Problem given to me by @barryWhiteHat.

Problem Definition

In human language: we are running an exchange with uniform price auctions (https://en.wikipedia.org/wiki/Multiunit_auction#Uniform_price_auction) for transactions and atomic swaps. We want to see how hard it is for a block proposer to maximize his/her gain when deciding which swaps to include. In eth1.x (First and second-price auctions and improved transaction-fee markets) we only accept a single fee token (eth); this problem seems harder.

In math language

- let's call the problem MAX_FEES_ATOMIC_SWAP
- we take the POV of a block proposer on an exchange.
- the messages on the exchange all take one of two forms:
- transactions

: (TOKEN, FEE): Alice is sending Bob some transaction in TOKEN, of which FEE is offered to the proposer who picks it up.

- atomic swaps:

(TOKEN1, FEE1, TOKEN2, FEE2): Alice and Bob trade some TOKEN1 exchanged with TOKEN2, of which FEE1 is offered "for" TOKEN1 and FEE2 is offered "for" TOKEN2.

- "obvious" things like sender, receiver, and volume are all part of the data, but are irrelevant

for us, the block proposer, since we only care about fees.

- transactions

: (TOKEN, FEE): Alice is sending Bob some transaction in TOKEN, of which FEE is offered to the proposer who picks it up.

- atomic swaps:

(TOKEN1, FEE1, TOKEN2, FEE2): Alice and Bob trade some TOKEN1 exchanged with TOKEN2, of which FEE1 is offered "for" TOKEN1 and FEE2 is offered "for" TOKEN2.

- "obvious" things like sender, receiver, and volume are all part of the data, but are irrelevant

for us, the block proposer, since we only care about fees.

- the game is that the block proposer can take as many swaps as possible in 1 block, but the fee for each transaction for a token is the MIN of all the fees for the token picked up in a block. (so for example, if I pick up an atomic swap for ETH and 0.5 fee and a transaction for ETH offering 0.2 fee, we pick up 0.2 from each)
- observation: The problem is equivalent to one where there is only

atomic swaps, since I can think of any transaction (TOKEN, FEE) as an atomic swap (TOKEN, FEE, NEW-TOKEN, 0) where NEW-TOKEN is a throwaway padding token added to the model.

Problem:

what's the complexity of how to find the subset of atomic swaps to maximize fees collected?

Proof of NP-Hardness

Claim 1

: MAX_INDEP_SET is NP-hard

- Definition of MAX_INDEP_SET given a graph, we want to find the maximum number of vertices that are independent (have no edges to each other)
- Proof of hardness: (https://en.wikipedia.org/wiki/Independent_set_(graph_theory)). It also seems hard to approxmiate

Claim 2

: if you can solve MAX_FEES_ATOMIC_SWAP as oracle, you can solve MAX_INDEP_SET in polynomial time

- Given a graph G

with n

vertices, make the following tokens: * E(X, Y)

; 2 for every edge (X,Y)

, one in each direction

- E(X, Y)

; 2 for every edge (X,Y)

, one in each direction

- pick a big number M such that 1 << M
- for each edge e = (x,y)

, make 2 transactions of the following form * (E(X,Y), 1)

; [technically as atomic swaps, so (E(X,Y), 1, TEMP, 0)

]

- (E(Y,X), 1)

;

call these the “edge transactions”

- (E(X,Y), 1)

; [technically as atomic swaps, so (E(X,Y), 1, TEMP, 0)

]

- (E(Y,X), 1)

;

call these the “edge transactions”

- now, we will list some logical statements of the form “P NAND Q”, where each P, Q is some E(X,Y)

. For each such statement R, we will make a set of swaps that we call the “NAND swaps for R” as follows: * (LP, M, NOTLP, 0)

[semi-important caveat; if P appears in 2 such logical statements, we make a new LP for each such statement]

- (LP, 0, NOTLP, M)
- (LP, M, P, 0)
- (NOTLP, M, Q, 0)
- observe that local to these 4 statements, the max you can possibly get is 2M, and that forces P or Q or both to be 0 (assuming M is huge). That’s why we call these the NAND swaps
- the set of logical statements will make NAND swaps are:

"E(X,Y)

NAND E(Y,X)

" for all edges e = (X,Y)

"E(X,Y)

NAND E(X,Z)

" for all edges (X,Y)

and (X,Z)

incident at X.

- call this set of logical statements L

. |L| = (2|E| + \sum_{x \in V} {deg(x) \choose 2})

, and we have 4L

“NAND swaps”

- (LP, M, NOTLP, 0)

[semi-important caveat; if P appears in 2 such logical statements, we make a new LP for each such statement]

- (LP, 0, NOTLP, M)
- (LP, M, P, 0)
- (NOTLP, M, Q, 0)
- observe that local to these 4 statements, the max you can possibly get is 2M, and that forces P or Q or both to be 0 (assuming M is huge). That’s why we call these the NAND swaps
- the set of logical statements will make NAND swaps are:

"E(X,Y)

NAND E(Y,X)

" for all edges e = (X,Y)

"E(X,Y)

NAND E(X,Z)

" for all edges (X,Y)

and (X,Z)

incident at X.

- call this set of logical statements L

. |L| = (2|E| + \sum_{x \in V} {deg(x) \choose 2})

, and we have 4L

“NAND swaps”

Suppose we can solve MAX_FEES_ATOMIC_SWAPS. We now show solving it for this setup solves MAX_INDEP_SET.

Lemma

: in the maximizing configuration, for each logical statement l \in L

, at least one of the 2 E(X,Y)

's will be set to 0

fee.

- each of the 4L logical statements contribute 2M

in their swaps.

- it is impossible to get more than 2M

for each set of 4 statements

- we can

get at least $2M$

for each set of 4 statements (by just picking one of LP

or $NOTLP$

to be nonzero, and pick up the 2 relevant statements)

- so if M

is huge (in particular, more than the total possible value of the edge transactions, so like $2|E|+1$

), we indeed get exactly $2M$

for each set of 4 statements, and we enforce P

or Q

in each statement to have zero fee.

Now, since there are many configurations which get exactly $2M$

for each of the NAND swap sets, the tiebreak must come from the edge transactions, where each $E(X,Y)$

wants to have value 1 (but we already set many of them to 0), so in effect we are maximizing the number of nonzero $E(X,Y)$

's.

In other words, for each edge, we are putting a pebble on either one (or neither) of the vertices, such that we cannot put more than 1 pebble on a vertex. We are now trying to maximize the number of pebbles. The resulting pebbles must form an independent set, and indeed, as we are maximized, we must be the maximum independent set (caveat: this is actually only equivalent to MAX_INDEP_SET in a graph where everything has degree >=1, but we can take away all the lone vertices to begin with). This is because given a maximum independent set, one can go “Backwards” and select edges giving pebbles to the vertices. Since MAX_INDEP_SET is NP-hard, our problem must also be NP hard.

Summary and actually-useful takeaways:

Uniform price auctions save data cost. If there were an obvious algorithm in P then we should just provide it to the block proposers automatically. Since there isn’t, it means looking for heuristic algorithms is totally fine (and probably the best we can do).

Followup work ideas (possibly good for next workshop):

- get a heuristic algorithm
- get a heuristic algorithm given some “real life” conditions (for example, real life market state is not going to be adversarial; are there good models of what the “order book” for such auctions look like? Then we can use that information and do better)
- have a way for users to calculate how much fee they should give to have a good chance of getting mined.

Thanks to Barry Whitehat (@barryWhiteHat) and Boris Alexeev for valuable conversations.