网格说明

1.Our Vision

We want to add auto-market bot into OrderBook to improve capital utilization and attract institutions to make market. This auto-market bot is similar to UniswapV3.

2.Design Introduction

Several considerations:

- We need as few signatures as possible
- Easy enough for users
- Easy enough for circuit development

First,Need to define base and quote; base is goods, for example UNI, DOT, ETH; quote is currency, for example USDT, USDC, ETH; in ETH/USDC pair, ETH is base, USDC is quote. Our auto-market bot always earn quote.

Second,A normal market making will include buying and selling orders,for easy, circuit only need to deal with a buying or selling order in one transaction, not the both. so, we have defined the type of the order, type = 6 is selling order, type = 7 is the buying order.

Third, both buying and selling orders have different prices in different levels, we called orderN - order0, the pricesare ranked from high to low.Since the price field is not included in the order, there are only amountS and amountB, so, weadjust the price by increasing or decreasing amountS or amountB.To this end, we introduce gridOffset and orderOffset fields. GridOffset is used to change the price of different levels, orderOffset is used to change the price of reverse orders.

Fourth, for buying or selling order market making, we only need one signed order by user. for selling order, we need order0, for buying order, we need orderN.Then, orders at different levels will carry the signed order, our circuit can calculate and verify the order information according to the signed order and the specified level.When we calculate the order information at different levels, the corresponding reverse order information can be easily obtained. To make ensure the accuracy of the signed order information, we will verify the signature of the signed order in each matching.

Fifth, we call the order with the same tokenS and tokenB as the signed order as the forward order, otherwise, the reverseorder.All orders at different levels always start with the forward order, if the forward order is completed, then the matcher will create the reverseorder, then cycle back and forth.

The following table can better illustrate our design:

	Base	UNI/DOT			
	Quote(Currency earned)	USDT/USDC			
		Bid	**Ask**	**Bid Reverse Order**	**Ask Reverse Order**
	Input	amountB/amountS/level/gridOffset/orderOffset			
	Signed Order	orderN	order0		
	type	7 (buy base)	6 (sell base)		
	amountB	amountB	amountB + gridOffset * (level-n)	amountB	amountB + gridOffset * (level-n) - orderOffset
	amountS	amountS + gridOffset*n	amountS	amountS + gridOffset*n + orderOffset	amountS
PRICE high ... low	**order N**	amountS + gridOffset*N	amountB + gridOffset * (level-N)	amountS + gridOffset*N + orderOffset	amountB + gridOffset * (level-N) - orderOffset
	...	...	....	...	....
	order 2	amountS + gridOffset*2	amountB + gridOffset * (level-2)	amountS + gridOffset*2 + orderOffset	amountB + gridOffset * (level-2) - orderOffset
	order 1	amountS + gridOffset * 1	amountB + gridOffset * (level-1)	amountS + gridOffset * 1 + orderOffset	amountB + gridOffset * (level-1) - orderOffset
	order 0	amountS + gridOffset * 0	amountB + gridOffset * (level-0)	amountS + gridOffset * 0 + orderOffset	amountB + gridOffset * (level-0) - orderOffset
	StorageID	order0 - orderN storageIDN = SignedOrder.storageID storageID(N-1) = SignedOrder.storageID + 1 ... storageID2 = SignedOrder.storageID + N -2 storageID1 = SignedOrder.storageID + N -1 storageID0 = SignedOrder.storageID + N	order0 - orderN storageID0 = SignedOrder.storageID storageID1 = SignedOrder.storageID + 1 storageID2 = SignedOrder.storageID + 2 ... storageID(N-1) = SignedOrder.storageID + N - 1 storageIDN = SignedOrder.storageID + N		
	Amount Verify	1.cauculate ordern, order is from matcher 2.s = order.amountS/ordern.amountS 3.b = order.amountB/ordern.amountB 4.s and b in [0.9999, 1.0001]			
	Order Completion Verify	1.calculate ordern 2.b = filledAmountB/ordern.amountB 3.b in [0.9999, 1]	1.calculate ordern 2.s = filledAmountS / ordern.amountS 3.s in [0.9999, 1]		
	Level Verify	order.level <= SignedOrder.maxLevel			
	Order Direction	forward = SignedOrder.tokenS == order.TokenS && SignedOrder.tokenB == order.TokenB reserve = SignedOrder.tokenS == order.TokenB && SignedOrder.tokenB == order.TokenS			
	Storage Data Reset	If the direction of the previous Order is inconsistent with that of the current order storage.Data = 0, then continue to execute the matching of the current order			
	Same Data Verify	Singed Order vs Order(from matcher) same with accountID, feeTokenID, maxFee, validUntil, taker, type, gridOffset, orderOffset, maxLevel			

Several important notes:

1. all orders on the same level share one storage space, when the order is completed, circuit will reset the data for next order.
2. we have recorded the 'forward' field to mark the direction of the last order, then circuit canclearly know the direction of the last order
3. type = 7: orderN = orderLevel0, order(N-1) = orderLevel1, ..., order0 = orderLevelN; type = 6: order0 = orderLevel0, order1 = orderLevel1, ..., orderN = orderLevelN
4. For establish the relationship between different levels, we specify the storageId of orderLevelN = signedOrder.storageID + level, this is very important.Without this relationship, there will be serious bug.
5. the significant digitsof the circuit are limited to 7, but the matcher matching may lead to exceeding 7. At this time, we allow rounding, so the amount numerical calculation will not be so accurate, and theerroris 1 / 10000. It should be noted that the amount of base currency will only be less than the calculated amount, not more, otherwise there will be problems in matching. Because we always earn quote.