fee*, denoted $\tau.tip$. Computation is priced in units of *gas*, an abstract measure of computational effort. When executed, τ consumes a certain amount of gas $\tau.gasU$, and yields a reward, $\tau.reward = \tau.gasU \cdot \tau.tip$, which goes to the block builder. The $\tau.tip$ could be 0, so $\tau.reward \geq 0$. We also assume that the time to execute a transaction τ is proportional to its gas consumption. Let us denote the execution time of a transaction τ

as $\tau.t$. Transactions are non-preemptive. We denote by $\tau.R$ and $\tau.W$ the subset of state keys a transaction τ reads and writes, respectively (Here "keys" refers to accounts or contracts by address).

Conflicts. We say that two transactions τ_i and τ_j conflict if and only if they access a common state key and at least one of them writes to it. Formally, $\text{conflict}(\tau_i, \tau_j) \equiv \big((\tau_i.W \cap \tau_j.W) \cup (\tau_i.W \cap \tau_j.R) \cup (\tau_i.R \cap \tau_j.W) \neq \varnothing\big)$.

Schedules and the PBC problem. Then, we can define:

▶ **Definition 1** (Schedule). *A schedule S of a set of transactions T on p cores assigns to each transaction in T a core and a starting time to run. Given a schedule S of a set of transactions T on p cores, its* makespan*, denoted $S.mspan$, is the time to complete the execution (without preemption) of all transactions in T as defined by S.*

We now formalize the problem we study.

▶ **Problem 2** (Parallel-Block Construction (PBC), optimization version).
Input: *a mempool $Mpool = \{\tau_1, \ldots, \tau_n\}$, where each transaction τ_i has an execution time $\tau_i.t \in \mathbb{Z} > 0$, a reward $\tau_i.reward \in \mathbb{Q} \geq 0$ and read/write sets $(\tau_i.R, \tau_i.W)$ over a universe of state keys; a number of cores $p \in \mathbb{Z} > 0$; and a runtime limit $B \in \mathbb{Z} > 0$. All numbers are encoded in binary.*
Output: *a subset $T \subseteq Mpool$ and a schedule S of T on p cores (Definition 1) such that (i) no two transactions assigned to the same core overlap in time, (ii) no two conflicting transactions overlap in time, and (iii) $S.mspan \leq B$.*
Objective: *maximize $T.reward = \sum_{\tau \in T} \tau.reward$.*

▶ **Problem 3** (PBC-D, decision version).
Input: *an instance of Problem 2 together with a target reward $K \in \mathbb{Q} \geq 0$.*
Question: *does there exist a pair (T, S) feasible for Problem 2 with $T.reward \geq K$?*

3 Homogeneous Transactions

3.1 Hardness of Approximation for Arbitrary p

In this section, we show that, unless $NP = ZPP$, Problem 2 (PBC) is hard to approximate to a factor of $n^{1-\epsilon}, \forall \epsilon > 0$. The assumption $NP \neq ZPP$ is standard in hardness-of-approximation results. Recall that ZPP is the class of problems solvable by zero-error randomized algorithms with expected polynomial running time [2]. Hence, $NP = ZPP$ would imply that every NP-complete problem admits a zero-error randomized algorithm with expected polynomial running time, which is widely believed to be unlikely.

The proof is done via an approximation-preserving reduction from the maximum clique problem that we define next.

▶ **Definition 4** (Clique). *Let $G = (V, E)$ be an undirected graph. A subset $C \subseteq V$ is a* clique *if every pair of distinct vertices in C is adjacent in G, i.e., $\forall\, u, v \in C,\ u \neq v \ \Rightarrow\ (u, v) \in E$. (Equivalently, the subgraph of G induced by C is complete.)*

▶ **Problem 5** (Maximum Clique Problem). *Let $G = (V, E)$ be an undirected graph. Given G, finding a clique $C \subseteq V$ of maximum cardinality, i.e., $C \in \arg\max\{\, |C'| : C' \subseteq V,\ C' \text{ is a clique in } G \,\}$, is called the* Maximum Clique *problem.*

▶ **Theorem 6** (Håstad [12]). *For any $\epsilon > 0$, unless $NP = ZPP$, there is no polynomial-time algorithm that approximates* Max-Clique *within a factor of $n^{1-\epsilon}$.*

We show next the reduction from Maximum Clique to Problem 2.

▶ **Theorem 7.** *If there exists a polynomial time c-approximation algorithm for Problem 2, then the Maximum Clique problem admits a polynomial time c-approximation algorithm.*

Proof. Let $G = (V, E)$ be an instance of Maximum Clique, where $V = \{v_1, \ldots, v_n\}$. We construct an instance $\mathcal{I}(G)$ of Problem 2 as follows.

Let $T = \{\tau_1, \ldots, \tau_n\}$ be the set of transactions corresponding to V. For each vertex $v_i \in V$, create a corresponding transaction τ_i with reward $\tau_i.reward = 1$ and processing time $\tau_i.t = 1$. For every non-edge, i.e., $(v_i, v_j) \notin E$, introduce a conflict between τ_i and τ_j as follows. Define a different item c_{ij} for each pair of tasks τ_i and τ_j, and include c_{ij} in both $\tau_i.W$ and $\tau_j.W$. We emphasize that c_{ij} is unique to the pair (i, j) and is not included in the read/write sets of any other transaction. Thus, two transactions are non-conflicting if and only if their corresponding vertices are adjacent in G. We set the number of cores to $p = n$ and the runtime limit to $B = 1$, so that any feasible schedule must execute all the transactions in a single round.

We now prove that for any integer k, G contains a clique of size k *if and only if* the constructed instance admits a feasible solution of total reward k.

(⇒) If direction. Let $C \subseteq V$ be a clique of size k in G. By definition of a clique, for any distinct $v_i, v_j \in C$ we have $(v_i, v_j) \in E$, hence τ_i and τ_j do not conflict. Therefore, all transactions $\{\tau_i : v_i \in C\}$ can be scheduled in parallel in a single round, yielding a feasible schedule with a reward k.

(⇐) Only-if direction. Let $T' \subseteq T$ be any feasible solution of reward k for Problem 2. Since every transaction has reward 1, $|T'| = k$. We define the corresponding vertex set $C = \{v_i : \tau_i \in T'\}$. Feasibility implies that no two transactions in T' conflict, and by construction, this holds if and only if for all distinct $\tau_i, \tau_j \in T'$ we have $(v_i, v_j) \in E$. Since every pair of distinct vertices in C is adjacent in G, the set C is a clique of size k in G. ◀

Given the result in Theorem 6, we have the following corollary.

▶ **Corollary 8.** *For any $\epsilon > 0$, unless $NP = ZPP$, there is no polynomial-time algorithm that approximates Problem 2 within a factor of $n^{1-\epsilon}$.*

3.2 $(1 - 1/e)$-Approximation for Constant p

We show in Section 3.1 that Problem 2 is hard to approximate if $p = n$, so p is part of the input. In this section, we show a polynomial time approximation algorithm that achieves a factor of $(1 - 1/e)$, if p is a constant and all the transactions are homogeneous, i.e., they take the same amount of time $\tau.t$.

We divide time into rounds of $\tau.t$ time each. The block builder can execute at most p transactions in parallel per round. Let $R = \lfloor B/\tau.t \rfloor$ be the maximum number of rounds possible in the block, where B is the runtime limit of the block. Let T denote the set of all transactions in the mempool, and we define the *compatibility graph* $G = (T, E)$ as follows. The vertex set is the set of transactions T, and an edge $\{\tau, \tau'\} \in E$ indicates that τ and τ' do not conflict and therefore can be executed in parallel. For a subset $T' \subseteq T$, we write $T'.reward = \sum_{\tau \in T'} \tau.reward$.

Algorithm 1 Greedy $(1-1/e)$-approximation for Problem 2 (constant p, homogeneous transactions)

1. **Input:** Set of transactions T, runtime budget B, compatibility graph $G = (T, E)$.
2. **Constant:** Number of cores p.
3. Set $R \leftarrow \lfloor B/\tau.t \rfloor$.
4. Set $T_0 \leftarrow T$.
5. Initialize the schedule $S \leftarrow \emptyset$.
6. For each round $i = 1, 2, \ldots, R$ do:
   a. In the induced subgraph $G[T_{i-1}]$, choose a clique $C_i \in \arg\max\{C.reward : (C \text{ is a clique in } G[T_{i-1}]) \wedge (|C| \leq p)\}$.
   b. If $C_i = \emptyset$, then **break** the loop.
   c. Schedule all transactions in C_i in parallel, in round i, i.e., add (i, τ) to S for every $\tau \in C_i$.
   d. Update the remaining transactions set: $T_i \leftarrow T_{i-1} \setminus C_i$.
7. **Output:** schedule S.

The algorithm follows a greedy strategy, has R rounds, and at each round it selects a maximum-reward clique C_i of size at most p in the current compatibility graph and schedules all transactions in that clique in parallel. (For convenience, we use C_i to denote the clique and its vertices. Its meaning is clear from the context.)

▶ **Theorem 9.** *Algorithm 1 is a polynomial time* $(1 - 1/e)$*-approximation and runs in time* $O(n^p R)$.

Proof. We first bound the running time and then prove the approximation ratio.

Running time. For each round $i \in \{1, \ldots, R\}$, recall that Algorithm 1 maintains the remaining transaction set $T_{i-1} \subseteq T$ and chooses a clique $C_i \subseteq T_{i-1}$ by maximizing the total reward over all cliques $C \subseteq T_{i-1}$ of size at most p in the compatibility graph $G[T_{i-1}]$. Since p is a constant, the number of candidate subsets is $\sum_{k=0}^{p} \binom{|T_{i-1}|}{k} = O(n^p)$, where $n = |T|$. For each candidate C, we can check in constant time (depending only on p) whether C is a clique in G and compute $C.reward$. Thus, one iteration of this loop takes $O(n^p)$ time. As there are at most R rounds, the total running time is $O(n^p R)$.

Approximation ratio. Let OPT denote the maximum total reward of any feasible schedule that uses at most R rounds. Let $(S_1, \ldots, S_R)$ be the schedule produced by Algorithm 1. Recall that C_i is the maximum-reward clique of size at most p chosen by Algorithm 1 in round i, so $S_i = C_i$. For each $i \in \{0, \ldots, R\}$, let $A_i = \bigcup_{j=1}^{i} S_j$ be the set of transactions scheduled by the algorithm in the first i rounds, with the convention $A_0 = \emptyset$. We also write $A_i.reward = \sum_{\tau \in A_i} \tau.reward$. Consider a fixed optimal schedule with rounds $O_1, \ldots, O_R$, where each O_ℓ is a clique of size at most p in the compatibility graph and $\sum_{\ell=1}^{R} O_\ell.reward = OPT$.

We first prove a key lemma that bounds the additional reward obtained by each greedy round of Algorithm 1 in terms of the remaining optimal reward.

▶ **Lemma 10.** *For every* $i = 1, \ldots, R$, $A_i.reward - A_{i-1}.reward \geq \frac{OPT - A_{i-1}.reward}{R}$.

Proof. Since rewards are nonnegative and $T_{i-1} = T \setminus A_{i-1}$, the reward of the transactions of the optimal schedule that remain available at round i is at least the optimal reward minus the reward already collected by the greedy schedule.

Formally, let $O = \bigcup_{\ell=1}^{R} O_\ell$ be the set of transactions scheduled by the fixed optimal solution. Then $O.reward = \sum_{\ell=1}^{R} O_\ell.reward = OPT$, and

$$\begin{aligned}(O \cap T_{i-1}).reward &= (O \setminus A_{i-1}).reward = O.reward - O \cap A_{i-1}.reward \\ &\geq O.reward - A_{i-1}.reward = OPT - A_{i-1}.reward.\end{aligned}$$

Moreover, $\bigcup_{\ell=1}^{R}(O_\ell \cap T_{i-1}) = O \cap T_{i-1}$, so

$$\Big(\bigcup_{\ell=1}^{R}(O_\ell \cap T_{i-1})\Big).reward \geq OPT - A_{i-1}.reward.$$

For each $\ell \in \{1, \ldots, R\}$ let $O'_\ell = O_\ell \cap T_{i-1}$. Then,

$$\sum_{\ell=1}^{R} O'_\ell.reward \;\geq\; OPT - A_{i-1}.reward.$$

Hence there exists an index $\ell^* \in \{1, \ldots, R\}$ such that

$$O'_{\ell^*}.reward \;\geq\; \frac{OPT - A_{i-1}.reward}{R}.$$

By construction, $O'_{\ell^*} \subseteq T_{i-1}$ and O'_{ℓ^*} is a clique of size at most p in G, so O'_{ℓ^*} is a feasible choice in round i. Since Algorithm 1 selects a clique C_i maximizing $C.reward$ among all cliques $C \subseteq T_{i-1}$ with $|C| \leq p$, we have

$$C_i.reward \;\geq\; O'_{\ell^*}.reward \;\geq\; \frac{OPT - A_{i-1}.reward}{R}.$$

Moreover, by construction $C_i \subseteq T_{i-1} = T \setminus A_{i-1}$, so C_i is disjoint from A_{i-1} and therefore $A_i.reward - A_{i-1}.reward = C_i.reward$. Combining the two inequalities yields the claim. ◀

We now unroll the inequality of Lemma 10. Define $\Delta_i = OPT - A_i.reward$ for $i = 0, \ldots, R$. Lemma 10 is equivalent to $A_i.reward - A_{i-1}.reward \;\geq\; \frac{\Delta_{i-1}}{R}$, which we can rewrite as

$$\Delta_i = OPT - A_i.reward \leq OPT - \left(A_{i-1}.reward + \frac{\Delta_{i-1}}{R}\right) = \Delta_{i-1}\left(1 - \frac{1}{R}\right)$$

Thus, by induction on i,

$$\Delta_i \;\leq\; \left(1 - \frac{1}{R}\right)^i OPT \qquad \text{for all } i = 0, \ldots, R.$$

In particular, for $i = R$ we obtain

$$OPT - A_R.reward = \Delta_R \;\leq\; \left(1 - \frac{1}{R}\right)^R OPT,$$

and therefore

$$A_R.reward \;\geq\; \left(1 - \left(1 - \frac{1}{R}\right)^R\right) OPT.$$

Using the standard inequality $(1 - 1/R)^R \leq 1/e$, we finally obtain $A_R.reward \;\geq\; (1 - 1/e)\, OPT$. Hence, Algorithm 1 achieves an approximation ratio of $(1 - 1/e)$. Combining the running-time analysis with the approximation guarantee completes the proof. ◀

▶ Remark 11. When both p and $R = \lfloor B/\tau.t \rfloor$ are constants, Problem 2 is solvable in $O(n^{pR} \cdot Rp^2)$ time by exhaustive enumeration over all ordered sequences of R disjoint cliques of size at most p. There are $O(n^p)$ candidate cliques per round, disjointness across rounds reduces the effective pool by at most pR transactions, and the feasibility of each candidate clique can be checked in $O(p^2)$ time. The NP-hardness results in this work both require R to grow with the input and do not apply when R is fixed.

3.3 NP-Completeness for $p = 4$

In this section, we prove that Problem 3 (PBC-D) is NP-complete even when the number of cores of the block builder is fixed at $p = 4$, and all transactions are homogeneous (i.e., all have identical execution times). It is immediate that PBC-D is in NP: a certificate is a schedule S, and we can in polynomial time check that the rounds in S are conflict-free, compute its total reward and its makespan, and compare them to K and B, respectively. We prove our result using a polynomial reduction from the 3D-Matching problem, which we define below.

▶ **Definition 12** (3D-Matching). *Let V_1, V_2, V_3 be three pairwise disjoint sets of equal size n, and let $E \subseteq V_1 \times V_2 \times V_3$ be a set of hyperedges, where each hyperedge $e = (v_1, v_2, v_3)$ contains exactly one vertex from each of the three sets V_1, V_2, V_3. Given an integer k, the* 3D-Matching *problem asks whether there exists a subset $E' \subseteq E$ such that $|E'| = k$ and such that every two distinct hyperedges in E' are vertex-disjoint: $\forall e, e' \in E', e \neq e' \implies e \cap e' = \emptyset$.*

It is well known that 3D-matching is NP-complete [8].

▶ **Theorem 13.** *For $p = 4$ and homogeneous transactions, PBC-D is NP-hard.*

Proof. Let $V = V_1 \cup V_2 \cup V_3$ be the vertex set of a 3D-matching instance, where V_1, V_2, V_3 are pairwise disjoint and $E \subseteq V_1 \times V_2 \times V_3$ is the set of hyperedges and an integer k. We construct an instance of PBC-D as follows.

- For each vertex $v \in V$ we construct a vertex-transaction τ_v with $\tau_v.t = 1$ and $\tau_v.reward = 1$.
- For each hyperedge $e = (v_1, v_2, v_3) \in E$ we construct a hyperedge-transaction τ_e with $\tau_e.t = 1$ and $\tau_e.reward = 1$.

We now define conflicts between these transactions. Let us define the sets $T_r = \{\tau_v : v \in V_r\}$, for $r \in \{1, 2, 3\}$. Then, we make all vertex-transactions in set T_r pairwise conflicting, by defining items c_r, and for each $\tau \in T_r$ including c_r in $\tau.W$. Thus, in any round, at most one transaction from each set V_r, $r \in \{1, 2, 3\}$, can be scheduled.

All hyperedge-transactions $\{\tau_e : e \in E\}$ are pairwise conflicting. Moreover, for every hyperedge $e = (v_1, v_2, v_3) \in E$, we make τ_e conflict with every vertex-transaction except its three endpoint vertex-transactions $\tau_{v_1}, \tau_{v_2}, \tau_{v_3}$ as follows. For each vertex $u \in V \setminus \{v_1, v_2, v_3\}$, introduce a fresh state key $c_{e,u}$ and include $c_{e,u}$ in both $\tau_e.W$ and $\tau_u.W$. Hence, τ_e conflicts with τ_u for every $u \notin e$, while this construction does not introduce additional conflicts among vertex-transactions in different partitions and in any feasible round, τ_e can be scheduled together only with τ_{v_1}, τ_{v_2} and τ_{v_3}.

We set the number of cores to $p = 4$ and the gas budget to $B = k$. Since all transactions have unit execution time and there is no preemption, the makespan equals the number of rounds used by the schedule. We have to show that the hypergraph has k independent hyperedges *if and only if* there exists a solution to Problem 3 on the corresponding instance of reward precisely $4k$.

(⇒) If direction. Suppose the 3D-matching instance has a matching $E' \subseteq E$ of size k. Let S denote the schedule we construct below. For each hyperedge $e = (v_1, v_2, v_3) \in E'$, assign a distinct round $i_e \in \{1, \ldots, k\}$ and schedule the four transactions $\{\tau_{v_1}, \tau_{v_2}, \tau_{v_3}, \tau_e\}$ in parallel during round i_e. This schedule is feasible because:

- Within each round i_e, the three vertex-transactions $\tau_{v_1}, \tau_{v_2}, \tau_{v_3}$ belong to different partitions V_1, V_2, V_3 respectively, so they do not conflict with each other.
- By construction, τ_e conflicts only with transactions other than $\tau_{v_1}, \tau_{v_2}, \tau_{v_3}$, so it can be scheduled together with exactly these three transactions.
- Since E' is a matching, the vertex sets $\{v_1, v_2, v_3\}$ for distinct $e \in E'$ are disjoint. Thus, no vertex-transaction τ_v appears in more than one round.

The schedule uses k rounds of length 1 (so $S.mspan = k = B$), uses 4 transactions per round on $p = 4$ cores, and has total reward $4k$.

(⇐) Only-if direction. Suppose there exists a feasible schedule S for Problem 3 with total reward exactly $4k$ and makespan $S.mspan \leq B = k$. Since each transaction has a reward 1 and there are exactly $4k$ transaction-slots available ($p \cdot B = 4k$), the schedule must use exactly 4 distinct transactions on all 4 cores in every one of the k rounds. To reach exactly 4 transactions per round, each round i must contain exactly:

- One vertex-transaction from each V_r: $\tau_{v_1^i} \in \{\tau_{v_1} : v_1 \in V_1\}$, $\tau_{v_2^i} \in \{\tau_{v_2} : v_2 \in V_2\}$, $\tau_{v_3^i} \in \{\tau_{v_3} : v_3 \in V_3\}$.
- One hyperedge-transaction τ_e and by construction, τ_e is compatible with $\{\tau_{v_1^i}, \tau_{v_2^i}, \tau_{v_3^i}\}$ if and only if $e = (v_1^i, v_2^i, v_3^i) \in E$.

Thus each round i corresponds to a hyperedge $e_i = (v_1^i, v_2^i, v_3^i) \in E$. Let $E' = \{e_1, \ldots, e_k\} \subseteq E$. Since all $4k$ transactions across all rounds are distinct (no transaction is reused), no vertex-transaction τ_v appears in more than one round, so the vertex sets $\{v_1^i, v_2^i, v_3^i\}$ for $i = 1, \ldots, k$ are pairwise disjoint. Therefore, E' is a matching of size k.

Polynomial size and time. The reduction can be constructed in polynomial time. It creates $|V|$ vertex-transactions and $|E|$ hyperedge-transactions. The conflict encoding uses a constant number of global keys (c_1, c_2, c_3, c_h), and for each hyperedge $e \in E$ it introduces at most $|V| - 3$ fresh keys $c_{e,u}$ (one for each non-endpoint vertex u), for a total of $O(|E| \cdot |V|)$ keys. Thus, the size of the constructed PBC instance and the time needed to build it are polynomial in the size of the 3D-Matching instance. ◀

▶ **Theorem 14.** *For $p = 4$ and homogeneous transactions, PBC-D is NP-complete.*

Proof. The problem is in NP, as argued at the beginning of this section and for $p = 4$ and homogeneous transactions, it is NP-hard by Theorem 13 via reduction from 3D-Matching. ◀

3.4 Exact Polynomial Time Algorithm for $p = 2$

In this section, we present an exact polynomial time algorithm for Problem 2 for $p = 2$, and all transactions are homogeneous, i.e., they have identical execution time $\tau.t$. In this setting, the runtime budget B bounds the number of rounds by $\lfloor B/\tau.t \rfloor$, and in each round we can schedule at most two transactions in parallel. Since every non-idle round contains at least one transaction and no transaction is scheduled twice, at most n rounds of any schedule are non-idle, so we may work with $R = \min\{\lfloor \frac{B}{\tau.t} \rfloor, n\}$ rounds without excluding any optimal schedule. The cap is what makes the construction polynomial: B is encoded in binary, so $\lfloor B/\tau.t \rfloor$ alone may be exponential in the input size, whereas $R \leq n$.

Algorithm 2 Exact algorithm for $p = 2$ for the homogeneous case of Problem 2

1. Construct the following weighted graph $G' = (V', E')$:
   - For each transaction $\tau_i \in \mathit{Mpool}$, create a corresponding vertex $\tau_i \in V'$.
   - If two transactions τ_i and τ_j *do not conflict*, add an edge (τ_i, τ_j) in G' with weight $\tau_i.\mathit{reward} + \tau_j.\mathit{reward}$.
   - For each transaction τ_i, add a dummy vertex τ_i' in G' and an edge (τ_i, τ_i') with weight $\tau_i.\mathit{reward}$. (This represents scheduling τ_i alone in a round.)
   - Add $2R$ additional dummy vertices $d_1, \ldots, d_{2R} \in V'$ and, for each $r = 1, \ldots, R$, add an edge $(d_{2r-1}, d_{2r}) \in E'$ of weight 0. (This represents an unused/idle round.)
2. Compute a maximum-weight matching in G' consisting of exactly R edges.
3. Convert each matching edge into a round:
   - $(\tau_i, \tau_j) \to$ schedule τ_i and τ_j in parallel,
   - $(\tau_i, \tau_i') \to$ schedule τ_i alone,
   - $(d_{2r-1}, d_{2r}) \to$ idle round (execute nothing).

   Finally, remove all idle rounds.

The key idea is to reduce this special case to a maximum-weight matching problem on a carefully constructed graph $G' = (V', E')$. Algorithm 2 shows the details. We assume rewards $\tau.\mathit{reward}$ are integers. Since the runtime budget B only imposes an upper bound on the makespan, an optimal solution may use fewer than R rounds, leaving part of the budget unused. We enforce a matching of exactly R edges so that each edge corresponds to one of the R available rounds; unused rounds are represented by 0-weight idle-round edges. These are proof artifacts and are not executed in the final schedule. Discarding idle rounds yields a feasible schedule for the original instance with the same total reward. We now show that Algorithm 2 is correct and runs in polynomial time.

▶ **Theorem 15.** *For $p = 2$ and homogeneous transactions, Algorithm 2 computes an optimal solution to Problem 2 in polynomial time.*

Proof. Step 1 runs in $O(n^2)$ time, since $R \leq n$ and the construction adds $O(n)$ dummy vertices and edges.

Implementing Step 2 in polynomial time. Let m denote the maximum cardinality of a matching in G'. By construction, $m \geq R$ since the R disjoint edges (d_{2r-1}, d_{2r}) form a matching of size R.

We compute a maximum-weight matching of *exactly* R edges in G' in polynomial time as follows. For an integer parameter λ, define modified edge weights $w_\lambda(e) = w(e) - \lambda$. Let M_λ denote a maximum-weight matching in G' under weights w_λ. White [21] shows that for every $k \leq m$ there exists a value λ such that *some* maximum-weight matching under w_λ has cardinality exactly k, and gives a polynomial-time algorithm (making only polynomially many calls to a maximum-weight matching routine) to find such a matching. Therefore, we can compute in polynomial time a maximum-weight matching of cardinality exactly R. Each subroutine call to compute a maximum-weight matching can be implemented in polynomial time using Edmonds' weighted matching algorithm (see also standard textbook treatments [18]).

Correctness: schedule ↔ matching. *(⇒)* Given any feasible schedule of reward W, at most n of its rounds are non-idle; discarding idle rounds and padding with $R - t$ idle rounds, where $t \leq R$ is the number of non-idle rounds, yields exactly R rounds. Construct a set of R edges in G' by:

- if a round schedules τ_i and τ_j together, include edge (τ_i, τ_j);
- if a round schedules only τ_i, include edge (τ_i, τ_i');
- if a round is idle, include one distinct idle-round edge (d_{2r-1}, d_{2r}).

Since no transaction is scheduled more than once and all idle-round edges are disjoint, these edges form a matching of size R. Its total weight is exactly W.

(⇐) Given a matching M of size R and weight W, construct an R-round schedule by interpreting edges as follows:

- each edge (τ_i, τ_j) corresponds to scheduling two non-conflicting transactions τ_i and τ_j in parallel;
- each edge (τ_i, τ_i') corresponds to scheduling τ_i alone;
- each edge (d_{2r-1}, d_{2r}) corresponds to an idle round.

Since M is a matching, no vertex τ_i appears in more than one edge, hence no transaction is scheduled more than once. Removing idle rounds preserves feasibility and does not change the total reward, so the resulting schedule is feasible and has reward W.

Therefore, Algorithm 2 computes an optimal schedule in polynomial time. ◀

4 Heterogeneous Transactions

4.1 NP-Completeness for $p = 2$, $\tau.t \in \{1, 3\}$

In this section, we show that Problem 3 (PBC-D) is NP-complete even when restricted to $p = 2$, unit rewards ($\tau.reward = 1\ \forall \tau$), and execution times $\tau.t \in \{1, 3\}$. The proof uses a reduction from 3D-Matching, similar to that of Section 3.3, but adapted to exploit the different execution times.

▶ **Theorem 16.** *PBC-D is NP-hard for $p = 2$, unit rewards, and execution time $\tau.t \in \{1, 3\}$.*

Proof. We reduce from 3D-Matching. Let (V_1, V_2, V_3, E) be a 3D-Matching instance with $|V_1| = |V_2| = |V_3| = n$ and $|E| = m$, where $V = V_1 \cup V_2 \cup V_3$. We construct an instance $\mathcal{I}$ of PBC-D as follows.

1. **Transactions.** Create $3n + m$ transactions, each with reward 1.
   - For each vertex $v \in V$, create a *vertex-transaction* τ_v with $\tau_v.t = 1$.
   - For each hyperedge $e = (v_1, v_2, v_3) \in E$, create a *hyperedge-transaction* τ_e with $\tau_e.t = 3$.
2. **Conflicts.** Define three conflict types:
   - All vertex-transactions $\{\tau_v : v \in V\}$ are *pairwise conflicting*.
   - All hyperedge-transactions $\{\tau_e : e \in E\}$ are *pairwise conflicting*.
   - For each hyperedge $e = (v_1, v_2, v_3) \in E$, τ_e conflicts with *every transaction except* $\tau_{v_1}, \tau_{v_2}, \tau_{v_3}$.
3. **Parameters.** Set $p = 2$ cores, runtime budget $B = 3k$, and $K = 4k$.

We prove that the 3D-Matching instance has a matching of size k if and only if $\mathcal{I}$ admits a feasible schedule of reward at least $4k$.

(⇒) If direction. Suppose $E' \subseteq E$ is a matching of size k. For each $e = (v_1, v_2, v_3) \in E'$, construct a round $i_e \in \{1, \ldots, k\}$ scheduling:

- Core 1: $\tau_{v_1}, \tau_{v_2}, \tau_{v_3}$ in *series* (total length $1 + 1 + 1 = 3$),
- Core 2: τ_e (length 3).

This schedule uses k rounds of length 3, so $S.mspan = 3k \leq B$. It is feasible because:

- The three vertex-transactions on Core 1 are executed in series; hence, although mutually conflicting, they do not overlap in time and therefore do not violate feasibility.
- By construction, τ_e is compatible exactly with $\tau_{v_1}, \tau_{v_2}, \tau_{v_3}$.
- E' is a matching, so vertex sets $\{v_1, v_2, v_3\}$ for each $e \in E'$ are pairwise disjoint; no transaction repeats.

The schedule uses $4k$ transactions, yielding a reward $4k$.

(⇐) Only-if direction. Suppose $\mathcal{I}$ admits a feasible schedule S with $S.mspan \leq 3k$ and $S.reward \geq 4k$.

By conflict rules, no round can schedule:

- Two vertex-transactions in parallel (since all vertex-transactions are pairwise conflicting).
- Two hyperedge-transactions in parallel (since all hyperedge-transactions are pairwise conflicting).

We first upper bound the total number of transactions that can be executed within a makespan $3k$. Since vertex-transactions are pairwise conflicting, at any time at most one vertex transaction can be executed across both cores; hence, within time $3k$, the schedule can execute at most $3k$ vertex-transactions in total. Moreover, since hyperedge-transactions are pairwise conflicting and each has length 3, within a makespan $3k$, the schedule can execute at most k hyperedge-transactions. Therefore, within a makespan $3k$ any feasible schedule can execute at most $3k + k = 4k$ transactions.

Since $S.reward \geq 4k$ and all rewards are 1, S must execute exactly $4k$ transactions; hence it executes exactly $3k$ vertex-transactions and exactly k hyperedge-transactions, and moreover $S.mspan = 3k$.

Now, we define a *round* to be the time interval during which one hyperedge-transaction is executed. Since hyperedge-transactions have length 3 and are pairwise conflicting, these k hyperedge-transactions define k disjoint rounds of length 3 that cover the entire time horizon $[0, 3k)$. Since the total processing time of the k hyperedge-transactions is $3k$, there can be no idle time between them (otherwise they would not all fit by time 3k). In each round, because at most one vertex can run at any time and we must execute $3k$ vertices overall, exactly three vertex-transactions are executed on the other core (in series) during that same round.

Let τ_{e_i} be the hyperedge-transaction executed in round i, where $e_i = (v_1^i, v_2^i, v_3^i) \in E$. By construction, τ_{e_i} conflicts with every transaction except $\tau_{v_1^i}, \tau_{v_2^i}, \tau_{v_3^i}$, so the three vertex-transactions executed during round i must be exactly $\tau_{v_1^i}, \tau_{v_2^i}, \tau_{v_3^i}$.

Let $E' = \{e_1, \ldots, e_k\} \subseteq E$. Since S executes $4k$ distinct transactions, no vertex-transaction is repeated, so the triples $\{v_1^i, v_2^i, v_3^i\}$ are pairwise disjoint. Hence E' is a matching of size k. ◀

▶ **Theorem 17.** *For $p = 2$, unit rewards, and execution times in $\{1, 3\}$, PBC-D is NP-complete.*

Proof. The problem is in NP: a certificate is a schedule S, and in polynomial time we can verify feasibility (conflict-freeness), compute $S.mspan$ and $S.reward$, and compare them to B and K. NP-hardness follows from Theorem 16. Hence, the decision problem is NP-complete. ◀

4.2 $(\frac{2}{3} - \delta)$-Approximation for $p = 2$, $\tau.t \in \{1, 2\}$

In this section, we consider the restricted setting of Problem 2 with $p = 2$, arbitrary rewards, and processing times $\tau.t \in \{1, 2\}$. Let B denote the runtime budget. We present a

deterministic polynomial-time approximation algorithm that, for any $\delta > 0$, returns a feasible schedule with total reward at least $\left(\frac{2}{3} - \delta\right) OPT$. The algorithm is as follows.

1. We define a simplified *non-overlap* variant of Problem 2.
2. We compute a $(1-\epsilon)$-approximation of its optimal solution OPT_{non} in polynomial time (for fixed $\epsilon > 0$).

We prove that the optimum reward of this variant satisfies $OPT_{\text{non}} \geq \frac{2}{3} OPT$ in this restricted setting. Hence, combining the two steps yields an approximation factor of $(1-\epsilon) \cdot \frac{2}{3}$. Equivalently, for any $\delta > 0$, by choosing $\epsilon \leq \frac{3}{2}\delta$ we obtain a $\left(\frac{2}{3} - \delta\right)$-approximation.

4.2.1 The non-overlap variant

The *non-overlap variant* has the same input as Problem 2, but restricts the schedule to consist of independent rounds, where each round executes either: (a) one transaction of length 1 or 2, or (b) two *non-conflicting* transactions in parallel, and the length of such a parallel round is the maximum of the two transaction lengths. In particular, the non-overlap variant does *not* allow scheduling one length-2 transaction in parallel with two length-1 transactions.

Reduction to a budgeted matching. We reduce the non-overlap variant to a maximum-profit matching problem under a time budget. Construct an undirected graph $H = (V_H, E_H)$ as follows.

- For each transaction τ_i, add two vertices $v_i, v_i' \in V_H$ and add the edge $(v_i, v_i') \in E_H$. To each of these edges, assign a profit $c(v_i, v_i') = \tau_i.reward$ and a cost $w(v_i, v_i') = \tau_i.t$.
- For every pair of non-conflicting transactions τ_i, τ_j, add an edge $(v_i, v_j) \in E_H$. To each of these edges, assign a profit $c(v_i, v_j) = \tau_i.reward + \tau_j.reward$ and a cost $w(v_i, v_j) = \max\{\tau_i.t, \tau_j.t\}$

The non-overlap variant is then equivalent to selecting a matching $M \subseteq E_H$ maximizing $\sum_{e \in M} c(e)$ subject to the time budget $\sum_{e \in M} w(e) \leq B$, and then interpreting each matching edge $e \in M$ as one round of length $w(e)$, which yields a feasible non-overlap schedule with the same total reward.

Let OPT_{non} denote the optimal reward for the non-overlap variant.

4.2.2 Computing the Non-Overlap Solution

The above is a *budgeted maximum-profit matching* problem (matching with a knapsack-style budget constraint), which is NP-hard in general. We therefore do not attempt to solve it exactly in polynomial time.

Instead, we use a PTAS for budgeted matching (e.g., the deterministic PTAS of Berger et al. [4]): for any fixed accuracy parameter $\epsilon > 0$, we can compute in polynomial time a matching M_ϵ satisfying $\sum_{e \in M_\epsilon} w(e) \leq B$ and $\sum_{e \in M_\epsilon} c(e) \geq (1-\epsilon) OPT_{\text{non}}$.

4.2.3 $\left(\frac{2}{3} - \delta\right)$-Approximation: Bounding the Loss

Let OPT be the optimal schedule reward for Problem 2 restricted to $p = 2$ and $\tau.t \in \{1, 2\}$. We show $OPT_{\text{non}} \geq \frac{2}{3} OPT$. Together with Step 1, this implies

$$\sum_{e \in M_\epsilon} c(e) \;\geq\; (1-\epsilon) OPT_{\text{non}} \;\geq\; (1-\epsilon) \cdot \frac{2}{3} OPT.$$

Hence, by choosing $\epsilon \leq \frac{3}{2}\delta$, we obtain a $\left(\frac{2}{3} - \delta\right)$-approximation.

▶ **Lemma 18.** *There exists an optimal schedule for Problem 2 (in this restricted setting) such that one of the two cores is* gap-free*, i.e., it executes its scheduled transactions back-to-back starting at time* 0*, with no idle interval that is followed by a later transaction on the same core.* [1]

Proof. Assume for contradiction that every optimal schedule has idle time on *both* cores. Fix an optimal schedule, and let i be the earliest time at which core 1 is idle *and some transaction is scheduled on core 1 at a later time* (i.e., i is an interior gap, not trailing idle). If no such i exists, core 1 is already gap-free and we are done.

If core 2 is also idle at time i, then we can shift every transaction that starts at or after time i on both cores to the left by the same amount, until at least one transaction hits time i. This preserves feasibility and makespan while eliminating the idle time at i on core 1, contradicting the choice of the schedule.

Hence, core 2 is not idle at time i. If a transaction starts on core 2 exactly at time i, then swap the assignments of the two cores for all transactions scheduled at times $\geq i$. This preserves feasibility and makespan and removes the idle time of core 1 at time i.

Otherwise, since $\tau.t \in \{1, 2\}$, core 2 must be executing a length-2 transaction during the interval $[i-1, i]$ (i.e., a transaction that starts at time $i-1$ and ends at time $i+1$). In this case, swap the assignments of the two cores for all transactions scheduled at times $\leq i$. Again, feasibility and makespan are preserved, and core 1 is not idle at time i, a contradiction.

Therefore, there exists an optimal schedule in which at least one core has no idle time. ◀

▶ **Theorem 19.** *The algorithm presented above is a* $\left(\frac{2}{3} - \delta\right)$*-approximation for Problem 2 in the case where* $p = 2$ *and every transaction has length* 1 *or* 2*.*

Proof. Let OPT be the maximum reward achievable in Problem 2 restricted to $p = 2$ and $\tau.t \in \{1, 2\}$, and let OPT_{non} denote the optimal reward of the non-overlap variant from Section 4.2.1. Since Step 1 computes M_ϵ with $\sum_{e \in M_\epsilon} c(e) \geq (1 - \epsilon) OPT_{\text{non}}$, it suffices to show $OPT_{\text{non}} \geq \frac{2}{3} OPT$.

Fix an optimal feasible schedule $S^\star$ of total reward OPT and makespan at most B. By Lemma 18, we may assume without loss of generality that core 1 is gap-free in $S^\star$. All processing times are integral.

Cut points and coarse segments. Call an integer time $t \in \{0, 1, \ldots, B\}$ a *cut point* if, at time t, both cores are at a boundary, i.e., for each core, either the core is idle at time t or a transaction starts/ends at t. Let $0 = t_0 < t_1 < \cdots < t_q = B$ be the cut points, and let $I_k = [t_{k-1}, t_k]$ be the resulting *coarse segments*. Write $S^\star|_{I_k}$ for the restriction of $S^\star$ to segment I_k.

We will construct, independently for each I_k, a schedule $\widehat{S}|_{I_k}$ feasible for the non-overlap variant on I_k such that $\widehat{S}|_{I_k}.\mathit{reward} \geq \frac{2}{3} S^\star|_{I_k}.\mathit{reward}$. Concatenating the segment-wise schedules yields a non-overlap schedule $\widehat{S}$ of makespan at most B and reward at least $\frac{2}{3} OPT$, proving the theorem.

[1] Idle time at the end of the horizon, after that core's last transaction, is permitted and is not a gap; in particular, when the budget cannot be exactly filled, e.g. an odd budget with only length-2 transactions, the unused tail does not violate gap-freeness.

Distinguished gap and local move. Fix a coarse segment $I = [a, b]$. If there exists any idle unit on either core within I, define the *distinguished gap* to be the leftmost idle unit interval $[t, t+1] \subseteq I$ among all idle units in I (breaking ties arbitrarily but deterministically). If I is *tightly packed* (both cores are busy throughout I), we will delete one transaction in I with the minimum reward to create an idle interval and then define the distinguished gap as above.

Our only schedule modification is the following *unit alignment move*: whenever a length-2 transaction τ is adjacent to the distinguished gap on its core (i.e., the gap is exactly one unit immediately before or after τ), we shift τ by one time unit into the distinguished gap (left or right), choosing a feasible direction that makes τ become fully aligned with a non-conflicting length-2 transaction on the other core with which τ was half-overlapping. This move preserves feasibility and does not create new conflicts, and it preserves total reward.

Segment types. We distinguish three cases for the structure of $S^\star|_I$.

Case 1: $S^\star|_I$ is already non-overlap-standard. Then set $\widehat{S}|_I := S^\star|_I$, and $\widehat{S}|_I.reward = S^\star|_I.reward$.

Case 2: I is a 2-vs-$(1+1)$ segment. That is, on one core there is a length-2 transaction τ occupying all of $I = [a, a+2]$, and on the other core there are two consecutive length-1 transactions occupying $[a, a+1]$ and $[a+1, a+2]$. This pattern is feasible in the original problem but forbidden by the non-overlap variant. Define $\widehat{S}|_I$ by deleting the minimum-reward transaction among these three, and keeping the remaining two transactions scheduled as in $S^\star|_I$. Then $\widehat{S}|_I$ is feasible for the non-overlap variant on I, and $\widehat{S}|_I.reward \geq \frac{2}{3} S^\star|_I.reward$.

Case 3: I is neither Case 1 nor Case 2. If I is not tightly packed, the distinguished gap already exists. If I is tightly packed, delete one transaction of minimum reward among all transactions scheduled in I by $S^\star|_I$, thereby creating at least one idle unit; then define the distinguished gap to be the leftmost idle unit in I.

Starting from this distinguished gap, repeatedly apply the unit alignment move to the transaction adjacent to the distinguished gap until no such move is possible. Each move shifts a length-2 transaction by one unit into idle time and thus preserves feasibility and reward. Moreover, each move strictly decreases the number of misaligned non-conflicting length-2 half-overlaps within I (i.e., non-conflicting pairs of length-2 transactions overlapping by exactly one unit), since the moved transaction becomes fully aligned with its designated non-conflicting partner. As this number is finite, the process terminates.

At termination, there are no remaining misaligned non-conflicting length-2 half-overlaps in I. Observe that with processing times in $\{1, 2\}$, any local obstruction to being a concatenation of non-overlap-standard rounds is necessarily either (i) a misaligned non-conflicting 2–2 half-overlap, or (ii) a 2-vs-$(1+1)$ pattern over an interval of length 2.

Since at termination (i) does not occur and Case 2 has already been handled exclusively, it follows that the resulting schedule on I is a concatenation of non-overlap-standard rounds. Let $\widehat{S}|_I$ denote the resulting schedule; it is feasible for the non-overlap variant on I. In Case 3, if I is not tightly packed, then no deletion is performed and therefore $\widehat{S}|_I.reward = S^\star|_I.reward \geq \frac{2}{3} S^\star|_I.reward$.

Assume now that I is tightly packed and that we delete one transaction $\tau_{\min}$ of minimum reward among the $n_I := |T(I)|$ transactions scheduled in I (where $T(I)$ is the set of transactions scheduled by $S^\star$ in I). All subsequent unit alignment moves are reward-

preserving, hence

$$\widehat{S}|_I.reward = S^\star|_I.reward - \tau_{\min}.reward \geq S^\star|_I.reward - \frac{1}{n_I} S^\star|_I.reward$$
$$= \frac{n_I - 1}{n_I} S^\star|_I.reward.$$

Moreover, in this case we have $n_I \geq 3$ (otherwise I would consist of two perfectly aligned length-2 transactions and would already be non-overlap-standard, contradicting that we perform a deletion in Case 3). Therefore $\frac{n_I-1}{n_I} \geq \frac{2}{3}$, and thus $\widehat{S}|_I.reward \geq \frac{2}{3} S^\star|_I.reward$.

Combining segments. Apply the above transformation independently to each coarse segment I_k and concatenate the resulting schedules $\widehat{S}|_{I_k}$. Since the I_k are consecutive and share cut points at their endpoints, the concatenation is a feasible schedule $\widehat{S}$ for the non-overlap variant of makespan at most B. Moreover,

$$\widehat{S}.reward = \sum_{k=1}^{q} \widehat{S}|_{I_k}.reward \geq \frac{2}{3} \sum_{k=1}^{q} S^\star|_{I_k}.reward = \frac{2}{3} S^\star.reward = \frac{2}{3} OPT.$$

Therefore $OPT_{\text{non}} \geq \widehat{S}.reward \geq \frac{2}{3} OPT$, and combining this with step 1 yields

$$\sum_{e \in M_\epsilon} c(e) \;\geq\; (1-\epsilon) OPT_{\text{non}} \;\geq\; (1-\epsilon) \cdot \frac{2}{3} OPT,$$

and choosing $\epsilon \leq \frac{3}{2}\delta$ gives a $(\frac{2}{3} - \delta)$-approximation. ◀

5 Experiments and Results

5.1 Experimental Setup

We empirically evaluate Algorithm 1 in the homogeneous (round-based) setting with constant p (see Section 3.2), and the approximation algorithm of Section 4.2 on heterogeneous instances with $p = 2$ and processing times in $\{1, 2\}$. Since the PTAS of Berger et al. [4] used in the algorithm of Section 4.2 is not straightforward to implement directly, we instead solve the non-overlap variant exactly via an MILP formulation for the experiments in this section.

Hardware and Software. All experiments were run on a Dell PowerEdge R7615 equipped with a single AMD EPYC 9654P (96 cores/ 192 threads) and ≈1.5TiB RAM, running Ubuntu 24.04.3 LTS. MATLAB R2025b is used to implement Algorithm 1 and invoke HiGHS 1.7.1 through MATLAB's `intlinprog` for MILPs. A maximum running time of $MaxTime := 6000$ s. was set for each experiment.

Workloads. We use Ethereum mainnet execution traces from blocks 21,631,019–21,635,079 (15–16 January 2025), from which we extract per-transaction gas used and read/write sets, and derive conflicts from overlapping read/write sets (see Section 2). To obtain homogeneous processing times to evaluate Algorithm 1, we restrict to pure ETH transfers (intrinsic gas 21,000), which do not invoke smart contracts. Across the dataset, the number of conflicts per transaction ranges from 0 to 11,369, with a mean of 1,363 and a median of 75, reflecting a heavily skewed distribution in which a small number of transactions generate the majority of conflicts. For the experiments of the algorithm in Section 4.2, we additionally generate

heterogeneous instances by bundling consecutive transactions in a 1:2 ratio. Hence, we have single transactions (of length 1), and double transactions (with length 2) obtained by merging pairs of two consecutive transactions, inheriting the union of their access lists and the sum of their rewards. Length-1 and length-2 transactions each comprise 50% of the pool, distributed uniformly at random. For this heterogeneous dataset, the number of conflicts per transaction ranges from 0 to 18,895, with a mean of 2159 and a median of 361.

Our workloads instantiate the specific theoretical regimes of this paper rather than the full diversity of Ethereum execution. The homogeneous workload matches the uniform-processing-time assumption; the heterogeneous $\{1, 2\}$ workload is a controlled synthetic construction realizing Section 4.2.

Experiment parameters. For the homogeneous transactions, experiment duration is discretized into equal-length rounds, so progress is naturally parameterized by the number of rounds R, with per-round capacity p. For a configuration (R, p, X), we emulate a mempool by selecting a sliding-window pool of $R \cdot p \cdot X$ consecutive transactions, where X is the pool factor (so the pool contains roughly X times the transactions that fit in R rounds). We evaluate each configuration with rounds $R \in \{10, 30, 100\}$, cores $p \in \{2, 4, 8\}$, and pool factor $X \in \{2, 4, 8\}$. For the heterogeneous experiments of Section 4.2, which concern the $p = 2$ setting by definition, we parameterize by runtime budget $B \in \{210\text{K}, 630\text{K}, 2100\text{K}\}$ gas units (corresponding to $R \in \{10, 30, 100\}$ rounds of pure ETH transfers) and pool factor $X \in \{2, 4, 8\}$, where X controls the size of the mempool: a pool factor of X means the mempool contains roughly X times the number of transactions that fit within the budget B. For both experiment types, we run five non-overlapping workloads and report the mean.

Random baseline. This baseline applies to the homogeneous experiments of Algorithm 1 only. To test whether near-optimality is due to the algorithmic scheduling decisions (rather than the workload being inherently easy), we compare Algorithm 1 not only against the MILP-based optimum [13], but also against a simple *random feasible* baseline. In each round, we generate a random permutation of the remaining transactions and scan it from left to right. Whenever we encounter a feasible transaction (i.e., non-conflicting with the transactions already selected for the current round), we add it to the round and mark all transactions conflicting with it as *ineligible for this round*. We continue scanning the same permutation until selecting p transactions or exhausting the list, then proceed to the next round using the remaining unscheduled transactions.

5.2 Results

We evaluate the greedy $(1-1/e)$-approximation algorithm (Algorithm 1) on the homogeneous (round-based) PBC setting and compare its achieved block reward against the MILP-based optimal baseline from prior work on parallel block construction [13]. We compare against two further baselines: a random feasible schedule, which tests whether near-optimality stems from the scheduling decisions or from the workload being inherently easy; and the fast deterministic heuristics of [13], which represent the kind of lightweight strategy a production builder would realistically run.

Table 2 reports normalized reward as a percentage of the MILP optimum. Across all tested configurations (varying R, p, and the pool factor X), Algorithm 1 consistently produces blocks whose total reward $\sum_\tau \tau.reward$ is very close to the MILP optimum. In practice, our algorithm almost always achieves rewards above 99% of the optimal, which is much better than the guaranteed factor of $1 - 1/e = 0.63$. In contrast, the random baseline performs

	X = 2								
	Random			Algorithm 1			Heuristics from [13]		
	p = 2	p = 4	p = 8	p = 2	p = 4	p = 8	p = 2	p = 4	p = 8
R = 10	73.07	69.72	61.02	99.66	99.08	99.92	99.66	98.46	99.84
R = 30	58.96	49.15	65.73	99.72	99.63	99.91	99.67	99.47	99.86
R=100	60.86	60.85	58.55	99.99	99.94	99.99†	99.99	99.94	99.95
	X = 4								
	Random			Algorithm 1			Heuristics from [13]		
	p = 2	p = 4	p = 8	p = 2	p = 4	p = 8	p = 2	p = 4	p = 8
R = 10	44.95	40.66	40.07	100	99.98	99.54	99.92	99.90	99.49
R = 30	40.14	46.68	42.34	99.56	99.44	99.86	99.47	99.44	99.85
R=100	41.3	48.8	39.73	100	99.99	99.89†	100	99.98	99.88
	X = 8								
	Random			Algorithm 1			Heuristics from [13]		
	p = 2	p = 4	p = 8	p = 2	p = 4	p = 8	p = 2	p = 4	p = 8
R = 10	30.81	25.27	28.33	99.99	99.99	98.76	99.98	99.99	98.75
R = 30	30.05	23.41	21.34	100	99.54	99.7	100	99.5	99.7
R=100	22.9	28.46	15.99	99.78	99.98	100†	99.78	99.97	100

Table 2 Random baseline, Algorithm 1, and the heuristics of [13]: block reward $\sum_\tau \tau.reward$ as a percentage of the MILP optimum from [13]. †Average of only instances that completed; the remainder timed out (See Section 5.2).

substantially worse, and the gap widens as the pool factor X increases: for example, at $X = 8$, random achieves only about 16–31% of optimal across our R, p settings. This indicates that the near-optimal behavior of Algorithm 1 is driven by its conflict-aware selection, not by the workload being trivially easy. The deterministic heuristics of [13] also reach above 99% of the optimum, matching Algorithm 1 in reward in some configurations, performing a bit worse in many cases but never better. The distinction between the two lies not in reward but in guarantees and cost, which we examine next.

We set the same per-instance timeout as prior work (6,000 seconds) [13]. The heuristics run in well under a second across every configuration (at most 0.9 s, at R=100, p=8, X=8). In contrast, each round of Algorithm 1 solves an exact maximum-reward clique of size at most p, which costs $O(n^p)$ in the worst case. For $p = 2$ and $p = 4$ it likewise remains sub-second, but at $p = 8$ the runtime grows sharply: at R=100, p=8 it reaches 557 s at X=4 and 2499 s at X=8, and a small number of the largest instances (1, 1, and 4 instances out of the 5 runs, for X=2, 4, 8 respectively, all at R=100, p=8) exceed the time cap. This is the empirical face of the $O(n^p)$ per-round cost: exact selection becomes expensive exactly where the heuristic remains cheap. That the exact search nonetheless terminates for most $p = 8$ instances reflects the structure of Ethereum conflict graphs, which are dense enough that each transaction is compatible with relatively few others, keeping the effective clique search far below its worst case.

Taken together, these results show that on realistic homogeneous workloads a cheap heuristic is enough to recover almost all the reward, and that the value of Algorithm 1 is not empirical dominance but its provable $(1 - 1/e)$ guarantee: unlike the heuristics, it retains a principled performance floor on adversarial instances, where reward-ordered strategies can be forced to a $1/p$ fraction of the optimum.

	X=2	X=4	X=8
B=210K	99.65	99.78	99.69
B=630K	99.94	100	99.95
B=2100K	99.988	99.99	99.98

Table 3 Non-overlap reward as a percentage of optimal solution, for $p = 2$ and transaction gas amounts 21K and 42K.

Table 3 reports, for $p = 2$ and the heterogeneous workload of Section 4.2, the reward achieved by the non-overlap MILP as a percentage of the optimal MILP solution, averaged

over five non-overlapping workloads per configuration. Please note that the theoretical algorithm of Section 4.2 computes a $(1-\varepsilon)$-approximation of OPT_{non} via a PTAS for budgeted matching. In our experiments, we instead solve the non-overlap variant exactly by formulating it directly as an MILP. The reported ratio OPT_{non}/OPT therefore measures the true gap between the non-overlap optimum and the overall optimum, without any approximation error from the PTAS. Across all tested values of B and pool factor X, the non-overlap variant consistently achieves above 99% of the optimal reward, well above the theoretical guarantee of $2/3-\delta$. The result aligns with the homogeneous setting: the theoretical worst-case bound of $2/3$ is far from tight for Ethereum workloads, and the non-overlap restriction incurs negligible reward loss in practice.

6 Conclusions and Future Work

We studied the *Parallel-Block Construction* problem that arises when a block builder jointly decides which mempool transactions to include and how to schedule them on p identical cores under a hard runtime budget, subject to conflict constraints induced by shared state access. Our results provide a detailed complexity and approximation map that explains when algorithmic optimization is plausible and when it is provably out of reach. Our experiments on both homogeneous and heterogeneous Ethereum workloads confirm that the theoretical worst-case bounds are far from tight in practice, with both algorithms consistently achieving above 99% of optimal. Together, these results reveal sharp phase transitions: small changes in p or in the allowed processing-time set can drastically alter what can be computed efficiently.

Future work. Several directions remain open. First, on the homogeneous side, our results place the tractability boundary between $p=2$ (exactly solvable, Theorem 15) and $p=4$ (NP-complete, Theorem 14), leaving $p=3$ unresolved in the present paper. We have preliminary results indicating that the problem is in fact NP-complete for every fixed $p\geq 3$, via a reduction from Partition into Triangles for $p=3$ [8], lifted to larger p by a padding argument, which would sharpen the boundary to $p\leq 2$ tractable and $p\geq 3$ hard [3]. We leave the full development to future work.

Second, our complexity map for heterogeneous transactions leaves several natural regions open. We resolve only the length sets $\{1,2\}$ (approximable) and $\{1,3\}$ (NP-complete), and even these are not fully settled: the approximability of $\{1,3\}$ remains open, as does the exact complexity of $\{1,2\}$ and whether its $2/3$ bound is tight. The behaviour of arbitrary length sets and of unbounded processing times is likewise open. Closing these cases, together with a better understanding of the parameterized complexity of PBC with respect to p, the number of rounds, or structural properties of the conflict graph, would complete the landscape.

Third, extending to richer workloads, both in conflict structure (e.g., DeFi and ERC-20 transactions) and in model assumptions, would clarify how quickly performance degrades as instances move away from the regimes where we have strong guarantees. Our model assumes transaction execution times and read/write sets are known at scheduling time; while systems such as Solana's Sealevel enforce this via declared access lists [19], Ethereum requires execution or simulation to recover these quantities, and a growing body of work on gas estimation and access-list prediction provides a starting point for such extensions.

Finally, bridging the gap between theory and practice suggests designing provably good but more scalable primitives for key subroutines (notably the per-round clique selection used in the $(1-1/e)$-approximation algorithm), as well as studying online/streaming variants that more closely reflect mempool dynamics during real-time block construction.

References

1 Aptos Dev Docs. Block-stm execution engine in aptos, 2026. URL: https://aptos.dev/network/blockchain/execution.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

3 Brenda S Baker and Edward G Coffman. Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2):225–243, 1996. URL: https://www.sciencedirect.com/science/article/pii/030439759600031X, doi:10.1016/0304-3975(96)00031-X.

4 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, Jun 2011. doi:10.1007/s10107-009-0307-4.

5 Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, Brandon Williams, and Lu Zhang. Sui lutris: A blockchain combining broadcast and consensus. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 2606–2620, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3658644.3670286.

6 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005. arXiv:https://doi.org/10.1137/S0097539700382820, doi:10.1137/S0097539700382820.

7 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

8 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

9 Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, page 232–244, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3572848.3577524.

10 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

11 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

12 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

13 Arivarasan Karmegam, Lucianna Kiffer, and Antonio Fernández Anta. Exploiting Multi-Core Parallelism in Blockchain Validation and Construction. In Martin Aumüller and Irene Finocchi, editors, *24th International Symposium on Experimental Algorithms (SEA 2026)*, volume 371 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:21, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SEA.2026.23, doi:10.4230/LIPIcs.SEA.2026.23.

14 Daniel Kowalczyk and Roel Leus. An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling*, 20(4):355–372, 2017. doi:10.1007/s10951-016-0482-0.

15 Dániel Marx. Graph colouring problems and their applications in scheduling. *Periodica Polytechnica Electrical Engineering*, 48(1-2):11–16, 2004. URL: https://pp5.omikk.bme.hu/ee/article/view/926.

16 Ulrich Pferschy and Joachim Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249, 2009.

17 Ankit Ravish, Yaron Hay, Piduguralla Manaswini, Rishabh Jain, Roy Friedman, and Sathya Peri. Efficient scheduling of smart contract transactions via conflict graph coloring. In *2025*

IEEE 30th Pacific Rim International Symposium on Dependable Computing (PRDC), pages 91–101, 2025. doi:10.1109/PRDC67299.2025.00019.

18 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.

19 Solana. Sealevel — parallel processing thousands of smart contracts. https://solana.com/news/sealevel---parallel-processing-thousands-of-smart-contracts, 2026. Accessed: 2026-01-19.

20 J. D. Ullman. Polynomial complete scheduling problems. In *Proceedings of the Fourth ACM Symposium on Operating System Principles*, SOSP '73, page 96–101, New York, NY, USA, 1973. Association for Computing Machinery. doi:10.1145/800009.808055.

21 Lee James White. *A parametric study of matchings and coverings in weighted graphs*. PhD thesis, University of Michigan, 1967.

22 Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain. White paper, 2017. Version 0.8.13. URL: https://solana.com/solana-whitepaper.pdf.