Decentralized combinatorial optimization.

CHRISTIE, L.A.

2020

Decentralized Combinatorial Optimization

Lee A. Christie[0000−0001−8878−0344]

School of Computing, Robert Gordon University, Aberdeen
l.a.christie@rgu.ac.uk

Abstract. Combinatorial optimization is a widely-studied class of computational problems with many theoretical and real-world applications. Optimization problems are typically tackled using hardware and software controlled by the user. Optimization can be competitive where problems are solved by competing agents in isolation, or by groups sharing hardware and software in a distributed manner.
Blockchain technology enables *decentralized applications* (DApps). Optimization as a DApp would be run in a trustless manner where participation in the system is voluntary and problem-solving is incentivized with bitcoin, ether, or other fungible tokens. Using a purpose-built blockchain introduces the problem of bootstrapping robust immutability and token value. This is solved by building a DApp as a smart-contract on top of an existing Turing-complete blockchain platform such as Ethereum.
We propose a means of using Ethereum Virtual Machine smart contracts to automate the payout of cryptocurrency rewards for market-based voluntary participation in the solution of combinatorial optimization problems without trusted intermediaries.
We suggest use of this method for optimization-as-a-service, automation of contests, and long-term recording of best-known solutions.

1 Introduction

1.1 Motivation

Combinatorial optimization is an important class of computational problem, and may be tackled using some form of incentivized cooperative optimization. However, this requires solving the cheating problem. It also requires management by a centralizing party, and some degree of trust between parties.

Smart contracts on an open, decentralized blockchain allow us to automate the issuance of reward without the need for a trusted third party. This can enable us to co-ordinate problem solving between multiple parties. The existing infrastructure provided by open, public blockchains such as Ethereum provides an opportunity to build a decentralised application for combinatorial optimization.

1.2 Cooperative Optimization

In a traditional approach to optimization, multiple participants may tackle the same problem, either at the same time or at different times with no co-ordination.

Each attempt to optimize the function is done without regard to other attempts. Optimization in isolation can be seen when solving benchmark problems, which is a competitive activity, with results published at the end of optimization process.

We define *cooperative optimization* as multiple *hosts* working together in collaboration to solve an optimization problem. In contrast to competitive optimization, the hosts are allied in a team and do not gain from the loss of another host. Hosts could all be under the control of a single entity, the *client*, or could be part of a volunteer computing project where the problem is specified by the client. Cooperative optimization may be practiced by 'grid search' over a search space, 'parameter sweep' varying the parameters of an algorithm, or by parallel algorithms such as a genetic algorithm with island models. The degree to which hosts share information varies by technique, however it is typically not the case that a host loses by helping another host succeed.

1.3 Incentivized Cooperative Optimization

When multiple independent participants take part in cooperative optimisation, some form of incentivization may be introduced. Incentivization may take the form of financial reward, or non-monetary reputation/scoring.

Typically under cooperative optimization systems, hosts are expected to carry out a set program/algorithm and report the result. This program could be, for example, exhaustive search over a small subspace, or running a particular algorithm with a given set of parameters. It may be the case that carrying out these instructions yields no useful result, for example if the subspace contained no good or viable solutions, or the parameter set was sub-optimal for the problem. Participants are rewarded for the amount of computation done.

It may be possible for hosts to 'cheat' by falsely reporting that the work was carried out. Verification is possible if the host is required to perform some 'residual' side-calculation, but often the only way to ensure the residual is computing correctly is for the client to redundantly re-issue the same work to another participant, assuming the verifier is incentivized to perform the verification diligently and not cheat at verifying.

This issue is more broadly described as *the cheating problem* in volunteer computing projects, and solutions have been proposed. [4] [6] In this work, we propose a novel system of distributing optimization without central control, and without the need to control for cheating.

1.4 Outline

In section 2, we give background on decentralized applications (DApps), the concept on which the proposed system is built. In section 3, we outline the concept of decentralized optimization (DOpt) proposed. In section 4, we discuss the work which has been done to implement a proof-of-concept for the DOpt system. In section 5, we propose potential practical applications which further motivate the development of decentralised optimisation. In section 6, we outline the further work which needs to be done.

2 Decentralized Applications (DApps)

2.1 Bitcoin

There has long been a desire to implement a peer-to-peer digital currency. Traditionally, a system without the need for trusted intermediaries faces the issue of *double-spending* - reversing a transaction after receiving a good or service. Many systems had been proposed and attempted, however the first widely-adopted decentralized digital currency has been Bitcoin.

The Bitcoin white paper [8] put forth a solution to the double-spending problem of digital currencies in the form of proof-of-work by partial hash inversion, a solution to the Byzantine generals problem [7]. This allows multiple parties in a distributed peer-to-peer network to reach a consensus about the current state of the system without any trusted parties or identification. Bitcoin is regarded as the first successful decentralised application (DApp).

2.2 Proof of Work (PoW)

Distributed ledgers are a means of coordinating on the state of a system. They need a means of agreeing on the state. A naive voting system would be vulnerable to Sybil attack [3], whereby a malicious agent gains multiple votes by adopting multiple identities. PoW serves as a way of randomly selecting a participant in the network who decides on which new transactions to append to a ledger.

Transactions are grouped into *blocks*. Proof of work requires miners to solve a satisfaction problem to mine a new block and attach it to the *blockchain*, a cryptography-hashed backwards-singly linked list of blocks. Rewriting old state in the blockchain is not possible without redoing the work at a rate faster than the honest miners, which provides security to the blockchain. The method used by Bitcoin is *partial hash inversion*, where miners update a nonce value in the block header until, by chance, the hash of the block header is less than a target value. If the target begins with n binary zeros, the probability of random data hash being satisfactory is 2^{-n}. The target is automatically adjusted such that the average period of block mining is 10 minutes and a transaction is considered practically irreversible after 6 blocks (1 hour).

It is important to note that each solve attempt is a statistically independent event. This is a key property known as *zero progress*. The proof of work algorithm is designed to have the following properties[1]:

- Zero Progress - There should be no learnable structure in the problem, so that each attempt is a statistically independent event.
- Asymmetric - It must be computationally expensive to solve the puzzle, but trivially easy to verify.
- Scalable Difficulty - The difficulty of the puzzle should be able to be automatically scaled to set the target amount of time to solve.

[1] Agreement on the exact desirable properties and their relative importance is debatable, and may vary between blockchain designs.

- Not Predictable - No user should be able to start on the next puzzle until the current is solved.
- Not Beneficial - The computation should serve no purpose other than proving that the miners consumed electrical energy to secure the blockchain.

Given that the current energy consumption of the bitcoin network use to proof of work is estimated at around 71 TWh/year [2], it is often suggested that proof-of-work be replaced by useful computation such as protein folding, or in our case, optimization. Such useful work fails to meet most of the desirable properties of a proof-of-work algorithm given above.

2.3 Smart Contracts

A *smart contract* is an agreement between two or more parties which is enforced, not by law, but by software. [10] Parties involved may be persons, companies, or autonomous software agents.

Platforms such as Bitcoin allow for smart contracts. In the context of Bitcoin, a smart contract is code embedded in the blockchain as a script and replicated across all nodes on the network. The code can be executed based on transactions and can be used to pay out native tokens in the form of *bitcoin* (BTC) when pre-defined criteria are met.

The script comes in two parts: the *locking script* and an *unlocking script*. The locking script is attached to a bitcoin output. The unlocking script redeems the value to use as an input to another transaction. If the two scripts concatenated together forms a valid execution, the spend is valid, otherwise is it rejected.

The *Bitcoin scripting language* (Script) is intentionally Turing-incomplete. [12] This is a limitation with implementing arbitrary functions. The rationale is to ensure that contracts are executed in a deterministic number of operations, preventing denial of service attacks. Additionally, scripts are stateless: the blockchain only records whether a given output has been *spent*, and spending is all-or-nothing. The following subsection will cover the basic concept of using scripts to control cryptocurrency ownership, whereas subsection 2.5 we will discuss a Turing-complete system, which would be necessary for distributed optimization.

2.4 Standard Payments vs Transaction Puzzles

Most Bitcoin transactions are simple payment of value (bitcoins) from one user to another using the standard Pay-to-Public-Key-Hash [11] (P2PKH) script.[2]

The details of the script are omitted for brevity. The effect of this script is checking that the following two conditions hold of a future unlocking transaction which attempts to spend the given transaction output:

1. reveals the public key whose double-hash (known as an *address*) matches the previously specified recipient address; **and**

[2] Other scripts such as P2SH-wapped and native bech32 are also common.

2. is signed by the private key corresponding to the revealed public key.

Note that both of these conditions are encoded in the locking script specified by the sender (constructed as standard by the sender's wallet software). The script is written to ensure the value is transferred to the intended recipient.

One uncommon use-case of Bitcoin scripts is known as a *transaction puzzle*. A transaction puzzle does *not* specify a recipient, and therefore is of the class of *anyone-can-spend* transaction outputs.

Instead of the locking script being designed to target a specific recipient, it pays anyone who can provide the solution to a satisfaction problem, such as providing a blob of data whose `SHA256` hash is equal to the predetermined value.

One potential application of a Bitcoin transaction puzzle could be set so as to require a solution to a satisfaction problem, however f must be implementable in the Turing-incomplete Bitcoin scripting language.

Other limitations exist, such as the requirement for the entire puzzle reward to be paid out to one single solver. The reward cannot be shared for partial solutions or best-so-far solutions.

Additional details need to be considered. For example, if a time-limit is to be set on the problem (after which the problem-setter recovers their funds), a time-lock will need to be set, which is an extra complexity on the script. This was not used in the hash puzzle example above.

One major limitation is that any anyone-can-spend transactions are highly vulnerable to *mempool attacks*. A mempool attack is one in which in which an attacker intercepts the solution (which is broadcast publicly) and re-transmits their own solution with a higher network priority. The result is that the attacker is rewarded with the entire prize amount and the honest participant receives nothing. Such attacks can be automated anonymously on the network.

2.5 Ethereum

Ethereum [1,14] is a platform for Turing-complete DApps. It uses a native token called *ether* (ETH) which functions as a currency and also is used to pay for execution time of smart contracts. Contracts execute on the global, decentralized *Ethereum Virtual Machine* (EVM).

A smart contract account can 'hold' ether just as a user can, and interactions with the contract can result in the contract sending and receiving ether. A simple example is a *faucet* contract containing two methods, one default method marked 'payable' into which users may donate ether, and another from which another user may specify a desired amount, and the contract will automatically payout the requested amount. Arbitrarily complex programming logic and statefulness can be used in the design of EVM DApps in determining payout conditions and amounts.

2.6 Game Theory and DApps

Participation in a DApp is typically voluntary, incentivized, and pseudonymous. Identities with negative reputation can be abandoned and replaced. This means

that iterated interactions with the same bad actor may not be detectable, and participants with high reputation my be engaged in negative behaviour under an alternate identity. It may be assumed that participants will use any possible exploits in the system for self-profit. DApps requires careful, and explicit consideration of game theory in their design and security auditing of their implementation.

It is important that all rules of the DApp be enforced either by the terms of smart contract directly, or indirectly. Rules enforced directly by smart contract cannot be broken as the software does not allow it. Rules enforced indirectly may used a 'watchtower' system of enforcing a financial penalty in the event that one participant cheats and another participant detects the cheating. this may be used where it is infeasible to enforce a given rule directly. Such penalties are only possible when potentially-cheating actions require collateralizing tokens, typically under some time-lock.

3 Decentralized Optimization (DOpt)

3.1 Decentralized Optimization

In this work we introduce the concept of *decentralized optimization* (DOpt), a system in which participants (*hosts*) race to solve an optimization problem in real time without central co-ordination or requirement for trust. Distributed optimization differs from cooperative optimization with incentivisation in that limited sharing of information (which could help competitors) is practiced in exchange for rewards. Hosts are not required to share information as the optimization proceeds, but do so out of incentivized self-interest.

DOpt does not face the issue of cheating in the same way as traditional volunteer computing projects. Part of the novelty of this approach is that following a prescribed process is not a required behaviour. In fact, participants are able to use any algorithm without the need to disclose the details of their algorithm. Hosts are given the freedom to tackle the problem how they wish, and are rewarded for progress, not process. This incentivizes the hosts to use methods and computational resources that will be competitive.

3.2 Collective Optimization Trajectory

In implementing a distributed optimization system, the proposed solution is to use *collective optimization trajectory*, a ledger which tracks the best candidate solution found so far over time.

Each host will receive reward proportional to the time spend in the lead. This reward structure has the following properties, we state here with justifications for why each is desirable:

- Verifiability - It is easy to check that a given solution has a given newly-leading fitness. This enables the system to run without the need for a centralized verifier.

- Domain Invariance - Improvement can always be noted regardless of the scale of the domain. This is desirable for maximum generalizability to arbitrary optimization problems.
- Codomain Invariance - Improvement can always be noted regardless of the scale of the codomain. Again, this is for generalizability.
- Sybil Resistance - Hosts do not gain from manufacturing alternate identities, removing incentive to Sybil attack. This is necessary, since vulnerability to Sybil attacks is a severe security issue.
- Divisible - Not all of the reward will go to one host, as lead changes between hosts. This is desirable to having smaller participants receive zero reward, desensitizing participation.
- Predictable Payout - The total cost to the client can be allocated ahead of time. This is required since under an EVM system, the client will need to have the funds available and deposited in the contract.

The payout is distributed to host as shown in Fig. 1. (1) Client submits problem. (2) First activity recorded when red (solid line) host submits candidate, red's reign as leader begins. (3) First improvement when blue (dotted line) host submits improving candidate, blue is now leader and red's reign ends. (4) At end of optimization (N blocks after start), blue (dotted line) host is the final leader. (5) Reward distributed proportionately to each host's total reign duration.

The process of updating new best fitness and candidate is outlined in Algorithm 1.

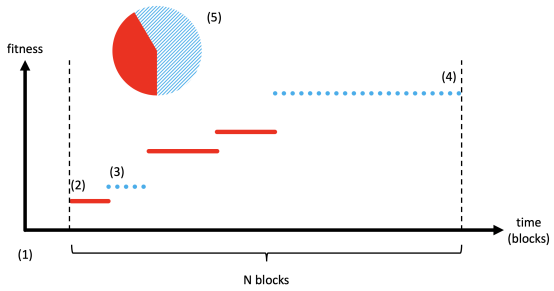

Fig. 1. Payout proportion based on time in the lead.

This proposed system does not perfectly reward effort, however it is easy to construct obvious alternatives which would fail to satisfy these properties. For example, rewarding improving solutions by a set amount (e.g. 10% higher fitness), or towards a target (closeness to 0) fails codomain invariance and predictable payout. Rewarding finding new good solutions within a distance to old solutions fails domain invariance and predictable payout. Rewarding detecting learnable structure in problems, or fully evaluating subspaces fails verifiability. Rewarding each host for discovering a good solution over certain fitnesses, or re-

warding hosts checking each other's work fails Sybil resistance. Rewarding only the best candidate fails divisibility.

3.3 Trajectory Broadcaster vs Recipient

When a host discovers a new best-fitness candidate and broadcasts it to the network, we call this host the *broadcaster*. For distributed optimization to work as a system distinct, collective optimization trajectory must posses certain properties. It must be more beneficial from a game theory perspective for hosts to broadcast a newly-found best candidate than it is to hoard the information for themselves. The problem of mempool attacks must be considered, as when a new candidate solution with the next best-so-far candidate, there is a period of time in which the transaction has not been included in the blockchain.

When hosts hear of a new candidate broadcast via the blockchain, we call these host the *recipients*. This process is illustrated in Fig. 2. (1) The current best candidate on the blockchain. (2) Candidate is imported by a new host. (3) Host algorithm(s) explores the space. (4) New solution is broadcast and host picks it up, redirecting the search. (5) Eventually this individual host finds a new best solution, broadcasting this and becoming the leading host.

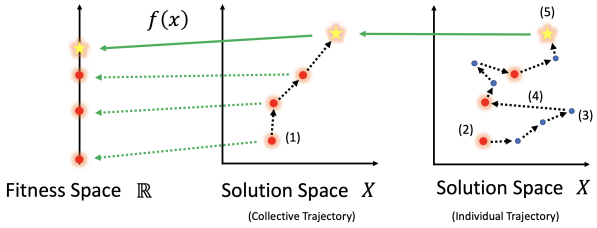

Fig. 2. Collective Optimization Trajectory.

It must be possible for a recipient's algorithm to learn from collective trajectory despite the information flow being severely limited. If algorithms cannot benefit from trajectory information, then the first mover with the fastest algorithm and most computational power has an unmitigated advantage and will not be overtaken, the reward will all go to one host and not be subdivided.

Proposed examples of algorithms benefiting from trajectory are[3]:

- GA - Treat received candidates migration events; add as elites.
- EDA - Contrast received candidates to candidates generated by current probabilistic model; adjust model.
- Hill-climber - Hill-climb to refine received candidate to local optimum; perturb from this local optimum to find adjacent local optima.
- PSO - Use received candidate to update global best information to swarm.

[3] Algorithms may be adapted, or specifically-designed for decentralised optimisation.

4 Proof-of-Concept

4.1 Implementation

A smart contract for decentralised solving of the OneMax problem has been implemented, with a Web 3.0 GUI using NodeJS and the React framework.[4] The function may be swapped out by replacing an arbitrary EVM function.

The current smart contract which has been built is using the Solidity [13] smart contract language for the EVM. Development and testing is running on the Truffle development framework [5] consisting of the *Truffle* development tools, *Ganche* development blockchain, and *Drizzle* Redux components for front-end.

The current implementation allows a client to pseudonymously connect to the DApp using a Web 3.0 enabled browser. Hosts can submit a candidate solution, and if it is an improvement, the new candidate and fitness will be recorded, and the broadcaster is recorded as the current leader.

A screenshot of the GUI is shown in Fig. 3. The two left boxes allow candidates to be submitted though a Web 3.0 enabled browser. The right box displays the current status of the contract, showing the current best candidate and fitness registered on the blockchain, with the address of the reigning leader. The UI runs entirely on the browser client without need for a back-end server. ♦ denotes a state-mutating action associated with a transaction fee.

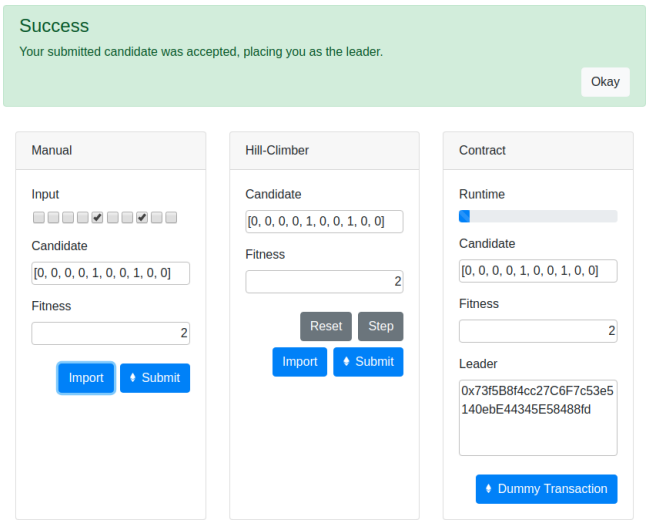

Fig. 3. A screenshot of the front-end for the prototype of DOpt.

[4] Source available at `https://github.com/leechristie/dopt-concept`

4.2 Security Considerations - Reentrancy

Payout for satisfying criteria is best done using a withdrawal system in which funds are allocated for a host and then the host initiates a withdrawal. This is a 'best-practice' design pattern in smart contracts which avoids triggering sending of funds with arbitrary conditions and reduces the potential for certain classes of security exploit known as 'reentrancy bugs' [9].

The 'current reign' duration is defined as the number of blocks since the last time an improving candidate was submitted to the contract. The 'total reign' for a given host is the sum of all reigns held by that host, i.e. the amount of time (in number of blocks) that the host was the leading host.

Currently, runtime is displayed from the block in which the first candidate was submitted until a fixed runtime of 15 blocks has elapsed. When a host is the leader for a contiguous n blocks, they are said to have a *reign* of duration n blocks. The client sets the *total reward* amount as the amount they are willing to pay for the optimization. The *total reign* is calculated as the sum of their reigns, and the reward is calculated is as

$$\text{assigned reward} = \text{sum of reigns} \times \text{total reward} \;/\; \text{total runtime}$$

for example if the total runtime is 15 blocks with a total reward of 0.0015 ether and the host has held the lead twice, for a duration of 4 then 3 blocks, the assigned reward is 0.0007 ether. A host may withdraw ether from the smart contract provided the requested amount is less than or equal to the assigned reward less the amount already withdrawn. At any time during the optimization of the function, the total reward may be increased by anyone (usually the client) who sends funds to the payable method. The method of computing balance given above will retroactively update all balances without additional effort, since the balance the tracked variables are sum of reign and amount withdrawn. The contract requires donated amounts be divisible by the runtime.

5 Applications

5.1 Optimization-as-a-Service

The primary application of DOpt is to set up a network whereby an optimization problem can be submitted to the DOpt network by anyone and solved by anyone. Both problem submitters and solvers may be anonymous on the network and do not need trust or communication between them.

Those submitting problems are in the role of *client*. The client designated the total reward amount along with a predefined total runtime (counted by N blocks mined on the blockchain). Optimization runtime begins when the first host submits a candidate and ends N blocks after the start time.

Those with spare CPU cycles which may be used to run optimization problems are in the role of *host*. The host decides which of the available active problems to attempt to optimize, and what algorithm(s) to run, which improving candidates to submit to the network, and when to stop and switch problems.

Some potential decision criteria are:

State: best, leader, max_runtime, sub_count, start, sub_block, completed
Data: candidate, fitness, sender, block
Result: updates host reign
if *fitness* $\neq$ *eval(candidate)* **then**
  revert;
if *fitness* $\leq$ *best.fitness* **then**
  revert;
if *runtime* $\geq$ *max_runtime* **then**
  revert;
best = fitness, candidate;
sub_count++;
if *start* = *null* **then**
  start $\leftarrow$ block;
else if *block* $\geq$ *submission block* **then**
  reign $\leftarrow$ block - sub_block;
  completed[leader] += reign;
  allocated += reign;
leader $\leftarrow$ sender;
sub_block $\leftarrow$ block;

Algorithm 1: Algorithm to update state on receiving new candidate.

- Age of Problem - The amount of time the problem has already been running before the host joins;
- Reward per Runtime - The rate at which reward is issued for being the leader;
- Competition - The number of other candidates which have been submitted and the rate they are being submitted; and
- Success - Whether a problem currently being attempted by the host is yielding successful improvement

A host may run as many or few processes and as much or as little compute power they wish on a given problem or problems.

As with many decentralized systems, price could be expected to be dictated by the market without the need to add pricing infrastructure. If the network contains a high number of active optimization problems and a low number of hosts, the price could be expected to trend higher. Clients submitting problems whose payout per unit time is low with respect to the market average will find little-to-no submitted candidates unless they increase the total reward amount to a competitive level. If the total number of hosts is high and the total number of active problems is low, most problems will be tackled by a high number of competitive hosts. However, deployment in a real-money scenario would be required to see how participants and pricing behaves in practice.

A client may set their reward level above market rate to increase the level of prioritization, resulting in more hosts devoting more CPU time to their problem. A client may set their reward level lower and increase the runtime if the importance of the problem is lower and they are prepared to wait longer.

DOpt may pay participants in ether (ETH). If the price volatility of ether as a currency disincentives use as a unit of account, it is possible to allow exchange of value using another token, such as tokens under the ERC-20 token standard, allowing use of application-specific utility tokens, or stable-coin.

5.2 Automation of Contests

In the optimization community, optimization contests are run wherein a set of benchmarks are given out and the most efficient algorithms provided for this benchmark set are awarded. The system could be used for regular (such as annual) or ad-hoc run contests.

DOpt is *not* a suitable direct replacement for this design of contest. As with PoW systems, as it does not attempt to maintain a level playing field for participants since advantage is given to both quality of algorithm, and level of compute power. Participants may still reveal the algorithms used however the system does not require this. Participants could compete for financial reward or prestige given a fixed time window in which to apply all algorithms and hardware resources they can to a given problem. The contest would be automatically run by the DOpt smart contract system.

5.3 Long-Term Recording of Best-Known Solutions

Some large optimization benchmark problems have best-known solutions which are improved over long periods of time. The same DOpt system run with a very long duration runtime, or modified to run endlessly, may be used to record the best-known solutions on the blockchain.

These current record-holder could receive a small payout over the span of holding the record. Note that if the system is modified to allow endless runtime, the client would need to top-up the total reward with additional deposits over time to allow the runtime to continue rewarding solvers.

Alternatively, the system could be run without financial incentive, the reward being only to have their record publicly available in the blockchain. Anyone citing the best-known solution to a given problem could cite the solution in the blockchain, which is verifiable by all.

6 Further Work

Work must be done to identify existing algorithms which can run competitively using the collective trajectory concept. In addition, which classes of problems are best suited to this approach needs to be identified. An API will be required to allow hosts to connect their algorithms to the DOpt network, automatically discover problems, submit candidates, and receive broadcast trajectory. A process will be required to prevent mempool attacks for secure deployment of DOpt.

References

1. Buterin, V.: A next-generation smart contract and decentralized application platform (2015), https://github.com/ethereum/wiki/wiki/White-Paper
2. Digiconomist: Bitcoin energy consumption index, https://digiconomist.net/bitcoin-energy-consumption
3. Douceur, J.R.: The sybil attack. Lecture Notes in Computer Science **2429**, 251–260 (2002)
4. Du, W., Jia, J., Mangal, M., Murugesan, M.: Uncheatable grid computing. Electrical Engineeringand Computer Science **26** (2004), https://surface.syr.edu/cgi/viewcontent.cgi?article=1025&context=eecs
5. Group, T.B.: Sweet tools for smart contracts (2019), https://www.trufflesuite.com/
6. Hine, J.H., Dagger, P.: Securing distributed computing against the hostile host. In: Proceedings of the 27th Australasian Conference on Computer Science - Volume 26. pp. 279–286. ACSC '04, Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2004), http://dl.acm.org/citation.cfm?id=979922.979956
7. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3), 382–401 (1982)
8. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), https://bitcoin.org/bitcoin.pdf
9. Solidity: Security considerations - re-entrancy (2019), https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#re-entrancy
10. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)
11. Wiki, B.: Pay-to-pubkey hash (p2pkh) - bitcoin wiki (2016), https://en.bitcoin.it/wiki/Hashlock
12. Wiki, B.: Script (2019), https://en.bitcoin.it/wiki/Script
13. Wood, G.: Solidity, the contract-oriented programming language
14. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (2017), http://gavwood.com/paper.pdf