Dynamic Pricing for Non-fungible Resources

Theo Diamandis
tdiamand@mit.edu

Alex Evans
aevans@baincapital.com

Tarun Chitra
tarun@gauntlet.network

Guillermo Angeris
gangeris@baincapital.com

August 2022

Abstract

Public blockchains implement a fee mechanism to allocate scarce computational resources across competing transactions. Most existing fee market designs utilize a joint, fungible unit of account (*e.g.*, *gas* in Ethereum) to price otherwise non-fungible resources such as bandwidth, computation, and storage, by hardcoding their relative prices. Fixing the relative price of each resource in this way inhibits granular price discovery, limiting scalability and opening up the possibility of denial-of-service attacks. As a result, many prominent networks such as Ethereum and Solana have proposed multi-dimensional fee markets. In this paper, we provide a principled way to design fee markets that efficiently price multiple non-fungible resources. Starting from a loss function specified by the network designer, we show how to compute dynamic prices that align the network's incentives (to minimize the loss) with those of the users and miners (to maximize their welfare), even as demand for these resources changes. Our pricing mechanism follows from a natural decomposition of the network designer's problem into two parts that are related to each other via the resource prices. These results can be used to efficiently set fees in order to improve network performance.

1 Introduction

Public blockchains allow any user to submit a transaction that modifies the shared state of the network. Transactions are independently verified and executed by a decentralized network of *full nodes*. Because full nodes have finite resources, blockchains limit the total computational resources that can be consumed per unit of time. As user demand may fluctuate, most blockchains implement a *transaction fee* mechanism in order to allocate finite computational capacity among competing transactions.

Smart contracts and gas. Many blockchains enable transactions that execute Turing-complete programs called *smart contracts*. Users can submit transactions to the network

that may trigger the execution of smart contracts. Once such a transaction is included in a block, full nodes must re-execute the transaction in order to obtain the resulting updated state of the ledger. All of these transactions consume computational resources, whose total supply is finite. To prevent transactions with excessive resource use and transaction spam, some smart-contract blockchains require users to pay fees in order to compensate the network for processing their transactions.

Most smart contract platforms calculate transaction fees based on a shared unit of account. In the Ethereum Virtual Machine (EVM), this unit is called *gas*. Each operation in the EVM requires a hardcoded amount of gas which is intended to reflect its relative resource usage. The network enforces a limit on the total amount of gas consumed across all transactions in a given block. This limit, called the *block limit*, prevents the chain from expending computational resources too quickly for full nodes to catch up to the latest state in a reasonable amount of time. Block limits must take into account the maximum amount of each resource that each block may consume (such as storage, bandwidth, or memory) without posing an extreme burden on full nodes meeting the minimal hardware specifications. Because the block limit fixes the total gas supply in each block, the price of gas in 'real' terms (*e.g.*, in terms of US Dollars) fluctuates based on demand for transactions in the block.

One-dimensional transaction fees. Calculating transaction fees through a single, joint unit of account, such as gas, introduces two major challenges. First, if the hardcoded costs of each operation are not precisely reflective of their relative resource usage, there is a possibility of denial-of-service attacks (specifically, *resource exhaustion attacks* [PL19]), where an attacker exploits resource mispricing to overload the network. Historically, the Ethereum network has suffered from multiple DoS attacks [But16b; But16c; Wil16] and has had to manually adjust the relative prices accordingly (*e.g.*, [But16a; BS20a]). Amending the hardcoded costs of each gas operation in response to such attacks typically requires a hard fork of the client software.

Second, one-dimensional fee markets limit the theoretical throughput of the network. Using a joint unit of account to price separate resources decouples their price from supply and demand. As an extreme example to illustrate this dynamic, if the block gas limit is fully saturated with exclusively CPU-intensive operations, gas price will increase as transactions compete for limited space. The cost of transactions that consume exclusively network bandwidth (and nearly no CPU resources) will also increase because these also require gas, even if demand and supply for bandwidth resources across the network remain unchanged. As a result, fewer bandwidth-intensive transactions can be included in the block despite spare capacity, limiting throughput. This limitation occurs because the shared unit of account only allows the network to price resources *relative to each other* and not in real terms based on the supply and demand for each resource. As we will discuss, allowing resources to be priced separately promotes more efficient resource utilization by enabling more precise price discovery. We note that this increase in throughput need not increase hardware requirements for full nodes.

Multidimensional fee markets. Due to the potential scalability benefits of more granular price discovery, a number of popular smart contract platforms are actively exploring multidimensional fee market mechanisms [Adl22; But22a]. We discuss some example proposals that are under active development, below.

Rollups and data markets. Rollups are a popular scaling technology that effectively decouples transaction validation and execution from data and consensus [But21]. In rollups, raw transaction data is posted to a base blockchain. Rollups also periodically post succinct proofs of valid execution to the base chain in order to enable secure bridging, prevent rollbacks, and arbitrate potential disputes by using the base chain as an anchor of trust. Rollups naturally create two separate fee markets, one for base layer transactions and one for rollup execution. As rollups have become a popular design pattern for achieving scalability, specialized blockchains (called *lazy blockchains*) that exclusively order raw data through consensus (*i.e.*, do not perform execution) have emerged [ABSB18; AB19; Pol21; NNT21; Tas+22]. These blockchains naturally allow for transaction data/bandwidth and execution to be priced through independent (usually one-dimensional) fee markets [Adl21]. Similarly, Ethereum developers have proposed EIP-2242, wherein users may submit special transactions which contain an additional piece of data called a *blob* [Adl19b; Adl19a]. Blobs may contain arbitrary data intended to be interpreted and executed by rollups rather than the base chain. Later, EIP-4844 extended these ideas by establishing a two-dimensional fee market wherein data blobs and base-chain gas have different limits and are priced separately [But22c]. EIP-4844 therefore intends to increase scalability for rollups, as blobs do not have to compete with base-chain execution for gas.

Incentivizing parallelization. Most smart contract platforms, including the EVM, execute program operations sequentially by default, limiting performance. There are several proposals to enable parallel execution in the EVM which generally fall into two categories. The first involves minimal changes to the EVM and pushes the responsibility of identifying opportunities for parallel execution to full nodes [Gel+22; Che+21; SH19]. The other approach involves *access lists* which require users to specify which accounts their transaction will interact with, allowing the network to easily identify non-conflicting transactions that can be executed in parallel [But17; BS20b]. While Ethereum makes access lists optional, other virtual machine implementations, such as Solana Sealevel and FuelVM, require users to specify the accounts their transaction will interact with [Yak20; Lab; Adl20]. Despite this capability, a large fraction of transactions often want to access the same accounts in scenarios including auctions, arbitrage opportunities, and new product launches. Such contention significantly limits the potential benefits of the virtual machine's parallelization capabilities. As a result, developers of Solana have proposed multi-dimensional fee markets that price interactions with each account separately in order to charge higher fees for transactions which require sequential execution [Yak21]. Such a proposal incentivizes usage of spare capacity on multi-core machines.

This paper. In this paper, we formally illustrate how to efficiently update resource prices, what optimization problem these updates attempt to solve, and some consequences of these observations. We also numerically demonstrate that this approach enhances network performance and reduces DoS-style resource congestion attacks. We frame the pricing problem in terms of an idealized, omniscient network designer who chooses transactions to include in blocks in order to maximize total welfare, subject to demand constraints. (The designer is omniscient as the welfare is unknown and likely unmeasurable in any practical setting.) We show that this problem, which is the 'ideal end state' of a blockchain but not immediately useful in itself, decomposes into two problems, coupled by the resource prices. One of these two problems is a simple one which can be easily solved on chain and represents the cost to the network for providing certain resources, while the other is a maximal-utility problem that miners and users implicitly solve when creating and including transactions for a given block. Correctly setting the resource prices aligns incentives such that the resource costs to the network are exactly balanced by the utility gained by the users and miners. This, in turn, leads to block allocations which solve the original 'ideal' problem, on average.

For convenience, we provide appendix A as a short introduction to convex optimization. We recommend readers unfamiliar with convex optimization at least skim this appendix, as it provides a short introduction to all the mathematical definitions and major theorems used in this paper. As a general guideline, we recommend those uninterested in theoretical results to skip §3.2 and §3.3 on a first reading.

1.1 Related work

The resource allocation problem has been studied in many fields, including operations research and computer systems. Agrawal, et al. [Agr+22] proposed a similar formulation and price update scheme for fungible resources where utility is defined per-transaction. Prior work on blockchain transaction fees varies from the formal axiomatic analysis of game theoretic properties that different fee markets should have [Rou21; CS21] to analysis of dynamic fees from a direct algorithmic perspective [Fer+21; Leo+21; Rei+21]. Works of the latter form generally focus on whether the system macroscopically converges to an equilibrium. Moreover, these mechanisms focus on dynamic pricing at the block level (*e.g.*, how many transactions should be allowed in a block?) versus questions of how capacity should be allocated and priced across different transaction types.

EIP-1559. EIP-1559 [But+19], implemented last year, proposed major changes to Ethereum's transaction fee mechanism. Specifically, EIP-1559 implemented a base fee for transactions to be included in each block, which is dynamically adjusted to hit a target block usage and burned instead of rewarded to the miners. We note that while EIP-1559 is closely related to the problem we consider, it ultimately has a different goal: EIP-1559 attempts to make the fee estimation problem easier in a way that disincentivizes manipulation and collusion [Rou21; But18]. We, on the other hand, aim to price resources dynamically to achieve a given network-specified objective. Finally, we note that prior work such as [Fer+21] has

proved incentive compatibility for a large set of mechanisms that are a superset of EIP-1559. It is likely (but not proven in this work) that our model fits within their incentive compatibility framework. We leave game theoretic analysis and strategies to ensure incentive compatibility as future work.

2 Transactions and resources

Before introducing the network's resource pricing problem, we discuss the general set up and motivation for the problem in the case of blockchains.

Transactions. We will start by reasoning about *transactions*. A transaction can represent arbitrary data sent over the peer-to-peer network in order to be appended to the chain. Typically, a transaction will represent a value transfer or a call to a smart contract that exists on chain. These transactions are broadcasted by users through the peer-to-peer network and collected by consensus nodes in the *mempool*, which contains all submitted transactions that have not been included on chain. A *miner* gets to choose which transactions from the mempool are included on chain. Miners may also outsource this process to a *block builder* in exchange for a reward [But]. Once a transaction is included on chain, it is removed from the mempool. (Any conflicting transactions are also removed from the mempool.)

Nodes. Every transaction needs to be executed by *full nodes* (which we will refer to as 'nodes'). Nodes compute the current state of the chain by executing and checking the validity of all transactions that have been included on the chain since the genesis block. Many blockchains have minimum computational requirements for nodes in a blockchain: any node meeting these requirements should be able to eventually 'catch up' to the head of the chain in a reasonable amount of time, *i.e.*, execute all transactions and reach the latest state, even as the chain continues to grow. (For example, Ethereum requires 4GB RAM and 2TB of SSD Storage, and they recommend at least a Intel NUC, 7th gen processor [Eth].) These requirements both limit the computational resources each block is allowed to consume and promote decentralization by ensuring the required hardware does not become prohibitively expensive. If transactions are included in a blockchain faster than nodes are able to execute them, nodes cannot reconstruct the latest state and can't ascertain the validity of the chain. This type of denial of service (DoS) attack is also referred to as a resource exhaustion attack. (As a side note, in some systems, it is possible to provide an easily-verifiable proof that the state is correct without the need to execute all past transactions to validate the state of the chain. In these systems, the time-consuming step is generating the proofs. A similar market mechanism might make sense for this case, but we do not explore this topic here.)

Resource targets and limits. There are several ways to prevent this type of denial of service attack. For example, one method is to enforce that any valid transaction (or sequence of transactions, *e.g.*, a block) consumes at most some fixed upper bound of resources, or combinations of resources. These limits are set so that that a node satisfying the minimum

node requirements is always able to process transactions quickly enough to catch up to the head of the chain in a reasonable amount of time. Another possibility is to disincentivize miners and users from repeatedly including transactions that consume large amounts of resources while allowing short 'bursts' of resource-heavy transactions. This margin needs to be carefully balanced so that a node meeting the minimum requirements is able to catch up after a certain period of time. This intuition suggests having both a 'resource target' and a larger 'resource limit,' which we will formalize in what follows.

Resources. Most blockchain implementations define a number of *meterable resources* (simply *resources* from here on out) which we will label $i = 1, \ldots, m$, that a transaction can consume. For example, in Ethereum, the resources could be the individual Ethereum Virtual Machine (EVM) opcodes used in the execution of a transaction. In this paper, the notion of a 'resource' is much more general than simply an 'opcode' or an 'execution unit'. Here, a resource can refer to anything as coarse as 'total bandwidth usage' to as granular as individual instructions executed on a specific core of a machine—the only requirement for a resource, as used in this paper, is that it can be easily and consistently metered across any node. For a given transaction $j = 1, \ldots, n$, we will let $a_j \in \mathbb{R}^m_+$ be the vector of resources that transaction j consumes. In particular, the ith entry of this vector, $(a_j)_i$, denotes the amount of resource i that transaction j uses. We note that the resource usage $(a_j)_i$ does not, in fact, need to be nonnegative. While our mechanism works in the more general case (with some small modifications), we assume nonnegativity in this work for simplicity.

Combined resources. This framework naturally includes combinations of resources as well. For example, we may have two resources R_1 and R_2, each cheap to execute once in a transaction, which are costly to execute serially (*i.e.*, it is costly to execute R_1 and then R_2 in the same transaction). In this case, we can create a 'combined' resource R_1R_2 which is itself metered separately. Note that, in our discussion of resources, there is no requirement that the resources be independent in any sense and such 'combined resources' are themselves very reasonable to consider.

Resource utilization targets. As mentioned previously, many networks have a minimum node requirement, implying a sustained target for resource utilization in each group of transactions added to the blockchain. (For simplicity, we will call this sequence of transactions a *block*, though it could be any collection of transactions that makes sense for a given blockchain.) We will write this resource utilization target as a vector $b^\star \in \mathbb{R}^m$ whose ith entry denotes the desired amount of resource i to be consumed in a block. The resource utilization of a particular block is a linear function of the transactions included in a block, written as a Boolean vector $x \in \{0,1\}^n$, whose jth entry is one if transaction j is included in the block and is zero otherwise. We will write $A \in \mathbb{R}^{m \times n}$ as a matrix whose jth column is the vector of resources a_j consumed by transaction j. We can then write the total quantity of consumed resources by this block as

$$y = Ax, \tag{1}$$

where $y \in \mathbb{R}^m$ is a vector whose ith entry denotes the quantity of resource i consumed by all transactions included in the block. Additionally, we can write the deviation from the target utilization, sometimes called the residual, as

$$Ax - b^\star,$$

i.e., a vector whose i^{th} element is the total quantity of resource i, consumed by transactions in this block, minus the target for resource i, $b_i^\star$. For example, in Ethereum post EIP-1559, there is only one resource, gas, which has a target of 15M. (We will see later how this notion of a 'resource utilization target' can be generalized to a loss function.)

Resource utilization limits. In addition to resource targets, a blockchain may introduce a resource limit b such that any valid block with transactions x must satisfy

$$Ax \leq b.$$

Continuing the example from before, Ethereum after EIP-1559 has a single resource, gas, with a resource limit of 30M.

Network fees. We want to incentivize users and miners to keep the total resource usage 'close' to $b^\star$. To this end, we introduce a *network fee* p_i for resource $i = 1, \ldots, m$, which we will sometimes call the *resource price*, or just the *price*. If transaction j with resource vector a_j is included in a block, a fee $p^T a_j = \sum_i p_i (a_j)_i$ is paid to the network. (In Ethereum, the network fee is implemented by burning some amount of the gas fee for each transaction and can be thought of as the *burn rate*.) For now, we will assume that as the fee p_i gets larger, the amount of resource i used in a block $(Ax)_i$ will become smaller and vice versa.

Resource mispricing. Given A and b, it is, in general, not clear how to set the fees p in order to ensure the network has good performance. (In other words, so that the resource utilization is 'close' to $b^\star$.) As a real world example, starting in Ethereum block number 2283416 (Sept. 18, 2016) an attacker exploited the fact that resources were mispriced, causing the network to slow down meaningfully. This mispricing was fixed via the hard fork on block 2463000 (Oct. 18, 2016) with details outlined in EIP-150 [But16a]. (The effects of this mispricing can still be observed when attempting to synchronize an full node. A dramatic slowdown in downloading and processing blocks happens starting at this block height.) Though usually less drastic, there have been similar events on other blockchains, underscoring the importance of correctly setting resource prices.

Setting fees. We want a simple update rule for the network fees p with the following properties:

1. If $Ax = b^\star$, then there is no update.
2. If $(Ax)_i > b_i^\star$, then p_i increases.

3. Otherwise, if $(Ax)_i < b^\star_i$, then p_i decreases.

There are many update rules with these properties. As a simple example, we can update the network fees as

$$p^{k+1} = \left(p^k + \eta(Ax - b^\star)\right)_+, \tag{2}$$

where $\eta > 0$ some (usually small) positive number, often referred to as the 'step size' or 'learning rate', k is the block number such that p^k are the resource prices at block k, and $(w)_+ = \max\{0, w\}$ for scalar w and is applied elementwise for vectors. Recently, Ethereum developers [But22b] proposed the update rule

$$p^{k+1} = p^k \odot \exp\left(\eta(Ax - b^\star)\right). \tag{3}$$

Here, $\exp(\cdot)$ is understood to apply elementwise, while $\odot$ is the elementwise or Hadamard product between two vectors. The remainder of this paper will show that many update rules of this form are attempting to (approximately) solve an instance of a particular optimization problem with a natural interpretation, where parts of the update rule come from a specific choice of objective function by the network designer.

We note that [Fer+21] has studied fixed points of such iterations in the one-dimensional case, extending the analysis of EIP-1559. However, the multidimensional scenario can be quite a bit more subtle to analyze. For instance, the multiplicative update rule (3) can admit 'vanishing gradient' behavior in high-dimensions [Hoc98]. We suspect that the one-dimensional fee model of [Fer+21] can be extended to the multidimensional rules (2) and (3) and leave this for future work.

3 Resource allocation problem

As system designers, our ultimate goal is to maximize utility of the underlying blockchain network by appropriately allocating resources to transactions. However, we cannot perform this allocation directly, since we cannot control what users nor miners wish to include in blocks, nor do we know what the utility of a transaction is to users and miners. Instead, we aim to set the fees p to ensure that the resource usage is approximately equal to the desired target, which we will represent as an objective function. We will show later that the update mechanisms proposed above naturally fall out of a more general optimization formulation. Similarly to the Transmission Control Protocol (TCP), each update rule is a result of a particular objective function, chosen by the network designer [LL99; Low03].

Loss function. We define a *loss function* of the resource usage, $\ell : \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}$, which maps a block's resource utilization, y, to the 'unhappiness' of the network designer, $\ell(y)$. We assume only that ℓ is convex and lower semicontinuous. (We will not require monotonicity, nonnegativity, or other assumptions on ℓ, but we will show that useful properties do hold in these scenarios.)

For example, the loss function can encode 'infinite dissatisfaction' if the resource target is violated at all:

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ \infty & \text{otherwise.} \end{cases} \tag{4}$$

(Functions of this form, which are either 0 or ∞ at every point, are known as *indicator functions*.) Note also that this loss is not differentiable anywhere, but it is convex. Another possible loss, which is also an indicator function, is

$$\ell(y) = \begin{cases} 0 & y \leq b^\star \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

This loss can roughly be interpreted as: we do not mind any usage below $b^\star$, but we are infinitely unhappy with any usage above the target amounts. Alternatively, we may only care about large deviations from the target $b^\star$:

$$\ell(y) = (1/2)\|y - b^\star\|_2^2,$$

or, perhaps, require that the loss is simply linear and independent of $b^\star$,

$$\ell(y) = u^T y, \tag{6}$$

for some fixed vector $u \in \mathbb{R}^m$. Another important family of losses are those which are separable and depend only on the individual resource utilizations,

$$\ell(y) = \sum_{i=1}^{m} \phi_i(y_i) \tag{7}$$

where $\phi_i : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$ for $i = 1, \ldots, m$, are convex, nondecreasing functions. (The loss (5) is a special case of this loss, while (6) is a special case when the vector u is nonnegative.) We will make the technical assumption that $\phi_i(0) < \infty$ for every i, otherwise no resource allocation would have finite loss.

We will see that each definition of a loss function implies a particular update rule for the network fees p. This utility function can more generally be engineered to appropriately capture tradeoffs in increasing throughput of a particular resource at the possible detriment of other resources.

Resource constraints. Now that we have defined the network designer's loss, which is a way of quantifying 'unhappiness' when the resource usage is y, we need some way to define the transactions that users are willing to create and, conversely, that miners are willing (and able) to include. We do this in a very general way by letting $S \subseteq \{0,1\}^n$ be the set of possible transactions that users and miners are willing and able to create and include. Note that this set is discrete and can be very complicated or potentially hard to maximize over (as is the case in practice). For example, the set S could encode a demand for transactions

which depend on other transactions being included in the block (as is the case in, *e.g.*, miner extractable value [KDC22; Dai+20; QZG21]), network-defined hard constraints of certain resources (such as $Ax \leq b$ for every $x \in S$), and even very complicated interactions among different transactions (if certain contracts can, for example, only be called a fixed number of times, as in NFT mints). We make no assumptions about the structure of this set S, but only require that the included transactions, $x \in \{0,1\}^n$, obey the constraint $x \in S$.

Convex hull of resource constraints. A network designer may be more interested in the long-term resource utilization of the network than the resource utilization of any one particular block. In this case, the designer may choose to 'average out' each transaction over a number of blocks instead of deciding whether or not to include it in the next block. To that end, we, as designers, will be allowed to choose convex combinations of elements of the constraint set S, which we will write as $\mathbf{conv}(S)$. (In general, this means that we can pick probability distributions over the elements of S, and x is allowed to be the expectation of any such probability distribution; *i.e.*, we only require that, for the designer, x is reasonable 'in expectation'.) Here, components of x may vary continuously between 0 and 1; these values have a simple interpretation. If x_i is not 0 or 1, then we can interpret the quantity $1/x_i$ as only including transaction i after roughly $1/x_i$ blocks. Of course, neither users nor miners can choose transactions to be 'partially included', so this property will only apply to the idealized problem we present below. While this relaxation might seem unrealistically 'loose', we will see later how this easily translates to the realistic case where transactions are either included or not (that is, x_i is either 0 or 1) by users and miners.

Transaction utility. Finally, we introduce the *transaction utilities*, which we will write as $q \in \mathbb{R}^n$. The transaction utility q_j for transaction $j = 1, \ldots, n$ denotes the users' and miners' joint utility of including transaction j in a given block. Note that it is very rare (if at all possible) to know the values of q. However, we will see that, under mild assumptions, we do not need to know the values of q in order to get reasonable prices, and reasonable update rules will not depend on q.

Resource allocation problem. We are now ready to write the *resource allocation problem*, which is to maximize the utility of the included transactions, minus the loss, over the convex hull of possible transactions:

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(y) \\ \text{subject to} & y = Ax \\ & x \in \mathbf{conv}(S). \end{array} \tag{8}$$

This problem has variables $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, and the problem data are the resource matrix $A \in \mathbb{R}^{m \times n}$, the set of possible transactions $S \subseteq \{0,1\}^n$, and the transaction utilities $q \in \mathbb{R}^n$. Because the objective function is concave and the constraints are all convex, this is a convex optimization problem (see appendix A). On the other hand, even though the set

$\mathbf{conv}(S)$ is convex, it is possible that $\mathbf{conv}(S)$ does not admit an efficient representation (for example, it may contain exponentially many constraints) which means that solving this problem is, in general, not easy.

Interpretation. We can interpret the resource allocation problem (8) as the 'best case scenario', where the designer is able to choose which transactions are included (or 'partially included') in a block in order to maximize the total utility. While this problem is not terribly useful by itself, since (a) it cannot really be implemented in practice, (b) we often don't know q, and (c) we cannot 'partially include' a transaction within a block, we will see that it will decompose naturally into two problems. One of these problems can be easily solved on chain, while the other is solved implicitly by the users (who send transactions to be included) and miners (who choose which transactions to include). The solutions to the latter problem can always be assumed to be integral; *i.e.*, no transactions are 'partially included'. This will allow us to construct a simple update rule for the prices, which does not depend on q. For the remainder of the paper, we will call this combination of users and miners the *transaction producers*.

Offchain agreements and producers. Due to the inevitability of user-miner collusion, we consider the combination of the two, the transaction producers, as the natural unit. For example, it is not easily possible to create a transaction mechanism where the users are forced to pay miners some fixed amount, since it is always possible for miners to refund users via some off-chain agreement [Rou21]. Similarly, we cannot force miners to accept certain transactions by users, since a miner always has plausible deniability of not having seen a given transaction in the mempool. While not perfect for a general analysis of incentives, this conglomerate captures the dynamics between the network's incentives and those of the miners and users better than assuming each is purely selfishly maximizing their own utility (as opposed to strategically colluding) and suffices for our purposes.

3.1 Setting prices using duality

In this section, we will show a decomposition method for this problem. This decomposition method suggests an algorithm (presented later) for iteratively updating fees in order to maximize the transaction producers' utility minus the loss of the network, given historical observations.

To start, we will reformulate (8) slightly by pulling the constraint $x \in \mathbf{conv}(S)$ into the objective,

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(y) - I(x) \\ \text{subject to} & y = Ax \end{array} \tag{9}$$

where $I : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is the indicator function defined as

$$I(x) = \begin{cases} 0 & x \in \mathbf{conv}(S) \\ +\infty & \text{otherwise.} \end{cases}$$

Dual function. The Lagrangian [BV04, §5.1.1] for problem (9) is then

$$L(x, y, p) = q^T x - \ell(y) - I(x) + p^T(y - Ax),$$

with dual variable $p \in \mathbb{R}^m$. This corresponds to 'relaxing' the constraint $y = Ax$ to a penalty $p^T(y - Ax)$ assigned to the objective, where the price per unit violation of constraint i is p_i. (Negative values denote refunds.) Rearranging slightly, we can write

$$L(x, y, p) = p^T y - \ell(y) + (q - A^T p)^T x - I(x).$$

The corresponding dual function [BV04, §5.1.2], which we will write as $g : \mathbb{R}^m \to \mathbb{R} \cup \{-\infty\}$, is found by maximizing over x and y:

$$g(p) = \sup_y \left(p^T y - \ell(y)\right) + \sup_x \left((q - A^T p)^T x - I(x)\right). \tag{10}$$

Discussion. The first term can be recognized as the Fenchel conjugate of ℓ [BV04, §3.3] evaluated at p, which we will write as $\ell^*(p)$, while the second term is the optimal value of the following problem:

$$\begin{array}{ll} \text{maximize} & (q - A^T p)^T x \\ \text{subject to} & x \in \mathbf{conv}(S), \end{array} \tag{11}$$

with variable $x \in \mathbb{R}^n$. We can interpret this problem as the transaction producers' problem of creating and choosing transactions to be included in a block in order to maximize their utility, after netting the fees paid to the network. We note that the optimal value of (11) in terms of p, which we will write as $f(p)$, is the pointwise supremum of a family of linear (and therefore convex) functions of p, so it, too, is a convex function [BV04, §3.2.3]. Note that since the objective is linear, problem (11) has the same optimal objective value as the nonconvex problem

$$\begin{array}{ll} \text{maximize} & (q - A^T p)^T x \\ \text{subject to} & x \in S, \end{array}$$

where we have replaced $\mathbf{conv}(S)$ with S. (See appendix A.2 for a simple proof.) Finally, since $f(p)$ is the optimal value of problem (11) for fees p, the dual function g can be written as

$$g(p) = \underbrace{\ell^*(p)}_{\text{network}} + \underbrace{f(p)}_{\text{tx producers}}.$$

Since the dual function g is the sum of convex functions ℓ^* and f, it, too, is convex. (We will make use of this property soon.) Having defined the dual function g, we will see how this function can give us a criterion for how to best set the network fees p.

Duality. An important consequence of the definition of the dual function g is *weak duality* [BV04, §5.2.2]. Specifically, letting $s^\star$ be the optimal objective value for problem (8), we have that

$$g(p) \geq s^\star,$$

for every possible choice of price $p \in \mathbb{R}^m$. This is true because we have essentially 'relaxed' the constraint to a penalty, so any feasible point x, y for the original problem (9) always has 0 penalty. (There may, of course, be other points that are not feasible for (9) but are perfectly reasonable for this 'relaxed' version, so we've only made the set of possibilities larger.) The proof is a single line:

$$g(p) = \sup_{x,y} L(x,y,p) \geq \sup_{y=Ax} L(x,y,p) = \sup_{y=Ax} \left(q^T x - \ell(y) - I(x)\right) = s^\star.$$

A deep and important result in convex optimization is that, in fact, there exists a $p^\star$ for which

$$g(p^\star) = s^\star,$$

under some basic constraint qualifications.[1] In other words, adding the constraint $y = Ax$ to the problem is identical to correctly setting the prices p. Since we know for any p that $g(p) \geq s^\star$ then

$$g(p^\star) = \inf_p g(p),$$

or, that $p^\star$ is a minimizer of g. This motivates an optimization problem for finding the prices.

The dual problem. The *dual problem* is to minimize g, as a function of the fees p. In other words, the dual problem is to find the optimal value of

$$\text{minimize} \quad g(p), \tag{12}$$

with variable $p \in \mathbb{R}^m$. If we can easily evaluate g, then, since this problem is a convex optimization problem, as g is convex, solving it is usually also easy. An optimizer of the dual problem has a simple interpretation using its optimality conditions. Let $p^\star$ be a solution to the dual problem (12) for what follows. If the packing problem (11) has a unique solution $x^\star$ for $p^\star$, then the objective value f is differentiable at $p^\star$. (See appendix A.2.) Similarly, under mild conditions on the loss function ℓ (such as strict convexity) the function ℓ^* is differentiable at $p^\star$, with derivative $y^\star$ satisfying $\nabla \ell(y^\star) = p^\star$. In this case, the optimality conditions for problem (12) are that

$$\nabla g(p^\star) = y^\star - Ax^\star = 0. \tag{13}$$

In other words, the fees $p^\star$ that minimize (12) are those that charge the transaction producers the exact marginal costs faced by the network, $\nabla \ell(Ax^\star) = p^\star$. Furthermore, these are exactly the fees which incentivize transaction producers to include transactions that maximize the welfare generated minus the network loss, subject to resource constraints, since $y^\star$ and $x^\star$ are feasible and optimal for problem (8).

[1] The condition is that the relative interior of $A\,\mathbf{conv}(S) \cap \mathbf{dom}\,\ell$ is nonempty. Here, we write $A\,\mathbf{conv}(S) = \{Ax \mid x \in \mathbf{conv}(S)\}$ and $\mathbf{dom}\,\ell = \{x \mid \ell(x) < \infty\}$, while the relative interior is taken with respect to the affine hull of the set. This condition almost always holds in practice for reasonable functions ℓ and sets S.

Differentiability. In general, g is not always differentiable, but is almost universally subdifferentiable, under mild additional conditions on ℓ (*e.g.*, ℓ does not contain a line). Condition (13) may then be replaced with

$$0 \in \partial g(p^\star) = -Y^\star(p^\star) + AX^\star(p^\star),$$

where

$$Y^\star(p) = \operatorname*{argmax}_y \left(p^T y - \ell(y)\right),$$

while $X^\star(p) \subseteq \mathbf{conv}(S)$ are the maximizers of problem (11) for price p. We define $AX^\star(p) = \{Ax \mid x \in X^\star(p)\}$, and write $\partial g(p^\star)$ for the subgradients of g at $p^\star$ (*cf.*, appendix A.5). The condition then says that the intersection of the extremizing sets $Y^\star(p^\star)$ and $AX^\star(p^\star)$ is nonempty at the optimal prices $p^\star$. We show a special case of this below, when $p = 0$, with a direct proof using strong duality that does not require these additional conditions.

3.2 Minimal demand conditions

We can give a condition for which we can guarantee that the optimal prices, *i.e.*, those which minimize the dual problem (12), satisfy $p \neq 0$. The condition is: when resources have zero fee, the optimal set of included transactions that would be included at no price, defined as $X^\star \subseteq [0,1]^n$, with

$$X^\star = \operatorname*{argmax}_{x \in \mathbf{conv}(S)} q^T x,$$

is 'disjoint' from the set of minimizers of the loss, $Y^\star \subseteq \mathbb{R}^m_+$, defined

$$Y^\star = \operatorname*{argmin}_y \ell(y),$$

in the following sense:

$$AX^\star \cap Y^\star = \emptyset,$$

where $AX^\star = \{Ax \mid x \in X^\star\}$. An intuitive version of this condition is that the demand for transactions, if they could be executed at no cost to the transaction producers, always incurs some loss for the network. This, in turn, implies that the optimal fees p for such resources must be nonzero.

For convenience, for the rest of this section, we will define $f^*(x) = q^T x$ when $x \in \mathbf{conv}(S)$ and $-\infty$ otherwise. This lets us write:

$$X^\star = \operatorname{argmax} f^*,$$

and, for any $x^\star \in X^\star$, we have

$$\sup_{x \in \mathbf{conv}(S)} q^T x = q^T x^\star = f^*(x^\star) = \sup f^*.$$

Proof. To see this, we will use strong duality. We will prove the contrapositive statement: if $g(0)$ is optimal, then there exists a point in the intersection $AX^\star \cap Y^\star$.

If $g(0)$ is optimal, then, using strong duality:

$$g(0) = \sup_{\substack{x \in \mathbf{conv}(S) \\ y = Ax}} q^T x - \ell(y) = \sup_{y = Ax} f^*(x) - \ell(y).$$

Rewriting the problem to remove the constraint $y = Ax$, we have

$$g(0) = \sup_x f^*(x) - \ell(Ax)$$

Since $\mathbf{conv}(S)$ is a compact set and ℓ is lower semi-continuous, there exists some $\tilde{x} \in \mathbf{conv}(S)$ which achieves this maximum. Now, using the definition of g (*cf.*, equation (10)),

$$g(0) = \sup_{x,y} f^*(x) - \ell(y) = \sup f^* - \inf \ell.$$

Putting both statements together, we find

$$f^*(\tilde{x}) - \ell(Ax^\star) = \sup f^* - \inf \ell.$$

Since we know, by definition of sup and inf that

$$f^*(\tilde{x}) \leq \sup f^* \quad \text{and} \quad \ell(A\tilde{x}) \geq \inf \ell,$$

then, putting these together with the above statements, we find that $\tilde{x}$ and $A\tilde{x}$ are minimizers for the first and second terms, respectively, *i.e.*,

$$f^*(\tilde{x}) = \sup f^* \quad \text{and} \quad \ell(A\tilde{x}) = \inf \ell.$$

This means that $\tilde{x} \in X^\star$ and $A\tilde{x} \in Y^\star$, or, equivalently, that $AX^\star \cap Y^\star$ is nonempty. The statement above follows from the contrapositive: if $AX^\star \cap Y^\star$ is nonempty, then $p = 0$ is not a minimizer for g.

Separating hyperplanes. The prices p that minimize the function g are intimately related to the geometry of the sets $X^\star$ and $Y^\star$. (We will see this soon.) For this purpose, we will define K to be the cone of hyperplanes that separate the sets $AX^\star$ and $Y^\star$, defined:

$$K = \{p' \in \mathbb{R}^m \mid p'^T Ax \geq p'^T y \text{ for all } x \in X^\star,\ y \in Y^\star\}.$$

Note that $0 \in K$, so this set is always nonempty, and K is closed and convex as it can be written as the intersection of closed halfspaces (which are also convex sets):

$$K = \bigcap_{\substack{x \in X^\star \\ y \in Y^\star}} \{p' \in \mathbb{R} \mid p'^T (Ax - y) \geq 0\}.$$

Conditions on prices. We will show that, if p satisfies $g(p) \leq g(0)$, then $p \in K$. In other words, any minimizer p of the dual function g must be a separating hyperplane of the extremizing sets $AX^\star$ and $Y^\star$. The proof is relatively simple. Since

$$g(p) \leq g(0) = \sup f^* - \inf \ell,$$

then, using the definition of g, we have that

$$f^*(x) - \ell(y) + p^T(y - Ax) \leq \sup f^* - \inf \ell,$$

for every $x \in \mathbf{conv}(S)$ and $y \in \mathbb{R}^m_+$, so

$$p^T(y - Ax) \leq \sup f^* - f^*(x) + \ell(y) - \inf \ell.$$

If we restrict x and y to be in $X^\star$ and $Y^\star$, respectively, and negate both sides, we then have

$$p^T(Ax - y) \geq 0,$$

or, that $p^T Ax \geq p^T y$ for all $x \in X^\star$ and $y \in Y^\star$, which is the definition of $p \in K$. (Note that we may replace the inequalities with strict inequalities and K with $\mathbf{int}\, K$, the interior of the cone K, to receive a second useful statement.)

Discussion. We note that the above definitions serve as a natural generalization of the condition that the resource utilization is equal to a target utilization $b^\star$. In our case, we can have many 'optimal' utilizations, given by $Y^\star$, with the additional granularity that any suboptimal resource utilization vector $y \notin Y^\star$ has a certain degree of displeasure, $\ell(y) > \inf \ell$. If the set of optimal utilizations (for the loss function ℓ) do not overlap with the zero-cost demand, $X^\star$, then the original condition states that the prices must be nonzero.

On the other hand, we know that any set of optimal prices p must be a separating hyperplane for the sets $AX^\star$ and $Y^\star$; *i.e.*, that $p \in K$. This leads to some interesting observations. If zero utilization is a possible target, *i.e.*, $0 \in Y^\star$, as is the case for any nondecreasing loss such as (7), then the set K is the dual cone of $\mathbf{cone}(AX^\star)$, where $\mathbf{cone}(AX^\star)$ is the set of conic (nonnegative) combinations of elements in $AX^\star$. For more information, see, *e.g.*, [BV04, §2.6].

Extensions. There is also a partial converse to the above conditions on the prices p. In particular, for any resource cost p in the interior of the cone $\mathbf{int}\, K$ (satisfying the technical condition that ℓ contains no line in the direction of p) there is always some scalar $t > 0$ such that $g(tp) < g(0)$; *i.e.*, 0 cannot be a minimizer for g if the interior of K is nonempty. While interesting, this point is slightly technical, so we defer it to appendix B. In general, we can view this statement as a stronger version of the original claim: if the compact set $AX^\star$ and closed set $Y^\star$ are disjoint, then there is a strict separating hyperplane between $AX^\star$ and $Y^\star$, say p, so the set $\mathbf{int}\, K$ is nonempty since it contains p. This, in turn, would immediately imply that $g(0)$ cannot be minimal.

3.3 Properties

There are a number of properties of the prices p that can be derived from the dual problem (12).

Nonnegative prices. If the objective function ℓ is separable and nondecreasing, as in (7), then any price p_i feasible for problem (12) must be nonnegative, $p_i \geq 0$. (By feasible, we mean that $g(p) < \infty$.) To see this, note that, by definition (7), we have

$$\ell^*(p) = \sup_y \left(p^T y - \ell(y)\right) = \sum_{i=1}^m \sup_{y_i} \left(p_i y_i - \phi_i(y_i)\right),$$

so we can consider each term individually. If $p_i < 0$ then any $y_i \leq 0$ must have

$$p_i y_i - \phi_i(y_i) \geq p_i y_i - \phi_i(0) \to \infty,$$

as $y_i \to -\infty$ since $\phi_i(y_i)$ is nondecreasing in y_i. So $g(p) \to \infty$ and therefore this choice of p cannot be feasible, so we must have that $p_i \geq 0$.

Superlinear separable losses. If the losses ϕ_i are superlinear, in that

$$\frac{\phi_i(z)}{z} \to \infty, \tag{14}$$

as $z \to \infty$ and bounded from below, in addition to being nondecreasing, then $\ell^*(p)$ is finite for $p \geq 0$. This means that the effective domain of g, defined as the set of prices for which g is finite,

$$\mathbf{dom}\, g = \{p \in \mathbb{R}^m \mid g(p) < \infty\},$$

is exactly the nonnegative orthant. (This discussion may appear somewhat theoretical, but we will see later that this turns out to be an important practical point when updating prices.) While not all losses are superlinear, we can always make them so by, *e.g.*, adding a small, nonnegative squared term to ϕ_i, say

$$\tilde{\phi}_i(z) = \phi_i(z) + \rho(z)_+^2,$$

where $(z)_+ = \max\{0, z\}$ and $\rho > 0$ is a small positive value, or by setting $\phi_i(z) = \infty$ for $z \geq 0$ large enough.

Subsidies. Alternatively, if the function ℓ is decreasing somewhere on the interior of its domain, then there exist points $y^\star$ for which prices p_i are negative—*i.e.*, sometimes the network is willing to subsidize usage by paying users to use the network to meet its intended target. The interpretation is simple: if the base demand of the network is not enough to meet the target amount, then the network has an incentive to subsidize users until the marginal cost of the target usage matches the subsidy amounts. We note that this may only apply to very specific transaction types in practice, as it is difficult to issue subsidies in an incentive-compatible manner that doesn't incentivize the inclusion of 'junk' transactions.

Maximum price. Another observation is that there often exist prices past which transaction producers would always prefer to not submit a transaction (or, more generally, will only submit transactions that consume no resources, if such transactions exist). In fact, we can characterize the set of all such prices.

To do this, write $S_0 \subseteq S$ for the set of transactions bundles that use no resources, defined

$$S_0 = \{x \in S \mid Ax = 0\}.$$

If $0 \in S$ then S_0 is nonempty (as $0 \in S_0$) and we usually expect this set to be a singleton, $S_0 = \{0\}$. Otherwise, we are saying that there are transactions that are always costless to include. Now, we will define the set

$$P = \bigcap_{x \in S \setminus S_0} \{p \in \mathbb{R}^m_+ \mid p^T Ax > q^T x\}.$$

This is the set of prices $p \in P$ such that, for every possible transaction bundle $x \in S$, the price of this transaction bundle, $p^T Ax$, paid to the network, is strictly larger than the total welfare generated by including it, which is $q^T x$. (That is, any transaction bundle x that is not costless is always strictly worse than no transaction, for transaction producers, at these prices.) The set P is nonempty since $\mathbf{1}^T Ax > 0$ for every $x \in S \setminus S_0$ (and $S \setminus S_0$ is finite) so, setting $p = t\mathbf{1}$, we have that

$$p^T Ax - q^T x = t\mathbf{1}^T Ax - q^T x \to \infty > 0,$$

as $t \to \infty$, so $t\mathbf{1} \in P$ for t large enough. The set P is also a convex set, as it is the intersection of convex sets. Additionally, if $p \in P$, then any prices p' satisfying $p' \geq p$ must also have $p' \in P$, where the inequality is taken elementwise. In English: if certain resource prices $p \in P$ would mean that transactions that consume resources are not included, then increasing the price of any resources to $p' \geq p$ also implies the same.

3.4 Solution methods

As mentioned before, the dual problem (12) is convex. This means that it can often be easily solved if the function g (or its subgradients) can be efficiently evaluated. We will see that, assuming users and miners are approximately solving problem (11), we can retrieve approximate (sub)gradients of g and use these to (approximately) solve the dual problem (12). In a less-constrained computational environment, a quasi-Newton method (*e.g.*, L-BFGS) would converge quickly to the optimal prices and be efficient to implement. However, these methods aren't amenable to on-chain computation due to their memory and computational requirements. To solve for the optimal fees on chain, we therefore propose a modified version of gradient descent which is easy to compute and does not require additional storage beyond the fees themselves.

Projected gradient descent. A common algorithm for unconstrained function minimization problems, such as problem (12), is *gradient descent*. In gradient descent, we are given an initial starting point p^0 and, if the function g is differentiable, we iteratively update the prices in the following way:

$$p^{k+1} = p^k - \eta \nabla g(p^k).$$

Here, $\eta > 0$ is some (usually small) positive number referred to as the 'step size' or 'learning rate' and $k = 0, 1, \ldots$ is the iteration number. This rule has a few important properties. For example, if $\nabla g(p^k) = 0$, that is, p^k is optimal, then this rule does not update the prices, $p^{k+1} = p^k$; in other words, any minimizer of g is a fixed point of this update rule. Additionally, this rule can be shown to converge to the optimal value under some mild conditions on g, *cf.* [Ber99, §1.2]. This update also has a simple interpretation: if $\nabla g(p^k)$ is not zero, then a small enough step in the direction of $\nabla g(p^k)$ is guaranteed to evaluate to a lower value than p^k, so an update in this direction decreases the objective g. (This is why the parameter η is usually chosen to be small.)

Note that if the effective domain of the function g, $\mathbf{dom}\, g$, is not $\mathbb{R}^m$, then it is possible that the $(k+1)$st step ends up outside of the effective domain, $p^{k+1} \notin \mathbf{dom}\, g$, so $g(p^k) = \infty$ which would mean that the gradient of g at price p^{k+1} would not exist. To avoid this, we can instead run *projected* gradient descent, where we project the update step into the domain of g, in order to get $p^{k+1} \in \mathbf{dom}\, g$, *i.e.*,

$$p^{k+1} = \mathbf{proj}(p^k - \eta \nabla g(p)) \tag{15}$$

where $\mathbf{proj}(z)$ is defined

$$\mathbf{proj}(z) = \operatorname*{argmin}_{p \in \mathbf{dom}\, g} \|z - p\|_2^2.$$

In English, $\mathbf{proj}(z)$ is the projection of the price to the nearest point in the domain of g, as measured by the sum-of-squares loss $\|\cdot\|_2^2$. (This always exists and is unique as the domain of g is always closed and convex, for any loss function ℓ as defined above.) There is relatively rich literature on the convergence of projected gradient descent, and we refer the reader to [Sho85; Ber99] for more.

Evaluating the gradient. In general, since we do not know q, we cannot evaluate the function g at a certain point, say p^k. On the other hand, the gradient of g at p^k, when g is differentiable, depends only on the solution to problem (11) and the maximizer for the dual function ℓ^*, at the price p^k. (This follows from the gradient equation in (13).) So, if we know that transaction producers are solving their welfare maximization problem (11) to (approximate) optimality, equation (13) suggests a simple descent algorithm for solving the dual problem (12).

From before, let $y^\star$ be a maximizer of $\sup_y \left(y^T p^k - \ell(y)\right)$, which is usually easy to compute in practice, and let x^0 be an (approximate) solution to the transaction inclusion problem (11) (observed, *e.g.*, after the block is built with resource prices p^k). We can approximate the gradient of g at the current fees p using (13), where we replace the true solution $x^\star$ with the observed solution x^0. Since x^0 is Boolean, we can compute the resource usage Ax^0 after

observing only the included transactions. We can then update the fees p^k in, say, block k, to a new value p^{k+1} by using projected gradient descent with this new approximation:

$$p^{k+1} = \mathbf{proj}(p^k - \eta(y^\star - Ax^0)). \tag{16}$$

Discussion. Whenever ℓ is differentiable at $y^\star$, we have that $\nabla\ell(y^\star) = p$. (To see this, apply the first-order optimality conditions to the objective in the supremum in the definition of $\ell^\star$.) We can then think of $y^\star$ as the resource utilization such that the marginal cost to the network $\nabla\ell$ is equal to the current fees p. Thus, the network aims to set p such that the realized resource utilization is equal to $y^\star$. We can see that (15) will increase the network fee for a resource being overutilized and decrease the network fee for a resource being underutilized. This pricing mechanism updates fees to disincentivize future users and miners from including transactions that consume currently-overutilized resources in future blocks. Additionally, we note that algorithms of this form are not the only algorithms which are reasonable. For example, any algorithm that has a fixed point p satisfying $\nabla g(p) = 0$ and converges to this point under suitable conditions is also similarly reasonable. One well-known example is an update rule of the form of (3):

$$p^{k+1} = p^k \odot \exp(-\eta\nabla g(p^k)),$$

when the prices must be nonnegative, *i.e.*, when $\mathbf{dom}\, g \subseteq \mathbb{R}^m_+$. We note that one important part of reasonable rules is that they only depend on (an approximation of) the gradient of the function g, since the value of g may not even be known in practice. Additionally, in some cases, the function g is nondifferentiable at prices p. In this case, the subgradient still often exists and convergence of the update rule (16) can be guaranteed under slightly stronger conditions. (The modification is needed as not all subgradients are descent directions.)

Simple examples. We can derive specific update rules by choosing specific loss functions. For example, consider the loss function

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ +\infty & \text{otherwise,} \end{cases}$$

which captures infinite unhappiness of the network designer for any deviation from the target resource usage $b^\star$. The corresponding conjugate function is

$$\ell^*(p) = \sup_y (y^T p - \ell(y)) = (b^\star)^T p,$$

with maximizer $y^\star = b^\star$. (Note that this maximizer does not change for any price p). Since $\mathbf{dom}\, g = \mathbb{R}^m$, then the update rule is

$$p^{k+1} = p^k - \eta(b^\star - Ax^0).$$

If the utilization Ax^0 lies far below $b^\star$, the fees p^k might become negative, *i.e.*, the network would want to subsidize certain resource usage to meet the requirement that it must be equal to $b^\star$.

Nondecreasing separable loss. A more reasonable family of loss functions would be the nondecreasing, separable losses:

$$\ell(y) = \sum_{i=1}^{m} \phi_i(y_i).$$

From §3.3 we know that the domain of g is precisely the nonnegative orthant when the functions ϕ_i are superlinear (*i.e.*, satisfy (14)) and bounded from below, so we have that

$$\mathbf{proj}(z) = (z)_+$$

where $(w)_+ = \max\{0, w\}$ for scalar w and is applied elementwise for vectors. Additionally, using the definition of the separable loss, we can write

$$\ell^\star(p) = \sum_{i=1}^{m} \sup_{y_i} \left(y_i p_i - \phi_i(y_i)\right).$$

Letting $y_i^\star$ be the maximizers for the individual problems at the current price p^k, we have

$$p^{k+1} = (p^k - \eta(y^\star - Ax^0))_+.$$

For example, if ϕ_i is an indicator function with $\phi_i(y_i) = 0$ if $y_i \leq b_i^\star$ and ∞ otherwise, as in the loss (5), then an optimal point is always $y_i^\star = b_i^\star$, when $p_i \geq 0$. Since no updates will ever set $p_i < 0$, we therefore have,

$$p^{k+1} = (p^k - \eta(b^\star - Ax^0))_+.$$

which is precisely the update given in (2).

Notes. While we used projected gradient descent rules for the examples above, we note that this class of update rules is not the only option. Other update rules naturally fall out of other optimization algorithms applied to (12). For example, if we only want to update some subset of the prices at each iteration, we can use block coordinate descent. We can also add adaptive step size rules or momentum terms to our gradient descent formulation. These additions would yield more computationally expensive algorithms, but they may result in faster convergence to optimal prices when the distribution of processed transactions shifts. This is a potentially interesting area for future research.

4 The cost of uniform prices

In this section, we show that pricing resources using the method outlined above can increase network efficiency and make the network more robust to DoS attacks or distribution shifts. We construct a toy experiment to illustrate these differences between uniform and multi-dimensional resource pricing, and leave more extensive numerical studies to future work.

The setup. We consider a blockchain system with two resources (*e.g.*, compute and storage) with resource utilizations y_1 and y_2. Resource 1 is much cheaper for the network to use than resource 2, so

$$b^\star = \begin{bmatrix} 10 \\ 1 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \leq b = \begin{bmatrix} 50 \\ 5 \end{bmatrix}.$$

Furthermore, we assume that there is a joint capacity constraint on these resources

$$y_1 + 10y_2 \leq 50,$$

which captures the resource tradeoff. Each transaction a_j is therefore a vector in $\mathbb{R}^3_+$ with

$$a_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ a_{1j} + 10a_{2j} \end{bmatrix}.$$

As in §3.4, we consider the simple loss function

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ +\infty & \text{otherwise,} \end{cases}$$

which has update rule

$$p^{k+1} = p^k - \eta(b^\star - Ax^0).$$

In the scenarios below, we compare our multi-dimensional fee market approach to a baseline, where both resources are combined into one equal to $a_{1j}+10a_{2j}$ with $b^\star = 20\% \times \max(b_1, b_2) = 10$. We demonstrate that pricing these resources separately leads to better network performance. All code is available at

`https://github.com/bcc-research/resource-pricing`.

We run simulations in the Julia programming language [Bez+17]. The transaction producers' optimization problem (11) is modeled with JuMP [DHL17] and solved with COIN-OR's simplex-based linear programming solver, Clp [For+22]. The solution is usually integral, but when it is not, we fall back to the HiGHS mixed-integer linear program solver [HH18].

Scenario 1: steady state behavior. We consider a sequence of 250 blocks. At each block, there are 15 submitted transactions, with resource usage randomly drawn as $a_{1j} \sim \mathcal{U}(0.5, 1)$ and $a_{2j} \sim \mathcal{U}(0.05, 0.1)$. (For example, these may be moderate compute and low storage transactions.) Transaction utility is drawn as $q_j \sim \mathcal{U}(0, 5)$. We initialize the price vector as $p = 0$ and examine the steady state behavior, where the price updates and miner behavior are defined as in the previous section. We use a learning rate $\eta = 1 \times 10^{-2}$ throughout.

The resource utilization, shown in figure 1, suggests that our multidimensional scheme more closely tracks the target utilities $b^\star$ than a single-dimensional fee market. Figure 2 shows the squared deviation from the target resource utilizations. Furthermore, the number of transactions included per block is consistently higher, illustrated in figure 3 (thicker line).

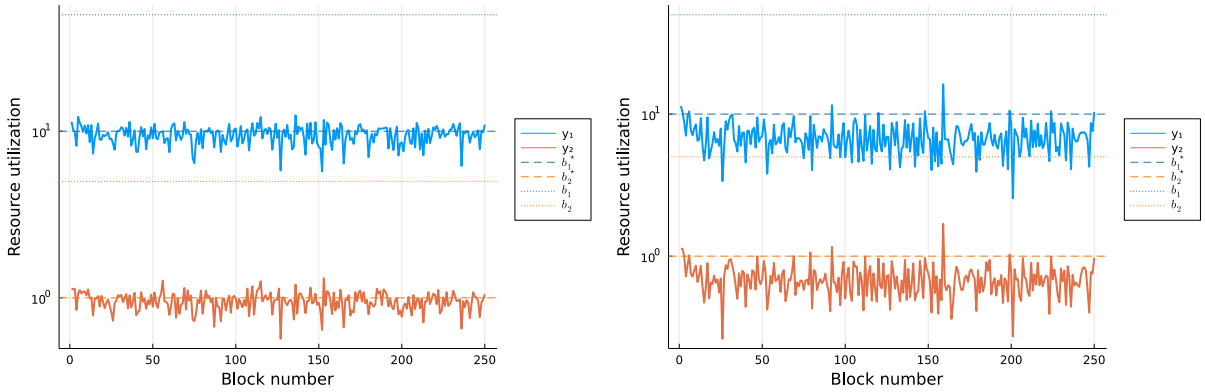

Figure 1: Resource utilization for multidimensional pricing (left) clusters closer to target values than for uniform pricing (right), which includes limited information about individual targets or caps.

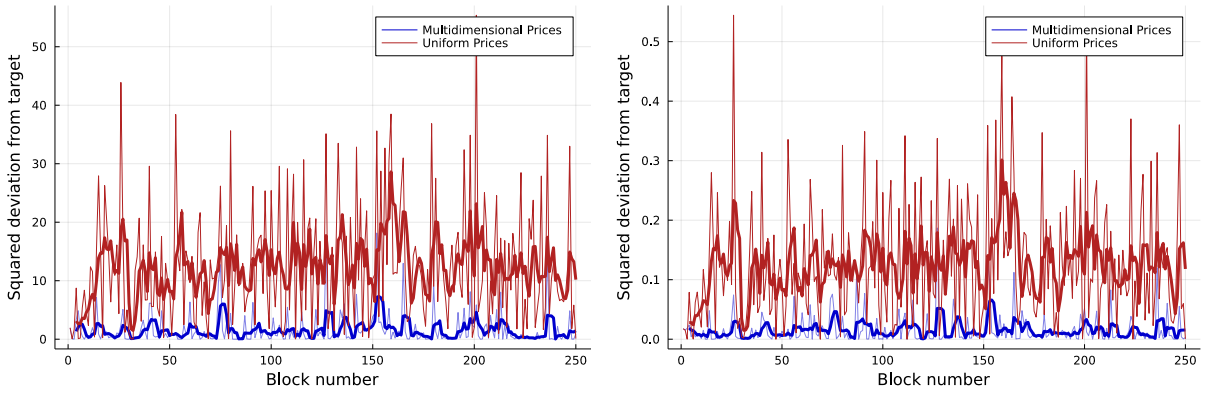

Figure 2: Squared deviation from the target values given by $b^\star$ in resource 1 utilization (left) and resource 2 utilization (right) for both uniform and multidimensional pricing. The thicker line is the four-sample moving average.

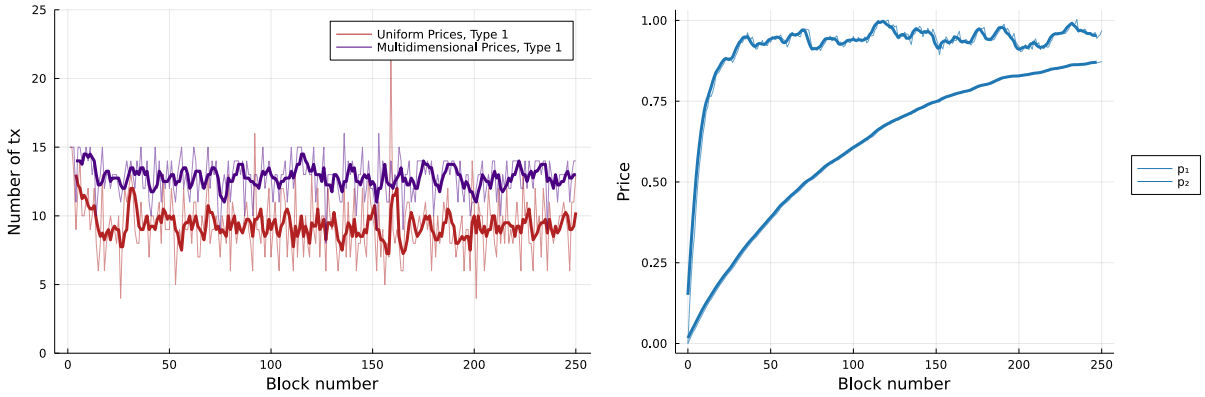

Figure 3: Multidimensional pricing allows us to include more transactions per block (left) by optimally adjusting prices (right). The thicker line is the four-sample moving average of the data.

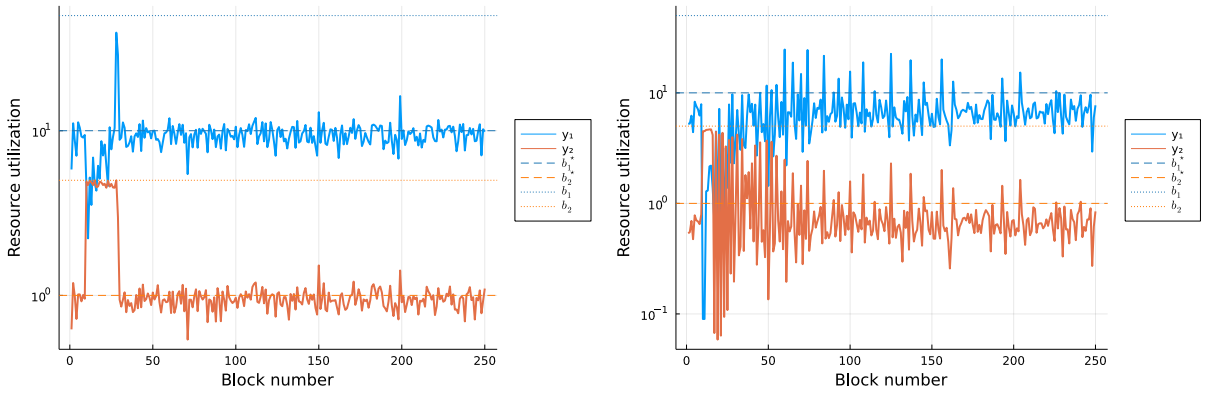

Figure 4: Resource utilization for multidimensional pricing (left) clusters closer to target values than for uniform pricing (right) after a burst to the resource limit to handle transactions that make heavy use of resource 2.

Scenario 2: transactions distribution shift. Often, the distribution of transaction types submitted to a blockchain network differs for a short period of time (*e.g.*, during NFT mints). There may be a change in both the number of transactions and the distribution of resources required. We repeat the above simulation but add 150 transactions in block 10; each transaction has a resource vector $a_j = (0.01, 0.5)$. (For example, these transactions may have low computation but high storage requirements.) We draw the utility $q_j \sim \mathcal{U}(10, 20)$ and begin the network at the steady-state prices from scenario 1.

In figure 4, we see that a multidimensional fee market gracefully handles the distribution shift. The network fully utilizes resource 2 for a short period of time before returning to steady state. Uniform pricing, on the other hand, does not do a good job of adjusting its resource usage. Figure 5 show that, as a result, multidimensional pricing is able to include more transactions, both during the distribution shift and after the network returns to steady state. We see that the prices smoothly adjust accordingly.

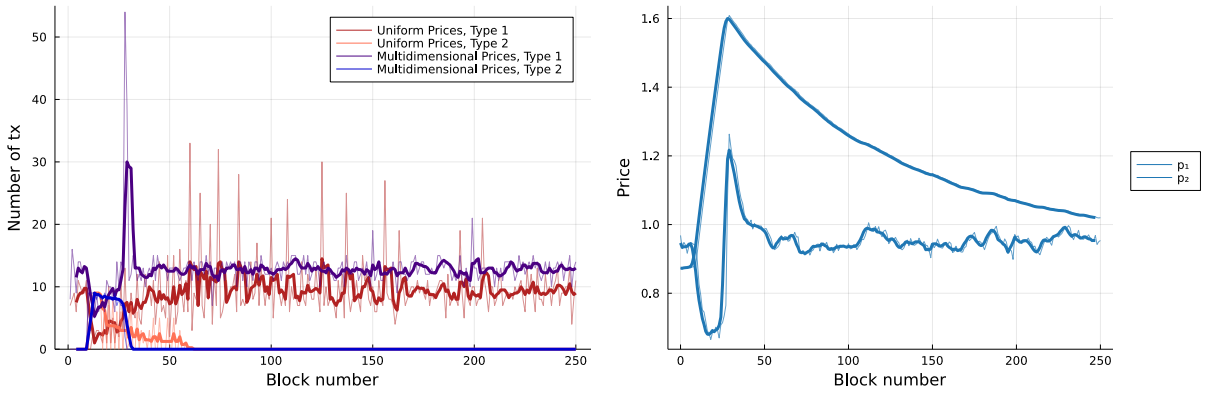

Figure 5: Multidimensional pricing allows us to include more transactions per block by optimally adjusting prices (right). The thicker line is the four-sample moving average of the data.

5 Extensions

Parallel transaction execution model. Consider the scenario where the nodes have L parallel execution environments (*e.g.*, threads), each of which has its own set of m identical resources. In addition, there are r resources shared between the environments. We denote transactions run on thread k by $x_k \in \{0,1\}^m$. The resource allocation problem becomes

$$\begin{array}{ll} \text{maximize} & \sum_k q^T x_k - \ell(y_1, \ldots, y_k, y^{\text{shared}}) \\ \text{subject to} & y_k = A x_k \\ & z = \sum_k x_k \\ & y^{\text{shared}} = Bz \\ & z \in \mathbf{conv}(S^{\text{shared}}) \\ & x_k \in \mathbf{conv}(S). \end{array}$$

As before, the Boolean vector sets S and S^{shared} encode constraints such as resource limits for each parallel environment and the shared environment respectively. In addition, we'd expect to have $z \leq \mathbf{1}$ if each transaction is only allocated to a single environment, which can be encoded by S^{shared}. By stacking the variables into one vector, this problem can be seen as a special case of (8) and can be solved with the same method presented in this work. (The interpretation here is that we are declaring a number of 'combined resources', each corresponding to the parallel execution environments along with their shared resources.)

Different price update speeds. Some resources may be able to sustain burst capacities for much shorter periods of times than other resources. In practice, we may wish to increase the prices of these resources faster. For example, a storage opcode that generates a lot of memory allocations will quickly cause garbage collection overhead, which could slow down the network. As a result, we likely want to increase its price faster than the prices of basic arithmetic, even under the same utilization. To do this, we can update (15) to include a

learning rate for each resource. We collect these in a diagonal matrix $D = \mathbf{diag}(\eta_1, \ldots, \eta_m)$:

$$p^{k+1} = \mathbf{proj}(p^k - D\nabla g(p))$$

These learning rates can be chosen by system designers using simulations and historical data.

Contract throughput. Alternatively, we can define utilization on a per-contract basis instead of a per-resource basis (per-contract fees were recently proposed by the developers of Solana [Yak21]). We define the utilization of a smart contract j as $z_j = (w^T a_j)x_j$, where w is some weight vector and $x_j \in \mathbb{Z}_+$ is the number of times contract j is called. In matrix form, $z = A^T w \odot x$, where $\odot$ is the Hadamard (elementwise) product. For each contract, the utilization z_j is 0 when $x_j = 0$, which can be interpreted as not calling contract j in a block. When $x_j > 0$, the utilization is $(\sum_i w_i a_{ij})\, x_j$. When we use per-contract utilizations, the loss function can capture a notion of fairness in resource allocation to contracts. For example, we may want to prioritize cheaper-to-execute contracts over more expensive ones by using, *e.g.*, proportional fairness as in [Kel97], though there are many other notions that may be useful. With this setup, the resource allocation problem is

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(z) \\ \text{subject to} & z = A^T w \odot x \\ & x \in \mathbf{conv}(S). \end{array}$$

Again, we can introduce the dual variable $p \in \mathbb{R}^n$ for the equality constraint, and, with a similar method to the one introduced in this paper, iteratively update this variable to find the optimal fees to charge for each smart contract call.

6 Conclusion

We constructed a framework for multi-dimensional resource pricing in blockchains. Using this framework, we modeled the network designer's goal of maximizing transaction producer utility, minus the loss incurred by the network, as a an optimization problem. We used tools from convex optimization — and, in particular, duality theory — to decompose this problem into two simpler problems: one solved on chain by the network, and another solved off chain by the transaction producers. The prices that unify the competing objectives of minimizing network loss and maximizing transaction producer utility are precisely the dual variables in the optimization problem. Setting these prices correctly (*i.e.*, to minimize the dual function) results in a solution to the original problem. We then demonstrated efficient methods for updating prices that are amenable to on-chain computation. Finally, we numerically illustrate, via a simple example, the proposed pricing mechanism. We find that it allows the network to equilibrate to its resource utilization target more quickly than the uniform price case, while offering greater throughput without increasing node hardware requirements.

To the best of the authors' knowledge, this is the first work to systematically study optimal pricing of resources in blockchains in the many-asset setting. Future work and improvements to this model include a detailed game-theoretic analysis, extending that of [Fer+21], along with a more concrete analysis of the dynamical behavior of fees set in this manner. Finally, a more thorough numerical evaluation of these methods under realistic conditions (such as testnets) will be necessary to see if these methods are feasible in production.

Acknowledgements

We would like to thank John Adler, Vitalik Buterin, Dev Ojha, Kshitij Kulkarni, Matheus Ferreira, Barnabé Monnot, and Dinesh Pinto for helpful conversations, insights, and edits. We're especially appreciative to John Adler for bearing with us through many drafts of this work and consistently providing valuable feedback.

References

[AB19] Mustafa Al-Bassam. *LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts*. May 22, 2019.

[ABSB18] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. *Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities*. 2018. DOI: 10.48550/ARXIV.1809.09044. URL: https://arxiv.org/abs/1809.09044.

[Adl19a] John Adler. *EIP-2242: Transaction Postdata*. 2019. URL: https://eips.ethereum.org/EIPS/eip-2242.

[Adl19b] John Adler. *Multi-Threaded Data Availability On Eth 1*. Ethresearch. 2019. URL: https://ethresear.ch/t/multi-threaded-data-availability-on-eth-1/5899.

[Adl20] John Adler. *Accounts, Strict Access Lists, and UTXOs - Research / Execution*. Celestia Forum. Sept. 21, 2020. URL: https://forum.celestia.org/t/accounts-strict-access-lists-and-utxos/37 (visited on 08/01/2022).

[Adl21] John Adler. "Wait, It's All Resource Pricing?" EthCC. 2021. URL: https://www.youtube.com/watch?v=YoWMLoeQGeI.

[Adl22] John Adler. "Always Has Been (or, Wait, It's All Resource Pricing? Part 2)". EthCC. 2022. URL: https://www.youtube.com/watch?v=Zq8uwpX39oI.

[Agr+22] Akshay Agrawal et al. "Allocation of fungible resources via a fast, scalable price discovery method". In: *Mathematical Programming Computation* (2022), pp. 1–30.

[Ber99] Dimitri P Bertsekas. *Nonlinear Programming*. 3rd ed. Athena Scientific, 1999.

[Bez+17] Jeff Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM review* 59.1 (2017), pp. 65–98.

[BS20a] Vitalik Buterin and Martin Swende. *EIP-2929: Gas cost increases for state access opcodes*. 2020. URL: https://eips.ethereum.org/EIPS/eip-2929.

[BS20b] Vitalik Buterin and Martin Swende. *EIP-2930: Optional access lists*. Ethereum Improvement Proposals. Aug. 29, 2020. URL: https://eips.ethereum.org/EIPS/eip-2930 (visited on 08/01/2022).

[But] Vitalik Buterin. *State of research: Increasing censorship resistance of transactions under proposer/builder separation (PBS)*. URL: https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance.

[But16a] Vitalik Buterin. *EIP 150: Gas cost changes for IO-heavy operations*. 2016. URL: https://eips.ethereum.org/EIPS/eip-150.

[But16b] Vitalik Buterin. *Geth nodes under attack again*. 2016. URL: https://www.reddit.com/r/ethereum/comments/55s085/geth_nodes_under_attack_again_we_are_actively/?st=itxh568s&sh=ee3628ea.

[But16c] Vitalik Buterin. *Transaction spam attack: Next Steps*. 2016. URL: https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/.

[But17] Vitalik Buterin. *Easy parallelizability Issue #648 ethereum/EIPs*. GitHub. June 17, 2017. URL: https://github.com/ethereum/EIPs/issues/648 (visited on 08/01/2022).

[But18] Vitalik Buterin. *First and second-price auctions and improved transaction-fee markets*. 2018. URL: https://ethresear.ch/t/first-and-second-price-auctions-and-improved-transaction-fee-markets/2410.

[But+19] Vitalik Buterin et al. *EIP-1559: Fee market change for ETH 1.0 chain*. 2019. URL: https://eips.ethereum.org/EIPS/eip-1559.

[But21] Vitalik Buterin. *An Incomplete Guide to Rollups*. vitalik. Jan. 5, 2021. URL: https://vitalik.ca/general/2021/01/05/rollup.html (visited on 08/01/2022).

[But22a] Vitalik Buterin. *Multidimensional EIP 1559*. Ethereum Research. 2022. URL: https://ethresear.ch/t/multidimensional-eip-1559/11651.

[But22b] Vitalik Buterin. *Multidimensional EIP 1559*. Ethresearch. 2022. URL: https://ethresear.ch/t/multidimensional-eip-1559/11651.

[But22c] Vitalik Buterin. *Proto-Danksharding FAQ*. HackMD. July 30, 2022. URL: https://notes.ethereum.org/@vbuterin/proto_danksharding_faq (visited on 08/01/2022).

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Che+21] Yang Chen et al. "Forerunner: Constraint-based Speculative Transaction Execution for Ethereum (Full Version)". In: Operating Systems Principles(SOSP, 21), October 26-29, 2021, Virtual Event, Germany. New York, NY: ACM, 2021. DOI: 10.1145/3477132.3483564.

[CS21] Hao Chung and Elaine Shi. "Foundations of transaction fee mechanism design". In: *arXiv preprint arXiv:2111.03151* (2021).

[Dai+20] Philip Daian et al. "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 910–927.

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A modeling language for mathematical optimization". In: *SIAM review* 59.2 (2017), pp. 295–320.

[Eth] *Run a Node*. Ethereum Foundation. July 26, 2022. URL: https://ethereum.org/en/run-a-node/ (visited on 08/02/2022).

[Fer+21] Matheus VX Ferreira et al. "Dynamic posted-price mechanisms for the blockchain transaction-fee market". In: *Proceedings of the 3rd ACM conference on Advances in Financial Technologies*. 2021, pp. 86–99.

[For+22] John Forrest et al. *coin-or/Clp: Release releases/1.17.7*. Version releases/1.17.7. Jan. 2022. DOI: 10.5281/zenodo.5839302. URL: https://doi.org/10.5281/zenodo.5839302.

[Gel+22] Rati Gelashvili et al. *Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing*. 2022. DOI: 10.48550/ARXIV.2203.06871. URL: https://arxiv.org/abs/2203.06871.

[HH18] Qi Huangfu and JA Julian Hall. "Parallelizing the dual revised simplex method". In: *Mathematical Programming Computation* 10.1 (2018), pp. 119–142.

[Hoc98] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[KDC22] Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. *Towards a Theory of Maximal Extractable Value I: Constant Function Market Makers*. 2022. arXiv: 2207.11835 [cs.GT].

[Kel97] Frank Kelly. "Charging and rate control for elastic traffic". In: *European transactions on Telecommunications* 8.1 (1997), pp. 33–37.

[Lab] Fuel Labs. *GitHub - FuelLabs/fuel-specs: Specifications for the Fuel protocol and the FuelVM, a blazingly fast blockchain VM*. GitHub. URL: https://github.com/FuelLabs/fuel-specs (visited on 08/01/2022).

[Leo+21] Stefanos Leonardos et al. "Dynamical analysis of the eip-1559 ethereum fee market". In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 2021, pp. 114–126.

[LL99] Steven H Low and David E Lapsley. "Optimization flow control. I. Basic algorithm and convergence". In: *IEEE/ACM Transactions on networking* 7.6 (1999), pp. 861–874.

[Low03] Steven H Low. "A duality model of TCP and queue management algorithms". In: *IEEE/ACM Transactions On Networking* 11.4 (2003), pp. 525–536.

[NNT21] Kamilla Nazirkhanova, Joachim Neu, and David Tse. *Information Dispersal with Provable Retrievability for Rollups*. 2021. DOI: 10.48550/ARXIV.2111.12323. URL: https://arxiv.org/abs/2111.12323 (visited on 08/01/2022).

[PL19] Daniel Perez and Benjamin Livshits. "Broken metre: Attacking resource metering in EVM". In: *arXiv preprint arXiv:1909.07220* (2019).

[Pol21] Polygon Team. *Introducing Avail by Polygon, a Robust General-Purpose Scalable Data Availability Layer*. Polygon | Blog. June 28, 2021. URL: https://blog.polygon.technology/introducing-avail-by-polygon-a-robust-general-purpose-scalable-data-availability-layer/ (visited on 08/01/2022).

[QZG21] Kaihua Qin, Liyi Zhou, and Arthur Gervais. "Quantifying blockchain extractable value: How dark is the forest?" In: *arXiv preprint arXiv:2101.05511* (2021).

[Rei+21] Daniël Reijsbergen et al. "Transaction Fees on a Honeymoon: Ethereum's EIP-1559 One Month Later". In: *2021 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2021, pp. 196–204.

[Rou21] Tim Roughgarden. "Transaction Fee Mechanism Design". In: *SIGecom Exch.* 19.1 (2021), 52–55. DOI: 10.1145/3476436.3476445. URL: https://doi.org/10.1145/3476436.3476445.

[SH19] Vikram Saraph and Maurice Herlihy. *An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts*. 2019. DOI: 10.48550/ARXIV.1901.01376. URL: https://arxiv.org/abs/1901.01376 (visited on 08/01/2022).

[Sho85] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*. Vol. 3. Springer Series in Computational Mathematics, 1985.

[Tas+22] Ertem Nusret Tas et al. *Light Clients for Lazy Blockchains*. _eprint: 2203.15968. 2022. (Visited on 08/01/2022).

[Wil16] Jeffrey Wilcke. *The Ethereum network is currently undergoing a DoS attack*. Ethereum Foundation. Sept. 22, 2016. URL: https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/ (visited on 08/01/2022).

[Yak20] Anatoly Yakovenko. *Sealevel, Parallel Processing Thousands of Smart Contracts*. Solana. Feb. 23, 2020. URL: https://medium.com/solana-labs/sealevel-parallel-processing-thousands-of-smart-contracts-d814b378192 (visited on 08/01/2022).

[Yak21] Anatoly Yakovenko. *Consider increasing fees for writable accounts Issue #21883 solana-labs/solana*. GitHub. Dec. 14, 2021. URL: `https://github.com/solana-labs/solana/issues/21883` (visited on 08/01/2022).

A A (very short) primer on convexity

This appendix serves as a short introduction to the basic notions of convexity used in this paper for readers familiar with basic real analysis and linear algebra. For (much) more, we recommend [BV04].

A.1 Basic definitions

Convexity. We say a set $S \subseteq \mathbb{R}^m$ is convex if, for any two points $x, y \in S$ and any $0 \leq \gamma \leq 1$, we have

$$\gamma x + (1-\gamma)y \in S.$$

In other words, a set S is convex if it contains all line segments between any two points in S. A classic example of a closed convex set is a closed halfspace

$$H = \{x \in \mathbb{R}^m \mid a^T x \leq b\},$$

where $a \in \mathbb{R}^m$ and $b \in \mathbb{R}$. (An otherwise silly but useful example is the empty set, which vacuously meets the requirements.)

We say a function over the extended reals $f : S \to \mathbb{R} \cup \{\infty\}$ is convex if S is convex and, for any $x, y \in S$ and $0 \leq \gamma \leq 1$,

$$f(\gamma x + (1-\gamma)y) \leq \gamma f(x) + (1-\gamma)f(y).$$

Equivalently: a function is convex if any chord of the function (a line segment between two points on its graph) lies above (or, strictly speaking, not below) the function itself. We say f is concave if $-f$ is convex. Some basic functions that are convex are linear functions $f(x) = q^T x$ for some $q \in \mathbb{R}^n$, norms $f(x) = \|x\|$ and indicator functions of convex sets:

$$f(x) = \begin{cases} 0 & x \in S \\ \infty & \text{otherwise,} \end{cases}$$

where $S \subseteq \mathbb{R}^m$ is a convex set.

Domain. Usually it is simpler to work with functions f defined over all of $\mathbb{R}^m$ as opposed to just a subset. We may extend the functions $f : S \to \mathbb{R} \cup \{\infty\}$ to a function $\tilde{f} : \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}$ by setting $\tilde{f}(x) = f(x)$ for $x \in S$ and $\tilde{f}(x) = \infty$ if $x \notin S$. We write the *effective domain* of a function f defined over $\mathbb{R}^m$ as

$$\mathbf{dom}\, f = \{x \in \mathbb{R}^m \mid f(x) < \infty\}.$$

Throughout the paper and the remainder of this appendix, we assume that all functions are extended in this way.

Characterizations of convexity. There are many equivalent characterizations of convexity for functions. A particularly useful one, if the function f is differentiable over its domain $\mathbf{dom}\, f$, is, for any two points $x, y \in \mathbf{dom}\, f$, we have:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x). \tag{17}$$

In other words, any tangent plane to f at x is a global underestimator of the function. A common characterization for twice-differentiable functions f is that the hessian (the matrix of all second derivatives) of f at every point is positive semidefinite, but this is often particularly restrictive. The gradient-based definition of convexity immediately implies that if we find a point x with $\nabla f(x) = 0$, then

$$f(y) \geq f(x),$$

for any $y \in \mathbf{dom}\, f$. Thus, x is a global minimizer of f. (The converse is similarly easy to show.)

Consequences of convexity. There are a number of important consequences of convexity. The simplest is: given two closed convex sets $S, T \subseteq \mathbb{R}^n$ with $S \cap T = \emptyset$ then there exists a vector $p \in \mathbb{R}^n$ and $p \neq 0$ that separates these sets, *i.e.*,

$$p^T (x - y) \geq 0, \text{ for all } x \in S,\ y \in T.$$

If one of the sets, say S, is also compact, then it is possible to make the stronger claim that there exists $p' \in \mathbb{R}^m$ and $\varepsilon > 0$ such that

$$p^T (x - y) \geq \varepsilon, \text{ for all } x \in S,\ y \in T.$$

The arguments for each of these are relatively simple and can be found in [BV04, §2.5]. Nearly all major results in convex optimization theory are a consequence of these two facts.

Convexity-preserving operations on sets. There are a number of operations which preserve convexity of sets. For example, any (potentially uncountable) intersection of convex sets is convex. The (finite) sum of convex sets $S, T \subseteq \mathbb{R}^m$, defined

$$S + T = \{x + y \mid x \in S,\ y \in T\},$$

is convex, while negation of a set $-S = \{-x \mid x \in S\}$ is also convex. Any linear function of a convex set, say $A \in \mathbb{R}^{n \times m}$ with $AS = \{Ax \mid x \in S\}$, is convex. In the special case that S is also compact, then AS is compact. (It is, on the other hand, not true in general that if S is closed then AS is closed.) All of these conditions can be easily verified from the definitions above. Additionally, there are other operations that preserve convexity, such as the perspective transform [BV04, §2.3.3], but we do not need those here.

Convexity-preserving operations on functions. Similar to the case with convex sets, there are a number of convexity-preserving operations on convex functions. The sum of convex functions is convex, while any nonnegative scaling $\gamma \geq 0$ of a convex function f is convex, *i.e.*, γf is convex. Affine precomposition of convex functions is convex, *i.e.*, if $f(x)$ is convex over $x \in \mathbb{R}^m$ then $f(Ay+b)$ is convex over $y \in \mathbb{R}^n$, for any $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Convexity is preserved over suprema:

$$f(x) = \sup_{y\in Y} f_y(x)$$

where f_y is a family of convex functions indexed by some (potentially uncountable) set Y. The set Y has no assumptions on it, other than being nonempty. A classic example is if

$$f(x) = \sup_{y\in Y} \left\{y^T x - g(y)\right\},$$

where $g : Y \to \mathbb{R} \cup \{\infty\}$ is any (potentially nonconvex) function and $Y \subseteq \mathbb{R}^m$ is any nonempty set, then the function f is convex as it is the supremum of a family of affine (and therefore convex) functions of x. (The function f is known as the *Fenchel conjugate*, often just the conjugate, of g and is denoted $g^*(x)$.) As before, all of these statements are easy to show given the definitions provided above.

Lines. In some cases it is important that convex functions have at least a certain amount of growth. To this end, we say the convex function f *contains a line* $p \in \mathbb{R}^m$ with $p \neq 0$ if, for some $x_0 \in \mathbb{R}^m$ and $q \in \mathbb{R}$, we have

$$f(x_0 + tp) \leq f(x_0) + tq,$$

for every $t \geq 0$. We say that f *contains a line in the direction of* $r \in \mathbb{R}^m$ if it contains a line p with $r^T p > 0$. It is not hard to show that functions which satisfy

$$\frac{f(tp)}{t} \to \infty$$

as $t \to \infty$ for every $p \neq 0$ contain no lines. Examples of such functions are set indicator functions for compact sets and square norms, $f(x) = \|x\|^2$.

Sublevel sets and semicontinuity. The α-sublevel set of a function f is defined as

$$S_\alpha = \{x \in \mathbb{R}^m \mid f(x) \leq \alpha\}.$$

If the function f is convex, the set S_α is also convex for every α. If the sublevel sets S_α are closed for every α, then we say the function f is *lower semicontinuous* (some authors say the function f is closed, instead). Any continuous function is lower semicontinuous, and any lower semicontinuous function f minimized over a set S such that $\mathbf{dom}\, f \cap S$ is compact achieves its minimum within that set

$$\inf_{x\in S} f(x) = f(x^\star),$$

for some $x^\star \in S$.

A.2 Convex hulls

Sometimes the sets that we deal with are not convex, but, in some special cases, we may treat them as if they were.

Convex hull. To this end, define the *convex hull* of a set $S \subseteq \mathbb{R}^m$, written $\mathbf{conv}(S)$, as the set containing all convex combinations of points in S. Here, a *convex combination* of points in S, say $x_i \in S$ for $i = 1, \ldots, n$, is any point $y \in \mathbb{R}^m$ (potentially not in S) which can be written as

$$y = \gamma_1 x_1 + \cdots + \gamma_n x_n,$$

where $\gamma_i \geq 0$ and $\sum_{i=1}^n \gamma_i = 1$. The set $\mathbf{conv}(S)$ is evidently convex (we encourage the reader to show this) and is, in fact, the smallest convex set which contains S. If the set S is compact, then its convex hull is also compact.

Linear functions. One special case where we may replace a set S with its convex hull is the maximization of linear functions. That is, for $S \subseteq \mathbb{R}^m$ and $q \in \mathbb{R}^m$, we have

$$\sup_{x \in S} q^T x = \sup_{x \in \mathbf{conv}(S)} q^T x$$

To see this, we will show that any point $x \in \mathbf{conv}(S)$ with value $q^T x$ has a point $y \in S$ with $q^T y \geq q^T x$. Since $x \in \mathbf{conv}(S)$, then it can be written as

$$x = \gamma_1 x_1 + \cdots + \gamma_n x_n,$$

where $x_i \in S$ and $\gamma_i \geq 0$ with $\sum_{i=1}^n \gamma_i = 1$. So we have that

$$q^T x = \gamma_1 q^T x_1 + \cdots + \gamma_n q^T x_n \leq (\gamma_1 + \cdots + \gamma_n) q^T x_j = q^T x_j,$$

where j is the index for which $q^T x_j$ is the largest, *i.e.*, $q^T x_j \geq q^T x_i$ for $i = 1, \ldots, n$. Since $x_j \in S$, then we are done. (This proof can be extended from linear functions to quasiconvex functions nearly immediately, but we only use this special case here.) The other direction immediately follows from the fact that $S \subseteq \mathbf{conv}(S)$. We note that while the maximum values of these two problems are the same, the set of maximizers may not be.

Derivatives. If the set S is finite, then the function

$$g(p) = \max_{x \in S} p^T x$$

is differentiable at points p when the optimal point $x^\star \in S$ is unique, and has $\nabla g(p) = x^\star$. This is easy to see, as $x^\star$ is unique only when

$$p^T x^\star > p^T x, \quad x \in S \setminus \{x^\star\},$$

so $g(p') = p'^T x^\star$ for p' in a neighborhood of p. Differentiating both sides gives the result.

A.3 Cones

A special case of convex sets are the convex cones. A set $K \subseteq \mathbb{R}^m$ is a *cone* if $x \in K$ means that $\gamma x \in K$ for any $\gamma \geq 0$. A cone K is a *convex cone* if K is also convex, which means that it is closed under conic (nonnegative) combinations: *i.e.*, for any $\gamma_i \geq 0$ and $x_i \in K$ for $i = 1, \ldots, n$, we have

$$\gamma_1 x_1 + \cdots + \gamma_n x_n \in K.$$

Convex cones are generally useful as they imply a partial ordering with respect to their elements, given by $x \succeq y$ if $x - y \in K$, and this ordering is closed under nonnegative multiplications and additions of inequalities. See [BV04, §2.4 & 2.6] for more information on cones and conic duality.

A.4 Convex optimization problems

In many cases, we are interested in optimization problems over convex sets and convex functions. We write a general *optimization problem* as

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in S, \end{array}$$

with variable $x \in \mathbb{R}^n$, objective function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, and constraints $x \in S \subseteq \mathbb{R}^n$. We say $x^\star$ is a *solution* to the problem if $x^\star \in S$ and $f(x) \geq f(x^\star)$ for all $x \in S$. Note that solutions may or may not exist, depending on f and S. Alternatively, we may switch 'minimize' for 'maximize,' but maximizing f over S is the same as minimizing $-f$ over S. Unlike a solution, the *optimal value* of this problem always exists and is equal to

$$\inf_{x \in S} f(x).$$

For convenience, we will say that if $S \cap \mathbf{dom}\, f = \emptyset$ the optimal value of this problem is ∞. (A minimizing sequence also always exists, but we do not use this notion here.)

Convex problems. If the objective function f and the set S are convex, we say that the problem is a *convex optimization problem*. From before, if we also know that the set $S \cap \mathbf{dom}\, f$ is compact and f is lower semicontinuous, then there is a solution to the problem, $x^\star \in S$. There is a very rich set of results for problems of this form, namely the theory of duality.

Dual problem. We will focus on the special case where S is an affine set and the remainder of the constraints are part of the effective domain of f, which is the case we use in this paper. This problem can be written as

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b, \end{array}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are the problem data and $x \in \mathbb{R}^n$ is our problem variable. For future reference, we will call the optimal value of this problem $s^\star$. We introduce a function called the *Lagrangian*, by relaxing the constraints to some price penalties $p \in \mathbb{R}^m$ (often called dual variables):

$$L(x, p) = f(x) + p^T(Ax - b).$$

The *dual function* is the optimal value of optimizing the (relaxed) objective $L(\cdot, p)$ against fixed price penalties p, instead of hard constraints:

$$g(p) = \inf_x L(x, p) = \inf_x \left(f(x) + p^T(Ax - b) \right).$$

Since any point x that is feasible for the original problem (*i.e.*, satisfies $Ax = b$) will suffer no penalty for any price p, we must have that

$$g(p) = \inf_x L(x, p) \leq \inf_{Ax=b} L(x, p) = \inf_{Ax=b} f(x) = s^\star.$$

In other words, for any price p, the value $g(p)$ is a lower bound to the optimal value.

One of the most important results in convex optimization is that, under a certain technical assumption, there exists some prices $p^\star$ for which this inequality is tight. More specifically, if there is a point in the relative interior of the domain of f, $x \in \mathbf{relint}\,\mathbf{dom}\, f$, then there exists a set of prices $p^\star$ with

$$g(p^\star) = s^\star.$$

Of course, since we know that $g(p) \leq s^\star$ for all p, then

$$g(p^\star) = \sup_p g.$$

So, to find an optimal set of prices, it suffices to find a point $p^\star$ that maximizes g, if we know that such prices exist. This problem:

$$\text{maximize} \quad g(p)$$

over p is known as the *dual problem* (the original problem is also referred to as the *primal problem*) and has the same optimal objective value.

A.5 Subdifferentials

In general, many convex functions are not differentiable, and, unfortunately, in many practical cases, the solution of optimization problems often lie at points of nondifferentiability. There is a very natural extension of differentials to convex functions that play a very similar role, called the subdifferentials, and generalize condition (17).

Subgradients. We say $q \in \mathbb{R}^n$ is a *subgradient* of $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ at $x \in \mathbb{R}^n$ if

$$f(y) \geq f(x) + q^T(y - x).$$

for every $y \in \mathbb{R}^n$. (Note the similarity between this and (17).) The set of all subgradients at x is called the *subdifferential* of f at x, written $\partial f(x)$. There may be many subgradients at some point, though in the case that f is differentiable at x, there will only be one. Similarly to (17) a simple optimality condition is, if $0 \in \partial f(x)$, then x is a minimizer of f, and vice versa.

In general, subgradients need not always exist everywhere, even for convex functions. On the other hand, if $x \in \mathbf{relint}\,\mathbf{dom}\, f$, then it can be shown that the set $\partial f(x)$ is nonempty and bounded. In almost all functions we consider in practice, the function f is indeed everywhere subdifferentiable (has nonempty subdifferential) on its effective domain.

Operations on subdifferentials. If f and g are subdifferentiable at x, then $\partial(f+g)(x) = \partial f(x) + \partial g(x)$. Additionally, if f is defined as the supremum over a family of functions f_y, written:

$$f(x) = \sup_{y \in Y} f_y(x),$$

and this supremum is achieved at some $y^\star \in Y$, then it is not hard to show that a subgradient q of $f_{y^\star}$ at x is a subgradient of f at x; *i.e.*,

$$\partial f_{y^\star}(x) \subseteq \partial f(x).$$

There are a number of other operations, but these are the only ones we will use in this paper.

B Partial converse

We will show that if $p \in \mathbf{int}\, K$ and ℓ contains no line in the direction of p, then there exists some $t > 0$ such that $g(tp) < g(0)$. In other words, if there exists a strict separating hyperplane p between the sets $AX^\star$ and $Y^\star$ (which is true if, and only if, $AX^\star \cap Y^\star = \emptyset$) then $g(0)$ is not optimal and p is a descent direction.

From before, the interior of the cone is defined

$$\mathbf{int}\, K = \{p' \in \mathbb{R}^m \mid p'^T Ax > p'^T y \text{ for all } x \in X^\star,\ y \in Y^\star\}.$$

Now, we will show the contrapositive: if $g(tp) \geq g(0)$ for all $t > 0$, then $p \notin \mathbf{int}\, K$. To see this, let $x_t \in \mathbf{conv}(S)$ and $y_t \in \mathbb{R}^n_+$ be the maximizers for $g(tp)$ (note that such maximizers exist since $\mathbf{conv}(S)$ is a compact set and ℓ is lower semicontinuous and contains no line in the direction of p) such that

$$g(tp) - g(0) = f^*(x_t) - \ell(y_t) + tp^T(y_t - Ax_t) - (\sup f^* - \inf \ell) \geq 0.$$

Rearranging slightly, we find

$$\frac{f^*(x_t) - \sup f^*}{t} + \frac{\inf \ell - \ell(y_t)}{t} + p^T(y_t - Ax_t) \geq 0. \tag{18}$$

Since $f^*(x_t) \leq \sup f^*$ and $\ell(y_t) \geq \inf \ell$ then

$$f^*(x_t) \to \sup f^* \quad \text{and} \quad \ell(y_t) \to \inf \ell,$$

since $p^T y_t$ and $p^T A x_t$ are bounded. (Otherwise, the left hand side of (18) would be unbounded from below.) Additionally, inequality (18) means

$$p^T(y_t - Ax_t) \geq 0,$$

for all t. The lower semicontinuity of f^* and ℓ mean that $x_t \to X^\star$ and $y_t \to Y^\star$, and, since these sequences are bounded, we must have that there exists a subsequence x_{t_k} and y_{t_k} with $x_{t_k} \to x \in X^\star$ and $y_{t_k} \to y \in Y^\star$. This, in turn, means that

$$p^T(y - Ax) \geq 0,$$

so $p \notin \mathbf{int}\, K$.