TL;DR This post discusses a programmable mechanism to enable smart contracts to optimise hard problems. A possible use case is decentralised combinatorial markets which is often used in the energy sector.

Combinatorial problems has many similarities to proof-of-work. They are hard to solve, but easy to verify. This means that if any problem solver on the system finds a good solution to a problem, it would be computationally easy for the rest of the network to verify it.

For the sake of simplicity, we will assume that we only have one combinatorial problem of interest. It is straight forward to generalise this idea to multiple problems. Associated with the problem, let f(P, R)

be a mining function of the problem instance P

and a random number R

. The result of s = f(P,R)

is a solution (not necessarily the best). Let further g(P, s)

generate a real number such that g(P,s_1) < g(P,s_2)

of s_1

is a better solution than s_2

. These definitions should be very familiar as it is just an abstraction for intractable problems - PoW being one of them.

To enable combinatorial optimisation in smart contract, we create two new Merkle trees associated with the block. The first Merkle tree is the problem statement P_i

and the second Merkle tree is the solution set S_i

submitted included in block i

. The solution set S_i

contains the solutions to problem P_{i-1}

. Each of these solutions can be compared by using the function g(P_{i-1}, s)

for s \in S_i

. The best solution can now be selected and used to update the state database to make the solution available inside the smart contracts.

As a concrete example, imagine solving the traveling salesman: We are building a smart contract that plans a route given some waypoints. At block time i

, clients submit way points to P_i

which defines the problem. At time i + 1

, solvers (i.e. problem miners) of the system uses the search function f(P_i, R)

to find the best solution and submits the solutions to S_{i+1}

. After the block i+1

is mined, every node verifies the solutions and applies the best to the special entry in the state database. This makes the solution available to other functions in the contract.

To make this protocol secure, we require that R = H(H(nonce + pk))

thereby engraving the public key into the solution and making it intractable for a malicious solver to dictate R

. In this setting, f

and g

could be programmable functions inside the smart contract. This could be done by introducing keywords mineable

and objective

to specify their purpose.

Finally, it is worth noting that for at least some classes of search functions, it can be demonstrated that the search is most efficient when repeated searches rather than long search. Simulated annealing is one example of this.

Applications

Probably the most intriguing use case of this technique is combinatorial markets. Combinatorial markets is a subclass of smart markets in which a bidder can pick a combination of non-fungible tokens and place a bit. Combinatorial markets are already used in the energy sector as well as for flight traffic. The market clearing algorithm then optimises for economic value freed.

Concretely, let A, B, C be three non-fungible tokens representing a hotel room, a train ticket and a flight ticket. The sellers wish to sell these for $120, $20 and $200. Let x, y, z and w be users who wants to buy these items. User x wants to by A and B for $230. User y wants to buy C for $160. User z wants to buy A for $130 and user w wants to buy B and C for $230. The bids of x and y amounts to $390 and the bids of z and w is $360. Users x and y wins the auction, and payments are distributed according to trading value to ensure fair deals all around the table.

I’d be happy to hear your thoughts on this proposal.