Scaling On-Chain Asset Exchanges via Arrow-Debreu Exchange Markets*

Geoffrey Ramseyer[1 [0000−0003−3205−2457]], Ashish Goel[2], and David Mazières[3]

[1] geoff.ramseyer@cs.stanford.edu
[2] ashishg@stanford.edu
Stanford University, Stanford, CA 94305, USA

Abstract. Digital currencies present an opportunity to equalize access to financial systems and reduce the cost and latency of trading assets or transferring money. An ideal exchange for these assets would itself be a blockchain; such a system could be decentralized and transparent, among other desirable properties. The primary difficulty in implementing a multi-asset exchange on a blockchain, as opposed to in a traditional centralized setting, is that blockchain nodes must operate deterministically and replicably. In this work, we introduce a new approach to matching requests to trade assets such that all transactions is a block commute with each other. That is to say, transaction execution can be parallelized over an arbitrary number of off-the-shelf CPU cores with minimal synchronization overhead. This approach also introduces independently useful properties, such as removing the need to trade in and out of a reserve currency.

Our approach is to view an asset exchange as an instance of the Arrow-Debreu exchange market, where each asset is valued in an abstract "phantom" currency. The central algorithmic difficulty of this approach is in computing equilibrium prices. For the case where the exchange only supports sell orders, we show how existing convex programming techniques can be adapted to compute equilibrium prices efficiently, in both theory and empirically. We also show how even approximate (or "noisy") prices can be used to find a market-clearing solution with many desirable properties. Finally, we show that supporting both sell and buy orders in these markets makes the price computation a PPAD-hard problem. We believe that our work points to a viable implementation path for sophisticated and robust multi-asset exchanges for digital currencies.

Keywords: Automated Market-Making · Blockchain Scalability · Equilibrium Computation

* Supported by the Stanford Center for Blockchain Research, and by an ARO grant

1 Introduction

Central bank digital currencies come closer to realization every year. However, effective deployment of CBDCs will require a mechanism for efficient CBDC trading.

Of course, individuals have been trading physical currencies for centuries. In order to sell an asset (say A) and acquire another asset (say B) in return, an agent must find a counterparty wishing to sell B in exchange for A, and negotiate a mutually agreeable exchange rate.

Exchanges provide an efficient way of finding counterparties and agreeing upon an exchange rate. Agents can post requests to sell one asset in exchange for another and their minimum requested prices at the exchange, and the exchange can algorithmically match mutually agreeable offers. In a digital asset context, the gold standard for an exchange is a blockchain-based open and transparent exchange. Built into a public blockchain, an ideal exchange could be censorship-resistant, inherently decentralized, fair, and low-cost. Conventional wisdom holds, however, that exchanges built within a blockchain cannot support sufficiently high transaction rates. This wisdom gives rise to a number of alternative scaling techniques, such as off-chain trade matching, automated market-makers, transaction rollup systems, or specialized hardware.

Happily, this conventional wisdom is wrong. The primary difficulty in implementing a multi-asset exchange on a blockchain, as opposed to in a traditional, centralized setting, is that blockchain nodes must operate deterministically and replicably. In this work, we introduce a new approach to matching requests to trade assets such that all transactions is a block commute with each other. This means that transaction execution can be parallelized over an arbitrary number of off-the-shelf CPU cores with minimal synchronization overhead.

Our approach is to view an asset exchange as an instance of an Arrow-Debreu exchange market. Instead of individually matching buyers with sellers, the approach views the entire set of open trade offers as one Arrow-Debreu market, and computes an equilibrium of this market. In essence, this means every asset gets a valuation (in an abstract or "phantom" currency, not in a real reserve currency) and assets trade at these valuations. Whether or not a request to trade executes depends only on the parameters of the request and the equilibrium prices, and thus each request can be processed independently.

Formulating an asset exchange in this manner induces some additional useful properties. There are automatically no arbitrage opportunities within the system; that is to say, there is no difference between a request to sell A and buy B, and a request to sell A to buy C that is immediately followed by a request to sell C to buy B. Furthermore, in real foreign exchange markets, some pairs of currencies have low direct liquidity. For example, many traders have to buy and then immediately sell USD as an intermediate good when exchanging one less-frequently traded currency for another. An exchange modeled on an Arrow-Debreu market gives users no need to hold intermediate goods or use a reserve currency.

The central algorithmic difficulty of this approach is in computing equilibrium prices. A number of algorithms exist in the theoretical literature for special cases of Arrow-Debreu markets. In this discussion, we primarily consider the case where an agent can put some units of an asset up for sale, and if the agent is satisfied by the equilibrium prices, it sells as many of these units as they can in order to buy its desired asset. Prices can be computed via a convex program, following [9], or via several distinct algorithmic approaches, such as an iterative Tâtonnement process [8]. However, directly applied, the algorithms of the literature scale pooly as the number of open trade offers increases, both theoretically and empirically. We show that our setting has additional structure that gives a simpler convex programming formulation and enables modification to some of the algorithms to make them efficient in practice, both empirically and asymptotically. Finally, some exchanges allow users the flexibility to sell only as much of an asset as is needed to buy a fixed amount of a target asset; we remark that introducing this flexibility makes equilibrium computation PPAD-complete.

In section 2, we outline the details of the types of offers in our exchange and how they lead to an Arrow-Debreu market. In section 3 we summarize some challenges implementing price computation that arise from using the Arrow-Debreu market model as a real-world exchange, and in section 4 we give an efficient algorithm for computing market equilibria and an empirical measure of its performance.

Related Work: Automated, decentralized trading is an active area of research, and a number of solutions are currently deployed on public blockchains. Some protocols, like Uniswap[3] and Bancor[14], provide smart contracts that act as automated market-makers, typically trading between two assets. Based on the contract's currency reserves, they calculate a price, and offer this price to everyone[4]. The 0x platform[16] provides an API for securing atomic asset transfers between untrusting parties, nominally as a building block for other distributed exchanges. StarkEx [1,6] provides cryptographic primitives that enable a centralized exchange to perform trades off of a blockchain but prove the validity of these trades publicly. The Stellar[2] blockchain provides a native trade request mechanism with a public orderbook, and matches offers when possible as part of its transaction processing logic.

The literature on equilibria computation in Arrow-Debreu exchange markets is quite broad. We do not give a complete survey here, but note that when agent utilities are linear (as we will describe later), the abstract market

can be solved by a variety of methods, including convex programs and other optimization-related methods [9,17], combinatorial "balanced-flow"-based algorithms [15,12,13], and iterative "tâtonnement" methods [8]

2 Our Approach

The typical role of an asset exchange is to help traders find agreeable counterparties. In a traditional asset exchange, one agent might publicly post a trade offer, which another agent could then select. The two can then trade goods according to the posted offer. Most large-scale exchanges facilitate this matching by automated processes.

One difficulty with this approach is that each pair of offers can trade assets at slightly different prices. This means that the resulting state of the ledger depends on exactly how orders are matched. Hence, the operation of identifying a matching pair of offers from the set of unmatched offers and executing a trade between the two (removing these offers from the unmatched set) is not a commutative operation.

This noncommutativity is not a problem for a centralized exchange, which can nondeterministically output any valid matching between offers (and thus parallelize the offer matching logic). However, a blockchain's state updates must be replicable, and thus deterministic. Thus, offers must be matched in some deterministic order, and the offer matching logic must be serialized.[3]

Example 1. Consider an exchange where two individuals A_1 and A_2 each offer 1 USD for sale, A_1 demands 1 Euro in exchange but A_2 demands 2 Euros. Suppose that two other individuals B_1 and B_2 each wish to buy one USD, and both are willing to pay 2 Euros. Without additional situational context, there is no reason for the system to prefer the matching $\{(A_1, B_1), (A_2, B_2)\}$ over the matching $\{(A_1, B_2), (A_2, B_1)\}$. However, depending on the market's choice of matching, one of the B_is will have to pay 2 Euros to get a USD, while the other will only have to pay 1.

2.1 Approach Details

Instead of individually matching offers in some sequence, our approach is to add the entirety of each block of new transactions to the set of preexisting open offers, look at the entirety of this pool of open offers, and execute as many trades as possible, with all trades executing at the same prices.

There are many types of offers used in real-world exchanges, but each type of offer is ultimately an instruction to trade some set of assets for some other set of assets, subject to specified conditions.

For example, a request to sell an asset might look like the following:

Definition 1 (Sell Offer). *A Sell Offer (S, B, e, α) is request to sell e units of good S in exchange for some number k units of good B, subject to the condition that $k \geq \alpha e$.*

.

A request to buy a fixed number of units of some asset might look like the following:

Definition 2 (Buy Offer). *A Buy Offer (S, B, k, α) is request to purchase k units of good B in exchange for selling some number e units of good S, subject to the condition that $e \leq k/\alpha$.*

The critical observation is that this type of system is representable as an Arrow-Debreu Exchange Market [5].[4].

Definition 3 (Exchange Market). *Suppose that N distinct types of divisible goods are traded on a market.*

Every agent i possesses some endowment $\{e_j\}_{j\in[N]} \in \mathbb{R}^N_{\geq 0}$ of assets and some utility function $u_i(\cdot)$ expressing preferences on bundles of goods.

We denote a bundle of goods x as a vector in $\mathbb{R}^N_{\geq 0}$, where the jth component refers to an amount of good j.

The market is equipped with prices $\{p_j\}$ for each asset.

Each agent i sells all of its assets at the market prices, receiving $s_i = \Sigma_j p_j e_j$ units of "money". It then immediately uses this "money" to buy back its preferred bundle of goods $x_i = \arg\max_{x:\Sigma x_j p_j \leq s_i} u_i(x)$.

[3] There may be opportunities for improvements over naive serialization. However, when matching buy offers to sell offers, each offer will have potential noncommutative interactions with many other offers. Under reasonable transaction semantics, like e.g. those in the Stellar blockchain, those on traditional exchanges, or automated market makers like Uniswap, noncommutativity between two transactions is the norm, not the exception.

[4] The term "Arrow-Debreu Exchange Market" can mean different things in different contexts. We mean here a market where there are no stock dividends and no production. Also, while of course traders might have time-varying strategies in the real-world, our system will look at snapshots of the market, frozen in time. Hence, for this discussion, we will consider only the single-round setting of the Arrow-Debreu market.

The agents have no utility for the intermediate "money"; in real-world terms, currencies like the dollar or euro are treated like any other traded asset.

Traders typically do not submit utility functions to exchanges. However, most types of natural trade requests are expressible using simple utility functions. It is not hard to see that bundles that maximize the following utility functions satisfy the requirements of buy and sell offers.

Example 2 (Sell Offer).

Suppose an agent wishes to sell Euros in exchange for USD, and would like at least α USD per Euro.

Implicitly, this agent values one Euro as equal to α USD, and its utility function for a bundle of goods x is $u(x) = \alpha x_{Euro} + x_{USD}$.

Example 3 (Buy Offer).

Suppose an agent wishes to buy exactly k USD in exchange for Euros, and would like to spend at most $1/\alpha$ Euro per USD.

Implicitly, this agent values one Euro as equal to α USD and has no utility for USD after the first k. Its utility function is therefore $u(x) = \alpha x_{Euro} + \min(x_{USD}, k)$.

Arrow and Debreu show that under certain technical conditions, there exist equilibria in this market; that is, there exist prices for each good j, and bundles of goods x_i for each agent i such that x_i maximizes agent i's utility (subject to i's budget constraint) and for each asset, the amount sold to the market equals the amount bought from the market[5].

Our approach will be to compute market equilibria. Before proceeding, suppose for a moment that we have reliable access to an exact equilibrium price oracle. Executing all orders at equilibrium prices has a number of desirable properties. Most importantly, this property makes transaction processing commutative. For all but a small number of offers, an agent's optimal bundle is unique and easily computable given equilibrium prices; hence, the results of each transaction can be computed in parallel.

Example 4. Consider an exchange where two individuals A_1 and A_2 each offer 1 USD for sale, A_1 demands 1 Euro in exchange but A_2 demands 2 Euros. Suppose that two other individuals B_1 and B_2 each wish to buy one USD, and both are willing to pay 2 Euros.

The prices $2 = 2 * p_{Euro} = p_{USD}$ are equilibrium prices. Each A_i loses 1 USD and receives 2 Euros, while each B_i loses 2 Euros and receives 1 USD.

2.2 Other Useful Properties

Constructing an exchange using an Arrow-Debreu market gives a number of other desirable properties.

- No (Internal) Arbitrage
  Given equilibrium prices, there is no benefit to be gained from exchanging assets through intermediate assets. If one agent sells one unit of asset A in exchange for as much asset B as possible, they will receive exactly p_A/p_B units of B. They could sell A in exchange for p_A/p_C units of C and then immediately then exchange their C for B, but this would still earn the agent $p_A/p_B = (p_A/p_C)(p_C/p_B)$ units of B.
  This is not to say that there is no arbitrage between our market and external markets, but if an agent possesses asset A and wishes to ultimately acquire asset B, there is no incentive for the agent to do anything other than to directly trade A in exchange for B, and vice versa.
- Increased Liquidity in Sparse Markets
  If orders are matched explicitly, then a market between two assets is liquid if and only if there are many agents trading directly between the two assets. Otherwise, a trader might have to deal in intermediate assets. In a traditional currency exchange, one user who ultimately wishes to trade Canadian currency for Australian currency might find it easier to trade Canadian dollars for US dollars and then trade US dollars for Australian dollars.
  However, in our setting, trading directly at market-equilibrium prices is equivalent to trading through intermediate assets (at market-equilibrium prices). This should provide liquidity between two assets even if the direct orderbook is empty, so long as their is a sequence of assets $a_1, ..., a_k$ such that each pair a_{i-1}, a_i is a liquid market.

  Example 5. Suppose there are 3 assets A, B, C, and there are three traders. The first wishes to sell one A for at least one B, the second wishes to sell one B for at least one C, and the third wishes to sell one C for at least one A. In a traditional market, no two trades of these trades could be matched pairwise. However, in our setting, the market can compute equilibrium prices (in this case, all assets have the same valuation), and all trades fully execute, with no agent ever holding an intermediate asset.

[5] The Arrow-Debreu market as outlined in [5] is a very general model, and as such one can construct agents and preferences such that equilibrium prices do not exist; however, with the agents and preferences discussed here, such scenarios either do not occur or can be easily identified and accounted for. We might simply check e.g. property (*) of [9].

- Less Dependence on Transaction Ordering
  In a blockchain system, transactions are grouped into blocks for consensus and processing. Making transactions commutative means that within one block, the ordering of transactions cannot affect transaction results.
  As such, every agent that can send a message to a block producer before some block production cutoff time will see their orders execute at the same prices.
  One potential method for exploiting a low-latency network connection in market that explicitly matches trades is to spy on network traffic to see incoming requests to purchase an asset. The spy can then send purchase requests ahead of the slower requests to buy the asset in question, then immediately resell the asset at the slightly higher price offered by the slower transaction.
  This type of attack is harder to pull off if the slower transaction can still arrive before a block cutoff time.
  Of course, one agent's trading behavior might inform the behavior of another agent in the longer-term, so low-latency communication with block producers or visibility into network traffic might still be valuable over longer periods of time.

3 Price Computation: Theory vs. Practice

Price computations is easiest in markets where all transactions are sell offers (as per Definition 1), as this corresponds to an Arrow-Debreu market where all utilities are linear (and sparse).

Unfortunately, much of the literature assumes that there is a one to one mapping between agents and goods. While we would like our market to support trading as many different types of assets as possible, more important than number of assets is the number of real-world agents who can participate. One real-world individual might wish to submit many orders at the same time.

Instead, we would like an algorithm whose runtime depends as little as possible on the size of the orderbook. Many of the algorithms for equilibria computation are iterative in nature or somehow repetitive; each iteration should be ideally be parallelizable and depend as little as possible on the size of the orderbook.

We will allow our equilibria computation algorithms to be approximate. Informally speaking, our market will be allowed to charge a small commission on transactions, and agents whose requested minimum prices are close to market prices – that is, agents who are close to indifferent, in terms of their utility functions, between selling or not selling – might have their trades only partially execute.

Several subtly varying notions of approximate equilibria appear in the literature (e.g. [15,8]); however, there are subtle differences between some definitions in the literature and the second type of approximation here. For formal details on the types of approximation error we allow, see Appendix D.

Our exchange also faces some requirements not typically present in the theoretical literature on equilibrium computation.

- Indivisible Goods
  Every asset has some smallest indivisible unit.
- Minimize Adverse Behavior
  An agent should not be able to earn money by engaging in socially harmful actions. For example, an agent who wishes to sell 100 units of an asset at the same price should be incentivized to submit only one request to sell 100 units, or at least not incentivized to submit 100 identical requests to each sell one unit.
- Fair Rounding
  Because goods are indivisible, there will be some trades and asset payments that will have to be rounded. However, an agent's losses due to rounding should be minimized. In particular, an agent should only lose at most one indivisible unit of a good to rounding.
- Efficent Verification
  The correctness of the exchange's operation, including computing prices and processing transactions, should be efficient to verify. When a replica comes online after being temporarily offline for a time, it needs to download and replay the exchange's state changes that occurred during its downtime. In order to sychronize with other replicas, the newly online replica must be able to process and verify these state changes faster than the rest of the system generates new state updates.

4 Algorithmic Results

4.1 Tâtonnement

The Tâtonnement algorithm of [8] satisfies both of these requirements. Starting at some arbitrary set of prices, the algorithm repeatedly computes excess supply and demand at those prices and raises the prices of overdemanded goods.

In a naive implementation of Tâtonnement, each excess demand computation would require iterating over every open offer. But sell orders naturally have an ordering induced by their desired prices; an order that requests a lower price will be satisfied if an order that demands a higher price finds the current prices satisfactory. As such, with some preprocessing, the excess demand queries used in Tâtonnement are computable via a series of binary searches (one search for every pair of assets being bought and sold).

Theorem 1. *In a market with M open offers trading N assets, an excess demand query takes time $O(N^2 \lg M)$.*

These searches can also run in parallel.

The last hidden difficulty in implementing Tâtonnement is that the exact total demand of a set of orders is not a continuous function of query prices. This makes it possible for a naive implementation to oscillate between two price vectors. We modify offer behavior to make the excess demand function continuous function by declaring that offers whose desired prices are close to the prices offered by the query execute only partially.[6]

Finally, we can refine the output of the above computation to identify any remaining possible trades, and minimize the total number of trades that execute partially (and thus minimize rounding error) at the given prices by solving a simple linear program. For details, see Appendix C.

4.2 Convex Programming

The equilibria of the convex program of [9] are exactly the equilibria of the corresponding Arrow-Debreu market with linear utilities.

Suppose there are N assets and M agents. Each agent i posesses e_i units of good a_i and would like to purchase good b_i, if the price of good a_i is at least u_i times the price of b_i (equivalently, agent i values one a_i as equal to u_i units of b_i).

The program has the following variables:

- $\{p_i\}_{i\in[N]}$ represents the price of good i.
- $\{y_i\}_{i\in[M]}$ represents the amount of money that agent i spends to purchase b_i (equivalently, how much money agent i earns by selling good a_i).
- $\{\beta_i\}_{i\in[M]}$ represents the inverse max bang-per-buck of agent i.

The program is as follows:

$$\min \Sigma_{i\in[M]} p_{a_i} e_i \lg(p_{a_i}/\beta_i) - y_i \lg u_i \tag{1}$$

$$s.t.\ 0 \le y_i \le p_{a_i} e_i \qquad \forall i \in [M] \tag{2}$$

$$\Sigma_{i\in[M]:b_i=j} y_i = \Sigma_{i\in[M]:a_i=j} y_i \qquad \forall j \in [N] \tag{3}$$

$$\beta_i \le p_{a_i} \qquad \forall i \in [M] \tag{4}$$

$$u_i \beta_i \le p_{b_i} \qquad \forall i \in [M] \tag{5}$$

$$\beta_i \ge 0 \qquad \forall i \in [M] \tag{6}$$

$$p_i \ge 1 \qquad \forall i \in [N] \tag{7}$$

Theorem 2. *The above convex program can be transformed to one whose objective is computable in time $O(N^2 \lg(M))$ using $O(N^2)$ variables.*

The catch is that this transformation makes the objective non-differentiable, which experimentally appears to cause trouble for general-purpose convex optimizers. For details, see Appendix A.

4.3 Solution Refinement via a Linear Program

The approximation errors of the above approaches might produce equilibria where a small number of agents might not trade, even if they both accept the computed set of prices. These situations can be eliminated by solving a simple linear program, outlined in Appendix C.

Theorem 3. *Suppose there are N assets being traded on an exchange market.*

The linear program of Appendix C has size $O(N^2)$ and transforms an approximate market equilibrium into an approximate equilibrium where at most N^2 offers execute partially. All other offers either execute fully or not at all.

[6] Any reasonable rule will do to determine what fraction of an order executes; we use a linear interpolation. Concretely, we say that an order selling USD for Euros and desiring at least 1.0 Euros per USD does not execute if offered fewer than 1.0 Euros/USD, executes fully if offered a price of at least $1 + \mu$ Euros/USD, and has an x/μ fraction execute if offered $1 + x \le 1 + \mu$ Euros/USD.

Proof. The size of the linear program follows by construction. The linear program computes, for every asset pair (A, B), an amount y_{AB} representing how much of asset A is sold to purchase asset B. We can sort offers to sell A for B by minimum acceptable price into some linear order. There is some minimal initial segment of offers with more than y_{AB} to sell, and only the last offer (the one whose minimum acceptable price is closest to the market price) of this initial segment is fractionally executed.

Minimizing the number of orders that fractionally execute minimizes the number of agents who lose fractional amounts of assets due to rounding.

4.4 Empirical Comparison

Figure 4.4 charts the average runtimes an implementation of Tâtonnement and Convex Program 1, which was run using the off-the-shelf solver ECOS [11] via CVXPY [10]. Both were run single-threaded on the first author's laptop. Tâtonnement was run with a 2^{-20} (roughly 1/1000000) transaction commission and required offers with minimum prices more than a $1+2^{-7}$ (roughly 1.01) factor larger than market prices to execute fully. ECOS was run with error parameters set to 0.001. Agents traded between 20 distinct assets.

Trade offers were generated from the same model. The simulator first generates underlying valuations for each asset. Every offer then picks a uniformly random buy and sell asset, and requests to sell a random amount with minimum price equal to the ratio of the underlying valuations plus a small amount of gaussian noise.

We set the error parameters of the convex program with the intent that they be less strict than those given to Tâtonnement. That said, the types of error that Tâtonnement gives are different from those that a convex solver give. As such, these raw numbers are somewhat incomparable. These numbers measure just algorithm runtimes, not including preprocessing transactions, managing account records, checking transaction signatures, or anything else related to running a real-world exchange. The trendlines are the important objects of consideration.

There are three important points to highlight from the graph:

- Tâtonnement performs extremely well with higher numbers of offers, with runtimes that appear feasible in practice.
- The performance of off the shelf convex solvers is very good for small numbers of offers, but runtime scales linearly or slighly superlinearly with number of offers.
- At very small numbers of transactions, Tâtonnement performs unreliably.

This would suggest that Tâtonnement-based approaches could work well when many traders are actively trading, but might need to fall back to a different solver if very few traders are active. Note that Tâtonnement appears to become reasonably reliable once more than $2-3$ offers are present for every pair of assets, on average.

Tâtonnement also benefits from starting at a point close to an equilibrium. In these tests, it started at the all-ones vector, but in practice it could start at the previous block's clearing prices.

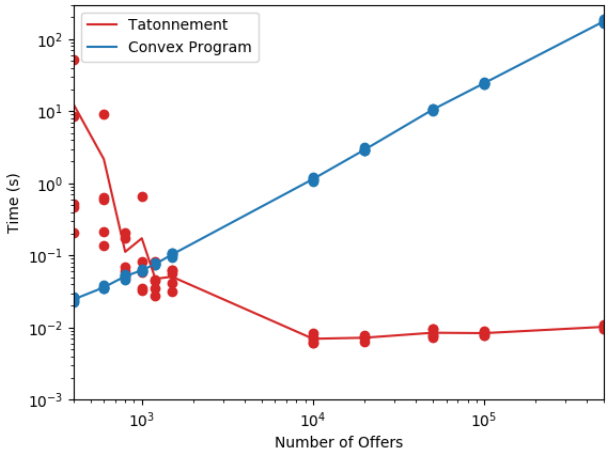

Fig. 1. Average Runtimes of Tâtonnement and the Convex Program of Section 4.2

5 Buy and Sell Offers Together are PPAD-Complete

Equilibrium computation in markets with linear utilities is solvable in polynomial time, and appears tractable in practice. However, integrating buy offers moves equilibrium computation to a much harder complexity class.

Theorem 4. *Computing approximate equilibrium prices in an Arrow-Debreu market generated by Sell Offers and Buy Offers is PPAD-complete.*

Proof. Follows from Corollary 2.4 of [7]. Details are in Appendix B.

It remains an open question whether including buy offers is empirically more difficult than just sell offers.

References

1. Starkex, `https://starkware.co/product/starkex/`
2. Stellar, `https://www.stellar.org/`
3. Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 core (2020)
4. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of uniswap markets. Cryptoeconomic Systems Journal (2019)
5. Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica: Journal of the Econometric Society pp. 265–290 (1954)
6. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. (2018)
7. Chen, X., Paparas, D., Yannakakis, M.: The complexity of non-monotone markets. Journal of the ACM (JACM) **64**(3), 1–56 (2017)
8. Codenotti, B., Pemmaraju, S.V., Varadarajan, K.R.: On the polynomial time computation of equilibria for certain exchange economies
9. Devanur, N.R., Garg, J., Végh, L.A.: A rational convex program for linear arrow-debreu markets. ACM Transactions on Economics and Computation (TEAC) **5**(1), 1–13 (2016)
10. Diamond, S., Boyd, S.: Cvxpy: A python-embedded modeling language for convex optimization. The Journal of Machine Learning Research **17**(1), 2909–2913 (2016)
11. Domahidi, A., Chu, E., Boyd, S.: Ecos: An socp solver for embedded systems. In: 2013 European Control Conference (ECC). pp. 3071–3076. IEEE (2013)
12. Duan, R., Mehlhorn, K.: A combinatorial polynomial algorithm for the linear arrow–debreu market. Information and Computation **243**, 112–132 (2015)
13. Garg, J., Végh, L.A.: A strongly polynomial algorithm for linear exchange markets. In: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing. pp. 54–65 (2019)
14. Hertzog, E., Benartzi, G., Benartzi, G.: Bancor protocol (2018)
15. Jain, K., Mahdian, M., Saberi, A.: Approximating market equilibria. In: Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques, pp. 98–108. Springer (2003)
16. Warren, W., Bandeali, A.: 0x: An open protocol for decentralized exchange on the ethereum blockchain (2017)
17. Ye, Y.: A path to the arrow–debreu competitive market equilibrium. Mathematical Programming **111**(1-2), 315–348 (2008)

A Convex Formulation

Repeated below is the convex program of [9] (slightly simplified).

Suppose there are N assets and M agents. Each agent i posesses e_i units of good a_i and would like to purchase good b_i, if the price of good a_i is at least u_i times the price of b_i (equivalently, agent i values one a_i as equal to u_i units of b_i).

The program has the following variables:

1. $\{p_i\}_{i\in[N]}$ represents the price of good i.
2. $\{y_i\}_{i\in[M]}$ represents the amount of money that agent i spends to purchase b_i (equivalently, how much money agent i earns by selling good a_i).
3. $\{\beta_i\}_{i\in[M]}$ represents the inverse max bang-per-buck of agent i.

With these variables, the convex program is:

$$\min \Sigma_{i\in[M]} p_{a_i} e_i \lg(p_{a_i}/\beta_i) - y_i \lg u_i \tag{8}$$

$$s.t.\ 0 \le y_i \le p_{a_i} e_i \qquad \forall i \in [M] \tag{9}$$

$$\Sigma_{i\in[M]:b_i=j} y_i = \Sigma_{i\in[M]:a_i=j} y_i \qquad \forall j \in [N] \tag{10}$$

$$\beta_i \le p_{a_i} \qquad \forall i \in [M] \tag{11}$$

$$u_i \beta_i \le p_{b_i} \qquad \forall i \in [M] \tag{12}$$

$$\beta_i \ge 0 \qquad \forall i \in [M] \tag{13}$$

$$p_i \ge 1 \qquad \forall i \in [N] \tag{14}$$

Of course, the number of variables in this program scales linearly with the number of outstanding offers. But the optimal value of β_i is easy to compute, and in fact, if we replace the variable β_i with the function $\beta_i(p) = \max(p_{a_i}, p_{b_i}/u_i)$, the objective function becomes:

$$\min \Sigma_{i\in[M]} p_{a_i} e_i \lg(p_{a_i}/\beta_i(p)) - y_i \lg u_i \tag{15}$$

This function is convex, and in fact, is equivalent to:

$$\min \Sigma_{i\in[M]:p_{b_i}/p_{a_i}>u_i} p_{a_i} e_i \lg(p_{a_i}/p_{b_i}) + p_{a_i} e_i \lg u_i - y_i \lg u_i \tag{16}$$

We can further simplify this objective. Rather than including one variable y_i per each agent, we can include one variable z_{ab} representing the spending choices of all agents. Any optimal solution for the original program will correspond to an optimal solution of the modified program via the equation $z_{ab} = \Sigma_{i:a_i=a,b_i=b} y_i$.

The last step is to simplify the term $\Sigma_i y_i \lg u_i$.

Let $\alpha_{ab,k}$ be the index of the agent selling a and buying b with the kth highest u_i.

Let $f_{ab}(e)$ denote the continuous piecewise linear function that starts at 0, has slope $\lg u_{\alpha_{ab,1}}$ for the first segment of length $e_{\alpha_{ab,1}}$, has slope $\lg u_{\alpha_{ab,2}}$ for the first segment of length $e_{\alpha_{ab,2}}$, and so on.

Given some value $z_{ab} = \Sigma_{i:a_i=a,b_i=b} y_i$, the configuration of the y_is that satisfies the constraints of the convex program and minimizes the objective corresponds to the allocation that lets the agent that places the greatest value on b buy b first. When that agent's budget is exhausted, the agent with the second highest value gets to buy b, and so on, until z_{ab} is exhausted.

Under this configuration of y_is, the term $\Sigma_{i:a_i=a,b_i=b} y_i \lg u_i = p_a f_{ab}(z_{ab}/p_a)$.

Finally, then, a simplified convex program is as follows:

$$\min \Sigma_{i\in[M]:p_{b_i}/p_{a_i}>u_i} p_{a_i} e_i \lg(p_{a_i}/p_{b_i}) + p_{a_i} e_i \lg u_i - p_a f_{ab}(z_{ab}/p_a) \tag{17}$$

$$s.t.\ 0 \le z_{ab} \le p_{a_i} \Sigma_{i:a_i=a,b_i=b} e_i \qquad \forall (a,b) \in [N]\times[N] \tag{18}$$

$$\Sigma_{b\in[N]} z_{ab} = \Sigma_{b\in[N]} z_{ba} \qquad \forall a \in [N] \tag{19}$$

$$p_i \ge 1 \qquad \forall i \in [N] \tag{20}$$

By sorting orders by u_i and some minimal additional preprocessing, this objective function can be evaluated by a series of binary searches, as in the case of the Tâtonnement algorithm.

Unfortunately, this objective is no longer differentiable. It is possible to integrate into this function a notion of partial execution of offers, similar to what was used in the Tâtonnement algorithm. Making the objective twice differentiable everywhere and still efficiently evaluatable while respecting agent preferences appears challenging but not necessarily impossible.

B Proof of Theorem 4

Proof. The theorem is a restatement of Corollary 2.4 of [7].

Corollary 2.4 follows from Theorem 7 of [7], and requires agents with general linear utility functions, not just the sparse ones generated by Sell Offers. However, the construction in Theorem 7 is quite general; when applied to the context of agents with utilities generated by Buy or Sell Offers, this requirement is not necessary.

In fact, the construction in this context uses only utilities that could be generated by Sell or Buy Offers.

One technical detail is that some agents have utility only for one good, and might have multiple types of goods in its endowment. An agent with linear utility and multiple types of goods in its endowment can be replaced by multiple agents with the same utility, each one with only one type of good, to the same overall effect on the market. An agent with utility for only one type of good is akin to a Sell offer with minimum price 0.

C Linear Program for Rounding

The output of an (ε,μ)-approximate price oracle (for a formal definition of (ε,μ)-approximate, see Appendix D) is a set of prices p_i for asset $i \in [N]$.

Denote by $E_{ab}(p)$ the total number of units of a endowed to agents who wish to purchase b and whose only optimal bundle is to sell their entire endowment of a. That is to say, the agents minimum price is greater than $(p_a/p_b)/(1-\mu)$.

Denote by $F_{ab}(p)$ the total number of units of a endowed to agents who wish to purchase b and who are willing to accept prices p. That is to say, the agents minimum price is greater than (p_a/p_b).

Note that given the output of a price oracle, the quantities p, $E_{ab}(p)$, and $F_{ab}(p)$ are all constants.

The question remaining is to maximize the amount of assets trading hands, subject to the constraint that for each pair of assets (a, b), the amount of a sold in direct exchange for b (which we denote y_{ab}) must be between $E_{ab}(p)$ and $F_{ab}(p)$, and no assets can be created or destroyed.

This notion corresponds to the following linear program.

$$\max \ \Sigma_{a,b\in[N]} p_a y_{ab} \tag{21}$$

$$s.t. \ E_{ab}(p) \leq y_{ab} \leq F_{ab}(p) \qquad \forall (a,b) \in [N] \times [N] \tag{22}$$

$$(1+\varepsilon) p_a \Sigma_{b\in[N]} y_{ab} \geq \Sigma_{b\in[N]} p_b y_{ba} \qquad \forall a \in [N] \tag{23}$$

$$\tag{24}$$

This linear program has size $O(N^2)$.

D Types of Approximation Error

1. Transaction Commission (ε)

   Definition 4 (Transaction Commission).
   The market can charge every transaction a ε fee.

   Equivalently, agents sell an asset A for valuation p_A, but must purchase A at price $p_A(1+\varepsilon)$.
2. Approximately Optimal Bundles (μ)
   TODO make this more clear Start with what is this approximation trying to achieve
   Our introductory statement about the existence of equilibrium prices was not quite correct. A fully specified market equilibrium includes not only prices but also an allocation of goods. At an exact equilibrium, every agent purchases a bundle of goods that maximizes their utility function, subject to their budget constraint. In many cases, there is a unique bundle of goods which an agent strictly prefers over all others (again subject to their budget constraint), so this distinction is trivial; but an agent might be indifferent between multiple bundles of goods. In such a case, a fully specified market equilibrium must specify which bundle an agent receives.
   Concretely, if an agent offers to sell one unit of good A for at least α units of good B, the agent is indifferent to not selling its A, receiving α units of B, or any linear combination thereof, if and only if $p_A/p_B = \alpha$.
   We will allow the market to give agents slightly suboptimal bundles, subject to the following conditions.
   (a) Agents do not receive goods for which they have 0 marginal utility. [7].
   (b) Agents spend their entire budget.
   Note that these requirements differ from a typical definition of approximately optimal agent behavior, as in e.g. [15,8].

   Definition 5 (μ-Approximately Optimal Bundle). *Suppose an agent has utility $u(\cdot)$ and initial bundle x, and the market prices are p. Let x^* maximize $u(\cdot)$ subject to $x^* \cdot p \leq x \cdot p$.*
   A bundle y is μ-approximate if $u(y) \geq u(x^*)(1-\mu)$, $y = x \cdot p$, and y does not include goods for which the agent has 0 marginal utility.

   This approximation notion is not an additional transaction commission. It should be thought of as authorizing the platform to partially execute orders (at current market prices) from agents who are close to indifferent to the current market prices.

   Example 6. Suppose that an agent offers to sell one unit of good A for at least α units of B.
   Suppose first $p_A/p_B \geq \alpha \geq (p_A/p_B)(1-\mu)$. Let y_β be the bundle formed by selling β units of A to receive $\beta p_A/p_B$ units of B. It is easy to see that y_β exhausts the full budget of the agent, that $u(y_\beta) \geq u(x)$, and that $u(y_\beta) \geq u(x^*)(1-\mu)$. Hence, y_β is a μ-approximately optimal bundle.
   Now suppose instead that $p_A/p_B < \alpha$. In this case, the only approximately optimal bundle for the agent is to keep all of its A. Alternatively, if $(p_A/p_B)(1-\mu) > \alpha$, then the only approximately optimal bundle for the agent is to exchange its entire stock of A for p_A/p_B units of B.

[7] For example, a Buy Offer requesting c units of a good might purchase up to but no more than c units, and a Sell Offer will not receive any goods in which it has no interest