Solving NP-Complete Problems Using the Ethereum Blockchain (*NP-Pay*)

Abhimanyu Gupta, Yizhou Yu, Claudia Dabrowski
aag245@cornell.edu, yy372@cornell.edu, cd432@cornell.edu
Cornell University

Abstract. Many blockchain platforms require computational power to produce proof-of-work. Miners solve computationally hard problems to provide these proofs. With the rise of technology available for miners to use, we wanted to harness this computational power into something other than proof-of-work. There exists a class of problems, NP-complete, which cannot be solved in polynomial time but a given solution can be verified in polynomial time. Such problems are common and in need of solutions. Our goal is to match users with computational power to solve NP-complete problems. We create a marketplace, called ***NP-Pay*** to facilitate interactions, outline a reward structure, and ensure payments by deploying it as a smart contract on the Ethereum blockchain.

I. INTRODUCTION

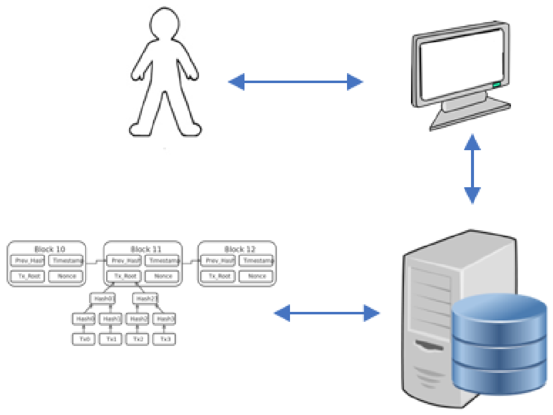
With the rise of blockchain ecosystems, miners gather to collect crypto assets by performing computationally expensive proof of work tasks and transaction verifications on the blockchain. These miners engage with the blockchain due to its guarantee of payment contingent on correct task completion, which is enforced by the rules of the ecosystem. Further, miners are incentivized to verify others' transactions into the blockchain by receiving a monetary reward. In Ethereum, these ”transactions” refer to asset exchanges, smart contract postings, and interactions with the smart contracts.

However, these market players can provide more value to the ecosystem by directly engaging with other players. By leveraging the guaranteed payment system and smart contract platform provided by Ethereum, players can be paired together to perform tasks for one another, creating a labor marketplace. We aim to create this marketplace called *NP-Pay* by allowing players to leverage computational power made available by other players to solve tasks for pay. Therefore, our system provides a means of solving computationally difficult problems through the power of markets.

II. HIGH LEVEL FRAMEWORK

A. *Structure*

The following diagram demonstrates the environment and interactions occurring within our system:



To increase usability, a call to the smart contract would be done through the interactions of a user, UI, server, and the blockchain. We assume that the server is secure. The UI gives users to option to issue, solve, or verify problems. User actions are sent to the server, which hashes some information and posts it to the blockchain. It is important to note that only the server interacts directly with the blockchain. Any information that the user sees it strictly through the UI.

B. *NP-Problems*

We restrict our set of problems in this market to NP-complete problems. This is because NP-complete problems share the property that, while there is not yet a way to find a solution in polynomial time, they can be verified in polynomial time given a solution. As such, we are able to determine whether a solution is correct in a reasonable amount of time, and thus verification is achievable. Furthermore, there is a need for solutions to NP-complete problems as they arise in many practical settings. For example, manufacturing companies solve SAT problems to configure the best set of features for their products[3], and newspaper editors generate crossword puzzles by encoding them into constraint generation problems. Through the use of a smart contract, we are creating a marketplace to match players with extra computational power and problem solving skills with people who are willing to pay for these solutions.

There are many different input examples that *NP-Pay* will be able to handle. We envision a call to the smart contract to be done through a webpage, with options to issue a problem, solve a problem, or verify a solution. We focus on the SAT problem for the scope of our research:

SAT, otherwise known as the Boolean satisfiability problem. This is the problem of determining if there exists an interpretation of variables that satisfies a given Boolean formula. A problem issuer would submit a text file to the server in the following format:

$$num\_variables$$

$$clause_1$$

$$clause_2$$

$$clause_n$$

where $num\_variables$ represents the number of variables to be assigned true or false of type int and each of the n $clauses$ represents a clause of the problem. The variables in each clause are represented by an int contained in [1, $num\_variables$], and are preceded with a '-' if the variables has been negated. The problem of finding an assignment to the variables is NP-complete, where a worker would return both a decision (Yes or No) and the optimal assignment if Yes, else null.

Posting an entire problem to the blockchain can be very expensive, especially when problem sizes are large. Therefore, once a problem is submitted to the server, the server will hash the problem and post only the hash to the blockchain. The blockchain will return a unique problemId that will be used to identify the problem in the future. In future iterations of this platform, it may be possible to use IPFS to expand the size of problems which are handled.

C. Problem Solution

After the problem hash has been posted to the blockchain, the server will post the full problem, un-hashed, to the UI. This will allow workers to see the problem and solve it. Once a worker finds a solution to the problem, they will submit it through the UI. The server will not automatically post the proposed solution to the blockchain. We recognize that if the server attempts to post a full solution to the blockchain, someone with more network power can take the solution before it is posted, submit it as their own to the blockchain, and later reap the rewards associated with the solution. To protect against such attacks, the server will first post a hash of the solution with their key to the blockchain. Once the server sees that the hash has been accepted and posted to the blockchain, it will then post the full solution.

D. Verification Process

After workers publish their solutions to the network, they can request the network to verify their solution in order to gain the reward of solving the problem. We recognize that the decision whether or not the workers have solved the problem cannot be made only by the problem issuer, because a malicious issuer can simply reject every solution that workers propose and prevent them from getting the reward. Hence, verification must be done by players in the network. We propose the following steps:

1) The problem issuer broadcasts the problem to the market, along with a master verification function and the reward that it puts up to pay the market.
2) A worker runs computations to solve the problem and posts the results to the market. It also puts in a deposit, which is used to prevent malicious workers from flooding the network with invalid solutions
3) Other players in the market can verify the solution off-chain. They can then cast a vote to indicate whether they believe the solution is correct or not.

The verifier will submit Yes for a correct solution. If the verifier finds that the solution is incorrect, they will submit No and a list of the clauses that are incorrect given the solution. The Yes or No votes will be posted to the blockchain, while the clauses provided by No votes will be retained on the server. If the verification function is ran, the blockchain will receive the incorrecly solved clauses from the server and run the verification function only on those clauses. Given the results on these clauses, it will determine if the solution is correct. This is to decrease the cost of running the verification function on the chain.

Each verifier will account for 1 vote, which is enforced by mapping the unique address of each voter to their vote. When a voter attempts to verify a solution, they will be required to place a deposit with their Yes or No vote that will be returned if their vote is correct. This is to disincentivize answers that are dishonest and instead provide incentive to vote if a verifier believes their answer is correct. We make the assumption that an honest player will return the correct answer to the contract, however we also assume that not all players are honest. As such, we define a protocol for disagreement resolution:

1) Uniform Response: All verifiers vote the same answer (Yes or No) and come to a universal consensus about the solution. No further action needs to be taken.

2) Divergent Response: At least one response differs from the others. Simply choosing the answer that the majority agrees with leaves this protocol susceptible to the problems outlined in the Selfish Mining paper[1] and pools of miners.

Therefore, we require a master verification function which is defined by the problem issuer at the start of the contract. This verification function is the truth function which holds the final decision over the validity of a solution. This can only be executed if there are dissenting verification votes. In order to run this function, we consider 3 situations which would trigger this action.

1) Manual Trigger: A user may trigger the master verification function by calling on it in the smart contract. This can only be executed if the caller has voted on the verification of the function, as to prevent a premature end to the verification voting session. Furthermore, this can only be called if a certain minimum number of votes have already been cast for the same reasons. This user will bear the gas cost associated with the function, but will also receive a higher reward for bearing extra risk if they are correct.

2) Time Period: The contract will automatically trigger the master verification if a specific time period has elapsed from the first vote and there are differing votes. This enforces that a final decision will be made while enabling an ample time period for any parties interested in engaging in verification to vote. This will occur regardless of the number of votes that have been cast, meaning that there is no minimum vote threshold in this scenario.

3) Vote Threshold: The contract will automatically trigger the master verification function if a number of votes has been cast before the time period has elapsed. This would indicate an adequate amount of activity has occurred without a manual trigger by a voter.

E. Reward Structure

In this section, we formally define the reward structure for players in the verification process. We define the gas cost for running the master function c, voting deposit cost d, total number of votes n, and proportion of voters with the correct answer $p \leq 1$. In the Uniform Response scenario, there is no dissent, and as such there is no need to run the master verification function. Every verifier is returned their initial deposit d and the solution is either approved or rejected based on the votes.

In the case of a Manual Trigger, the caller bears a cost of $d+c$, with $np-1$ other voters who were also correct, bearing a cost of d each. If the caller's verification vote is correct, the total that needs to be returned to cover deposit costs is $ndp+c$. This leaves a residual reward total of $nd-ndp-c = nd(1-p)-c$ for the correct voters. The caller would receive a 10% cut of this net amount for bearing extra risk, and the remaining 90% would be split evenly amongst all correct voters, including the caller. In summary, a caller would reap the benefit:

$$d + c + \frac{nd(1-p)-c}{10} + \frac{0.9 \cdot (nd(1-p)-c)}{np}$$

and a voter on the caller's side would reap:

$$d + \frac{0.9 \cdot (nd(1-p)-c)}{np}$$

If the caller was on the wrong side, however, each verifier on the correct side would receive:

$$d + \frac{nd-ndp}{np} = \frac{dp}{p} + \frac{d-dp}{p} = \frac{d}{p}$$

In our other two scenarios, a vote threshold or time period passing, no single voter bears extra risk by calling the master function. Rather, the network of voters share the risk. Thus the final reward for the correct side would be the same as above:

$$d + \frac{nd-ndp-c}{np} = \frac{d}{p} - \frac{c}{np}$$

F. Market Dynamics

Finally, a major component of our framework relies on the power of marketplace dynamics to fairly match labor to workers, and force changes to either party if it is necessary. For example, consider a job J which requires a solution to the NP-complete Traveling Salesman Problem with 100 cities and some maxDistance d. If company A offers this task at compensation \$1, a worker would need to determine if this compensation is sufficient for the work they will be completing. In this marketplace, a worker would only have this task as an option for work, and their choice is restricted. In another scenario, if company A and company B both offer these tasks at rates \$1 and \$2 respectively, a worker would gravitate towards the higher compensation, and company A would have to modify their compensation to meet demand such that they will attract workers. In creation of this framework, we aim to create a free market where tasks can be priced at their true value, determined by all parties in the marketplace.

III. CODE

This framework was written in Solidity for the Ethereum Blockchain. The code for the smart contract, along with the UI, can be found here.

IV. RELATED WORK

In their paper *Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems*[2], Oliver et al. propose a blockchain for which miners have the choice to solve NP-complete puzzles or proof-of-work algorithms. The only similarity this paper has to our model is that it uses the blockchain to find solutions to NP-complete problems. Their implementation is completely different from ours. We create a marketplace to match problem issuers with solvers, while their paper proposes implementing NP-complete puzzles into the blockchain. They allow miners to work on bettering solutions to the puzzles in return for a mining difficulty reduction. They also do not have a structured verification and reward system like we propose, and merely rely on the network to verify and accept it.

V. Future Work

We look to advance *NP-Pay* in the future by performing an expansion on the set of problems available for issue. Namely, we restrict problems to SAT, but can see the use of Traveling Salesman, Vertex Cover, Independent Set, and other commonly known NP Problems. Furthermore, we would like to see the impacts of *NP-Pay* on the network in terms of network load. Finally, we would like to study the behavior of problem issuers, problem solvers, and verifiers to determine the efficacy of our rewards and incentive structure, namely to see if any adjustments need to be made.

VI. Conclusion

There exists a lot of computational power that is not being taken advantage of. We presented *NP-Pay* as a marketplace to match users with computational power to problem issuers with NP-complete problems. It facilitates the matchings, but also has a framework for the reward structure and verification of solutions.

Acknowledgement. Thanks to Emin Gun Sirer for his help throughout this project, especially with providing direction at the start of our research.

References

1) Eyal, I., Sirer, EG. Majority is Not Enough: Bitcoin Mining is Vulnerable. http://fc14.ifca.ai/papers/fc14_submission_82.pdf

2) Oliver, C., Ricottone, A., Philippopoulos, P. Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems. https://arxiv.org/pdf/1708.09419.pdf

3) Volk, M. Using SAT Solvers for Industrial Combinatorial Problems: https://pdfs.semanticscholar.org/abf6/49d44ca622a559315d8f44c50d6ba141b2b8.pdf