EAI Endorsed Transactions
on Serious Games
Research Article EAI.EU

Business-Savvy Blockchains with Gamification: A Framework for Collaborative Problem-Solving

Stefano Dalla Palma[1,*], Damian Andrew Tamburri[2,†], Remo Pareschi[1,†], Carmine Cerrone[1], Willem-Jan van den Heuvel[2]

[1]University of Molise, C.da Fonte Lappone 86090 Pesche (IS), Italy
[2]Jheronimus Academy of Data Science (JADS), Sint Janssingel 92 5211 DA 's-Hertogenbosch, The Netherlands

Abstract

This paper proposes a design pattern that combines gamification dynamics along with blockchains for the purpose of using blockchain technology in a business-savvy fashion as support to a framework for collaborative problem solving, i.e., leveraging gamification to incentivize people to produce efficient, freely available and easily accessible solutions in the optimisation research and the potentiality of blockchain technology to safekeep the intellectual property on such solutions, marking the progress of problem solving as intellectual capital. The proposed gamification design pattern is then instantiated in the context of optimisation.

Received on 18 July 2018; accepted on 10 August 2018; published on 13 September 2018

Keywords: blockchain technology, gamification, optimization, design pattern, software engineering

doi:10.4108/eai.13-7-2018.155567

1. Introduction

Distributed ledgers are databases shared across a public or private computing network. Each computer node in the network holds a copy of the ledger, so there is no single point of failure. Blockchains are specific kinds of distributed ledgers such that every piece of information is mathematically encrypted and added as a new "block" to the chain of historical records held within the ledger. Various consensus protocols are used to validate a new block with other participants before it can be added to the chain. This prevents fraud or double spending without requiring a central authority. Blockchain technology [21] has recently gained massive attention from the media and companies worldwide, mostly because of its intriguing architectural property of maintaining a consistent copy of every transaction across the blockchain, hence guaranteeing overall tamper-resistance [5]. They are by far the most proven and widespread kind of distributed ledger, as a consequence of the consolidated use of blockchains as the distributed ledgers supporting cryptocurrencies like Bitcoin [14], Ether (https://www.ethereum.org/ether) and Litecoin (https://litecoin.org/). Indeed, the clearest and most striking success of blockchain technologies has so far been the creation of the universe of cryptocurrencies, whose constituent matter is given by digital currencies capable of self-certifying their authenticity in an algorithmic way, without the need for third parties such as central banking institutions and monetary authorities. While, to the broader public, blockchain is synonymous with cryptocurrency, the business community is increasingly realizing the enormous potential of this technology (well beyond the yet notable success of Bitcoins and companions) (i) for its ability to create a history of certified information, and (ii) to automate the execution and reporting of transactions across multiple stakeholders. Experimentations with blockchains that aim to meet existing business needs do exist, e.g., think of the several IBM and Ethereum initiatives around the matter (available online: https://www.ibm.com/blockchain/use-cases/ and https://www.stateofthedapps.com/). The table in figure 1 taken

*First author. Email: s.dallapalma@studenti.unimol.it
†Corresponding author. Email: d.a.tamburri@tue.nl, remo.pareschi@unimol.it

from the McKinsey report on blockchains[1] sums up well the applications made possible by these two uses of blockchain technology, which however have to be considered as communicating vessels rather than as insulated silos.

Supporting this trend, there are the numerous business blockchains that have been hosted by Ethereum and the Linux Foundation's Hyperledger technology forming an environment in which communities of software developer and companies meet and coordinate to build blockchain frameworks. For instance, Hyperledger Indy[2] is a software ecosystem for private, secure, and powerful identity that aim to put people in charge of decisions about their own privacy and disclosure. This enables all kinds of rich innovation: connection contracts, revocation, novel payment workflows, asset and document management features, creative forms of escrow, curated reputation, integrations with other cool technologies, and so on. Hyperledger Iroha[3] is a business blockchain framework designed to be simple and easy to incorporate into infrastructural projects requiring distributed ledger technology. The Australian PowerLedger[4], which utilizes the Ethereum blockchain, provides a peer-to-peer marketplace for renewable energy that raised 34 million Australian dollars through its ICO (Initial Coin Offering).

Within this paper we propose the use of gamification [7] along with blockchain technology as a basis for a design pattern to create tangible value for the players involved, i.e., for game participants and for game administrators. Gamification applies game design and game principles to non-gaming contexts, so as to improve the involvement of stakeholders in social ecosystems such as companies, government organizations and citizenries. A number of case studies has shown that gamification has generally performed very well, with employees producing more and better, information flowing more smoothly and effectively, citizens more involved in relevant events such as electoral consultations and so on and so forth. The use of gamification for crowdsourcing purposes has been studied even further, particularly in the context of ICT technologies [13] and one of the most successful recent applications has been the Netflix prize challenge where concurrent machine-learning campaigns were encouraged to improve Netflix's own recommendation system [1]. Our purpose is to bring this trend one step further up to its full potentialities for social good and economic development by marrying gamification with another powerful technology: the blockchain.

The use of gamification in the context of blockchain turns the problem of architecting with blockchains into the goal of building a software-defined, network-based ecosystem, a *marketplace* of participants (human, machine, or otherwise) who contribute computational power and creativity towards a predefined computational goal in a decentralized and secure fashion. Our target is given by optimization problems that find practical applications in a variety of fields such as e-commerce, logistics, scheduling, traffic management, storage allocation etc. A considerable number of problems can be defined via the maximization or minimization of some desired objective, hence approaches to problem solving, learning or discovery that employ a practical method to reach that objective (e.g., heuristsic and metaheuristics) are applicable to a broad range of disciplines including the ones listed above. Softwares that "solve" a mathematical problem, namely "solvers", are routinely applied to problems of real-world concern, producing highly-significant financial and resource savings, e.g., minimizing fuel by optimizing delivery routes; minimizing energy consumption by optimizing wind-farm/generator placement; minimizing the number of vehicles used to deliver products by optimizing the arrangement of products in those vehicles; minimizing the total amount of time (or cost) required to complete a set of tasks by optimizing personnel scheduling and tasks assignment, etc.; resulting in benefices for both product (or service) users and providers. The general criteria that we apply to justify our proposal are the following: (1) it is a small world: meaning that for someone who needs an optimized solution for a given problem, there might be someone who has concocted exactly that solution; (2) half of the world does not know what the other half is doing: meaning that, even if the condition as above applies, we might not be aware that someone next door has exactly what we need (and in the Internet age the whole Earth is one single village); (3) there must be a virtuous circle of mutual returns and benefits that binds together proposers of problems and producers of solutions, so as to create the pre-conditions for a marketplace of optimizations; (4) transactions must be recorded in an evolving archive shared among the stakeholders for the purpose of certifying transaction execution and of watermarking the latest point of optimization of a given problem, so as to clearly indicate where to go next. This is where technologies such as distributed ledgers in general, and blockchains in particular, turn out to be particular useful and promising. The theoretical basis for the above idea stems from the synergy we observed between blockchain technology and computational approaches that can be defined as *human-competitive* [15], namely,

[1] "Blockchain beyond the hype: What is the strategic business value?" Available online: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/blockchain-beyond-the-hype-what-is-the-strategic-business-value

[2] Available online: https://www.hyperledger.org/projects/hyperledger-indy

[3] Available online: https://www.hyperledger.org/projects/iroha

[4] Available online: https://powerledger.io/

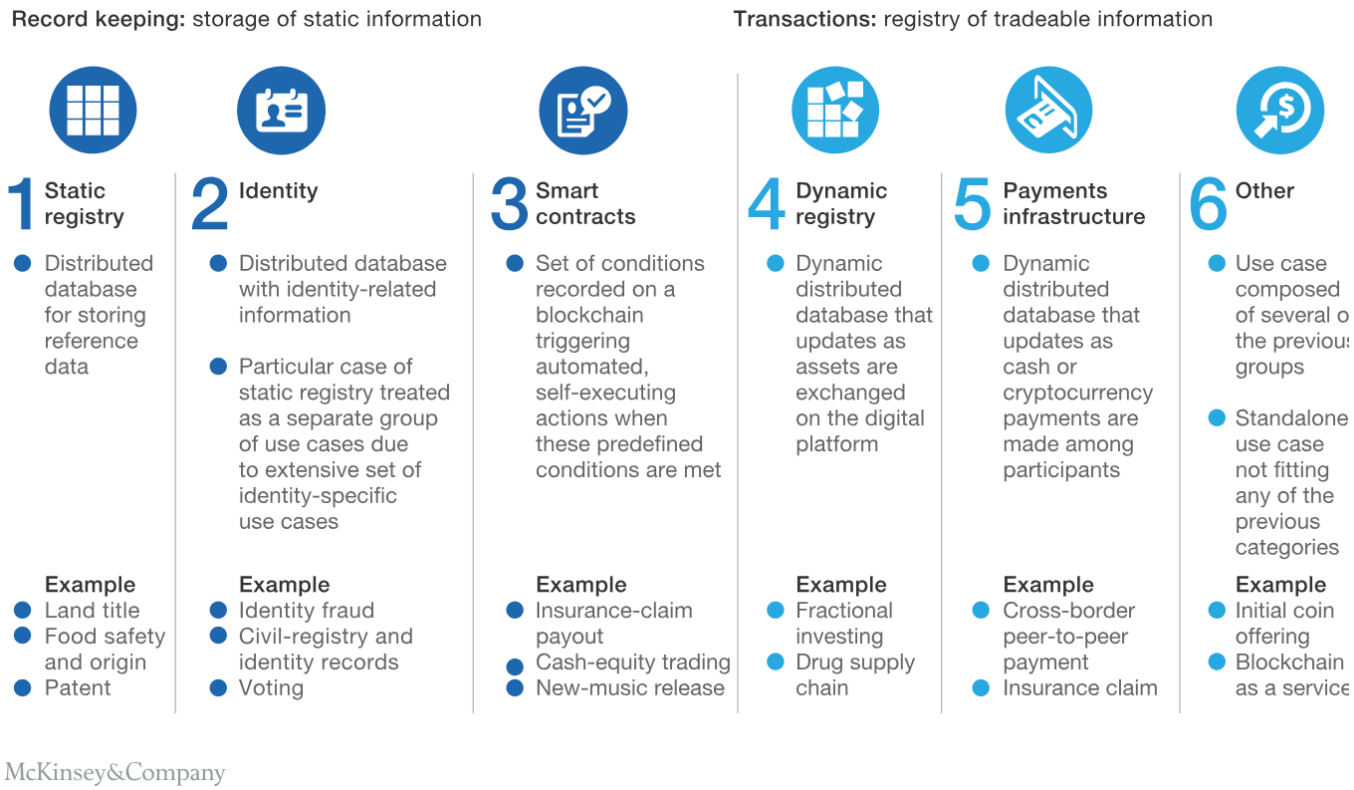

Figure 1. There are six distinct categories of blockchain use cases addressing two major needs: *record keeping* and *transactions*.

computational solutions which satisfy at least one of the following criteria:

1. The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
2. The result is equal or better than a result that was accepted as a new scientific result at a time when it was published in a peer-reviewed scientific journal.
3. The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
4. The result is publishable in its own right as a new scientific result, independent of the fact that the result was mechanically created.
5. The result is equal to or better than the most recent human-created solution to a long- standing problem for which there has been of increasingly better human-created solutions.
6. The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
7. The result solves a problem of indisputable difficulty in its field.
8. The result holds its own or wins a regulated competition involving human contestants (in the form of either live human player or human-written computer programs).

Our proposal envisions the use of blockchains as means to structure a "marketplace" of human-competitive solutions, a concurrent *community of practice* [19] aiming for shared solutions to address the pre-defined computational problem; rather than using a blockchain to architect a single software to address it, the blockchain shall be used as a support technology to structure its crowdsourced resolution [11] and continuous improvement. This very same pattern was adopted in the famed aforementioned Netflix challenge. The major benefits of using blockchains to structure a competitive marketplace in much the same vein are manifold, among which: (1) registration on the blockchain of the effective superiority of an algorithm or a solution, with consequent entitlement to automatically generated and unequivocal royalties both for game participants and for game owners, that is, the company administering the marketplace; (2) certified algorithms and solutions — each algorithm and solution are certified to belong to a provider and to

resolve a given problem; (3) facilitated registration and re-use of both open-source and proprietary algorithms and solutions; (4) payment through cryptocurrency coins which can acquire value on the cryptocurrency market and be exchanged even outside their primary use contexts in the scope of the marketplace.

In the next sections we illustrate the proposed design pattern to optimisation problems in general. The blockchain-based marketplace to address this solution consists of people and machines who compete to resolve computational optimization problems and are incentivized to create efficient, freely available and easily accessible solutions, making sure their ownership over proposed solutions is maintained in a safe, secure, and rewardable fashion by means of blockchain technology.

2. Gamified, Blockchain-based Marketplaces: How does it work?

The marketplace we propose as a design pattern to use blockchain technology in a business-savvy fashion is implemented in a public decentralized database that stores the computational problem to be addressed concurrently.

2.1. Design Principles

Design principle 1 – marketplace-based information sharing. In the case of optimization problems, the marketplace stores problem definitions, algorithms, and proposed solutions up to a certain point in time; marketplace participants are people who can contribute by posting their own algorithms and solutions for a given problem and share them with others upon payment.

Design principle 2 – marketplace cryptocurrency. A cryptographic private-public key pair is assigned to each user joining the community in order to edit the database to post a solution or to update an existing one. Each entry in the database can be edited only by its creator, who is identified by her public key. Public key is used to sign (encipher) the solution to avoid their free re-use. In this way, only the owner can decipher it with her private key and see in clear the solution or edit it. When the owner of a solution receives the payment for the solution after having sold it, the private key is used to decipher the solution and make it available to and usable by the buyer. A solution posted as open-source is not enciphered. In much the same way, the creator of a solution or an algorithm maintains the ownership over them.

Design Principle 3 – marketplace coopetition. The research literature in optimization is positively obsessed with playing the up-the-wall game [3]. There are no rules in this game, just a goal, which is to get higher up the wall (which translates to "obtain better results") than your opponents. Although some competition between researchers or research groups can certainly stimulate innovation, the goal of science is to understand. To this end, cooperation is also needed. Coopetition is a neologism coined to describe cooperative competition [2]. Coopetition games are statistical models that consider the ways in which synergy can be created by partnering with competitors. This tactic is thought to be a good business practice between two businesses because it can lead to the expansion of the market and the formation of new business relationships. We use the concept of coopetition in the context of the proposed marketplace. Nodes on the network are in *competition* to find the best solution for a given problem to win a reward; at the same time they *cooperate* and speed up the optimization research. The marketplace itself puts in competition people to find the best solution of a given problem in order to gain rewards and reputation as well as money by selling their own solutions. This way everyone proposing a problem could easily find efficient solutions. Moreover, the creator of a solution or an algorithm is certified to be his/her owner. Indeed, the marketplace can rely on existing blockchain techonologies (e.g. Ethereum) to ensure ownerships of solutions and more granular properties of blockchains to ensure security, e.g., proof of existence, proof of identity and proof of authorship – at the same time, consistency can be maintained with the three classic security approaches across typical blockchains: (1) identification; (2) authentication; and (3) authorization.

Design principle 4 – marketplace data-storage. Data stored in the blockchain-based marketplace and hence, within the public decentralized database are:

- *Problems to be solved* - these are described with a textual specification, an objective function, and a validation algorithm that automatically evaluates the proposed solutions.
- *Gamification board* - solutions and algorithms are stored and shared by all peers active across the marketplace.
- *Gamers data* - Gamers are encoded as marketplace users with their nickname, reputations and cryptographic public key to authenticate him/her.
- *Smart contracts* - they are a set of conditions recorded in the database such that transactions automatically trigger when the conditions are met [4]. Contracts are used to buy algorithms and solutions or transfer their ownership.

2.2. Architecture Elements

The software architecture of the proposed solution entails a *Solution-as-a-service* architectural style wherefore a solution marketed through the proposed marketplace, e.g., for an optimization problem, is addressed through the marketplace itself, and worked out by peers, but, at the same time, the solution is also *sold* through the same marketplace. Therefore, whenever someone buys a solution (e.g., a ML algorithm), she buys it as "a service", meaning that she gets that solution (e.g., the same ML algorithm) following the *pay-per-use schema*, while the ownership of the solution still belongs to its original producer. The style also envisions and ensures fairness: only when the ownership of a solution (e.g., a ML algorithm) is willingly transferred between marketplace users through smart contracts, then the ownership passes from the original owner to the new one.

The aforementioned architectural style requires the architecture elements listed below.

- ***Blockchain Node*** - is any user (proponent, competitor or validator) or machine that propose a problem to the community or resolve it, validate transactions, sell or buy solutions and algorithms as well as transfer their ownership through Smart Contract replicated on the blockchain and validated from other peers.

- ***Database*** - stores each problem specification, solution or algorithm and users' information. Only user that add an entry in the database, e.g. a solution, can modify that entry. Every operation performed on the database is stored on the blockchain. See *transaction* later on.

- ***Problem Specification*** - is any optimisation problem that can be *mathematically formulated* and require a solution. Mathematically formulation can be useful to have an unambiguous description but it is not mandatory.

- ***Proponent*** - is any user that proposes a problem to the community posting it into the database. She does not have to pay any fee or amount of tokens to post the problem into the database. A proponent only has to pay the owner of a solution for using it or use a freely available solution, if any. Proponent can also use solutions of problems similar to the proposed one. Proponent must provide a validation algorithm together with the problem description to allow the system to validate the solutions of the resolvers, automatically. She can also provide own algorithm to resolve the problem that competitors can set up and run to find a solution.

- ***Competitor*** - is a node which competes to find and efficient solution to resolve a problem and post it in the database. Competitor is not forced to find the best solution or to make it freely available to others. However, the mechanism of reward incentivizes her to find the best solution and make it open-source (i.e, freely available to the community) in order to gain both reward and reputation. Otherwise, the competitor must pay to post the solution into the database. See also section 2.3.

- ***Solution*** - is any solution provided by competitors to a given problem proposed by a proponent. Solutions are automatically validated by the system through the validation algorithm provided by the proponent and each solution value is kept visible to everyone (i.e., is not enciphered) in order to compare different solutions. If the validation algorithm accepts the solution, i.e., the solution is admissible for the problem, it is stored into the database. A solution can be stored in the database as a private solution, that is, a solution provided to other peers behind payment or it can be shared freely to others. See also the reward mechanism in section 2.3 on how constraints bind the insertion of solutions into the database.

- ***Validator*** - is any node that validate transactions between competitors and database, database and proponent/competitor(s) or among proponent/competitor(s).

- ***Transaction*** - is any interaction performed among nodes and database or among nodes themselves that move rewards (i.e., tokens) or solution/algorithm ownership and must be recorded on the blockchain to guarantee the integrity of the system and the ownership of the solutions. A transaction can consist of "posting a solution into the database" (no matter whether the node gains reward or not), "transferring a solution or algorithm ownership" or "selling/buying a solution" etc.. Transactions are stored on the blockchain and validated to maintain the integrity of data and guarantee the ownership of solutions and algorithms for any given user.

- ***Smart Contract*** - in the context of the proposed framework, they are used to sell and buy solutions as a service or transfer their ownership and guarantee the success of such operations.

2.3. Gamified Rewarding Rules: Incentive and Reputation Schema

The gamification by-design aspects behind the proposed blockchain marketplace follows a set of rules for

compensating competitors and validators. The rules in question follow below:

(i) Each time a competitor posts a new successful validated solution into the database, that is better than the current best one for the related problem, she gets rewarded with a variable positive amount of tokens x, which is proportional to the improvement made by the new solution compared to the best one (we call the new solution an highest-higher solution "HH"), e.g. if a competitor posts a solution that improves the current best one by 10%, she gets rewarded by the reward for the best solution plus a propotion given by the improvement of 10% made by the solution. Indeed, the last few bits of improvement are much harder to find than the early ones, i.e. we will have to work harder to get 1% improvement late in the game, than 1% early in the game. The larger the difference between the proposed and the previous solution, the larger the reward gained by the proponent. This establishes a rewarding mechanism that increase with increasing solution improvement. If no best solution exists up to that time, i.e., no solution has been already posted, the competitor get rewarded by a fixed amount of tokens.

(ii) Each time a competitor posts a new successful validated solution into the database, that is no better than the best one for the related problem (i.e., mediocre solution), she must pay a positive amount of tokens y, which are deposited in the proposed problem specification as deposit and put on the market when other users will get rewarded. However, competitor still gets rewarded in a proportional fashion as in (i) with regard to the highest solution lower than the proposed one (we call that solution an highest-lower HL for the proposed one), i.e., the larger the difference between the posted solution and a HL solution, the larger the reward gained by the competitor. The gain or the loss of tokens depends on the quality of the solution: if the value of the solution is such as to allow to overcome the amount y the competitor obtain the reward minus y, otherwise she must pay y. If at the time of posting the solution there already exists a solution with the same value, the previous schema does not apply and the competitor must pay the amount of tokens y. The same applies for private solutions.

(iii) If a competitor posts a solution or edit a previous one as public - i.e., freely available to everyone - she gets rewarded with a positive amount of tokens x' >> x, that is, an amount of token greater than the amount she would gain posting the best solution. If the solution results to be the best one for the related problem and is inserted into the database for the first time, the competitor gains twice: the amount of tokens x for having posted the best solution and the amount of tokens x' for having made it freely available to the community. If the solution was previously stored into the database, making it public allows the competitor to gain the reward x'.

(iv) If a competitor, that has previously posted a solution as public, wants to make it private (i.e., available to everyone behind payment) she must pay a positive amount of tokens y' >> y, that is, an amount of tokens greater than the amount she gained for making it public. These tokens are deposited in the deposit for S and put on the market when other users will get rewarded.

(v) Each time an open-source solution is used indirectly or directly by the community, the owner of that solution gets rewarded with an increment of her reputation. (See Reputation Schema).

(vi) When validating a transaction, the validator receives a positive amount of tokens. Validators can require fees to validate specific transactions.

Incentive Schema. (i) and (ii) incentivize the nodes to post always the best solution to gain rewards, improving a lot the best one up to that time. Furthermore, rewards are proportional to the improvement made by the posted solutions with regard to a HH solution. Thus, competitor are incentivized to post very good solutions to gain more rewards. However, sometimes a suboptimal solution or algorithm - which is radically different from the state of the art or the best one - can be extremely valuable. Finding a new solution or algorithm opens new doors for future work, even if at the moment it does not compete with other ones. (ii) also incentivizes this aspect, still rewarding the owner of the "suboptimal" solution to the extent that this improves some HL solution.

(iii) and (iv) incentivize to post open-source solutions. Otherwise, the competitor it can define a price for that solution and wait and hope that someone buys it.

(vi) incentivize people to maintain integrity on the blockchain.

Reputation Schema. Reputation is the sum of all our actions that is reflected by the people around us in the way they treat us or interact with us. It is an indirect result of anything and everything that we do. The basic idea behind reputation is somewhat similar to the idea of reward and punishment. In this paragraph we briefly describe the reputation schema used to assign reputation to the entities on the platform that allow them to become more known and be considered more reliable.

We introduce the concept of *transitive reputation* which is very close to the transitive credit wherein a credit map for product A, which is used by product B, feeds into the credit map for B [10]. First, we consider each open-source solution or algorithm to rely implicitly on the previous solutions for the problem or class of problems they are supposed to solve. Arguably, this is a strong assumption that overestimate the

implicit knowledge embedded in open-source solutions a competitor could exploit to create her own solution. Nevertheless, it is plausible that people rely on existing solutions to create new improved solutions. As its name, the transitive reputation plans to reward users in a transitive fashion, i.e. when someone posts a solution for a specific problem, all owner of the solutions lower than the new posted one gain an increment of their reputation proportionally to the quality of their solutions. For example, if a competitor posts a solution A which is better than a solution B with regard to some quality measure *f*, i.e. *f(A)>f(B)*, which in turn is better than a previous solution C, i.e., *f(B)>f(C)*, than owner of solution B would gain a reputation given by the reciprocal of the difference *f(A)-f(B)*, while the owner of solution C will gain a reputation given by the reciprocal of the difference *f(B)-f(C)*. Note: only owner of open-source solutions are rewarded by reputation, unless the competitor posting a solution has bought a private solution in the past, within the context of the specific problem.

In the context of private solutions, the reputation schema behaves differently. As by the pay-per-use schema, the owner of a private solution will gain a positive amount of reputation r each time her solution will be used (i.e. bought as-a-service) by others.

The primary value of transitive reputation is in measuring the indirect contributions to a solution, which today are not quantitatively captured. Because they are not captured, they are not rewarded, and there is a disincentive to perform them. If they were captured, this disincentive would be replaced by an incentive, which in this case means to publish and share solutions and algorithm in a reusable form. Other reputational mechanisms can apply [17].

2.4. Attacks aiming at exploiting the reward mechanism

We now analyze two possible ways a malicious user can try to exploit the rewarding mechanism to gain more and propose solutions to prevent this behavior, which themselves are embedded in the design.

Gradual introduction of a suboptimal solution. If a competitor finds a solution which is *n* times better than the previous one, with respect to some quality measure, the competitor could be incentivized not to reveal it, but to design worse variants of it, so as to publish them gradually and reap the reward each time, since she's introducing different "best" or suboptimal (mediocre) solutions. Although it could be considered a (apparent) fault of the rewarding mechanism exploitable by a malicious user, it is not, because the user is still introducing better solutions usable by other people, which are incentivized by the design we propose. Furthermore, we argue that the versions of an improved solution the user can introduce are a finite number, and it does not affect too much the behavior of the mechanism.

Plagiarism Countermeasures. The Oxford dictionary defines plagiarism[5] as *"The practice of taking someone else's work or ideas and passing them off as one's own"*.

By design it is possible for a peer to replicate an open-source solution, obfuscate it and reintroduce it into the database as private, to obtain reputation, or as public, to obtain the reward for having shared it with others. Although it seems a problem or a possible attack at first glance, rule (ii) in section 2.3 prevents this behavior. As the rule states: *"If at the time of posting the solution there already exist a solution with the same value, the previous schema does not apply and the competitor must pay the amount of tokens y"* it is not worth to replicate an existing solution since the rewarding rules establish a *first come, first served* policy on equal solutions' value, i.e., at the same solution's value, only the first people to improve a previous solution for a specific problem will receive the reward. Thus, the plagiarist will obtain no reward or reputation. Moreover, she must pay to posts the solution. However, the plagiarist could use the solution and post a new improved one based on it. In this case the problem does not arise, since she is introducing a new improved solution, which is rightly and properly incentivized by our design.

There still exists the opportunity that the plagiarist would introduce the plagiarized solution as private to gain from its sales. Since an open-source solution with the same value would already exists, people would be disincentivized to buy solution by the plagiarist.

2.5. Gamified, Blockchain-based Marketplaces: Workflow

The workflow of operations allowed across the marketplace is as follows:

a) A proponent proposes a problem, providing its representation and the representation of the candidate solution together with the objective function, used to evaluate the quality of potential solutions, and a validation algorithm used to automatically validate the solutions, i.e., to verify they are feasible solutions for the problem. The representation is important to shaping the nature of the search problem and it should be one can be rapidly understood. To ensure that all is working, the proponent can eventually provide a simple algorithm (e.g. a random search) that allows competitors to check that the objective function is working correctly and that can be used by them to resolve the problem (e.g., running it several times, with different paramenters). The problem is replicated in

[5] https://en.oxforddictionaries.com/definition/plagiarism

order to store it in the entire distributed database to make it consistent.

b) Competitors download the problem and (eventually) the algorithm provided by the proponent and try to resolve it. They can set up their own algorithms, or use existing ones, regardless of the type (i.e., exact methods, heuristics, metaheuristics etc.), store them into the database and share the obtained solutions on the marketplace.

c) Competitors post their solutions: (i) the one who posts the best solution for the problem up to that moment get rewarded; (ii) those who post solutions as open-source – regardless whether they are the best ones – get rewarded for having shared them with everyone. The reward is bounded by the gamified rewarding rules in section 2.3; (iii) those who want to post solutions with the intent of selling it – and are not the best ones for the problem up to that moment – must pay an amount of tokens to do it.

d) When nodes perform actions on the network or on the database, e.g., they post, sell or buy a solution or transfer their ownership, those are validated by other peers to guarantee the integrity and the ownership of each operation.

Figure 2 represents the ideal workflow previously outlined using a simple UML sequence diagram. The diagram outlines the scenario in which a proponent posts a problem and a competitor resolves it and posts the solution as open-source (or the solution result is the best one for the problem). If the solution is not the best one for the problem and it is not open-source, arrow at step 7 must be reversed and labelled "pay to public solution as private".

3. Example: Competing for the Knapsack Problem

The Knapsack Problem (KP) is a well-known NP-hard problem: given a set of items, each with a weight and a value, determine which item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography, applied mathematics, and daily fantasy sports. In this section we show an example of the application of Genetic Algorithm (GA) to the Knapsack Problem [6] and how competitors can take advantage from the properties of techniques such as GA and compete to find the "best so far" solution for such a problem.

Genetic Algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). GAs are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. During the course of evolution, natural populations evolve according to the principles of natural selection and "survival of the fittest". Individuals which are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individuals which are less fit will be eliminated. This means that the genesfrom the highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted ancestors may produce even more fit offspring. In this way, species evolve to become more and more well adapted to their environment. A GA simulates these processes by taking an initial population of individuals and applying genetic operators in each reproduction. In optimisation terms, each individual in the population is encoded into a string or chromosome which represents a possible solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or solutions are given opportunities to reproduce by exchanging pieces of their genetic information, in a crossover procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e., children), which share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (generational approach) or replace less fit individuals (steady-state approach). The basic steps of a simple GA are: (1) generate the initial solution; (2) valuate fitness of individuals in the population; (3) select parents from population; (4) recombine parents to produce children; (5) evaluate fitness of the children; (6) replace some or all of the population by the children. The evaluation-selection-reproduction cycle (steps 3 to 6) is repeated until a satisfactory solution is found or a stop criterion is reached.

3.1. Problem representation and objective function

The one-dimensional Knapsack Problem can be formulated as follows:

$$maximise \sum_{i=1}^{n} c_i x_i \tag{1}$$

$$subject\ to \sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

wherein n is the number of items; c_i and w_i are the value and the weight of the i-th item, respectively; W is the maximum capacity; and x is a integer variable that indicates the possibility that an item is included

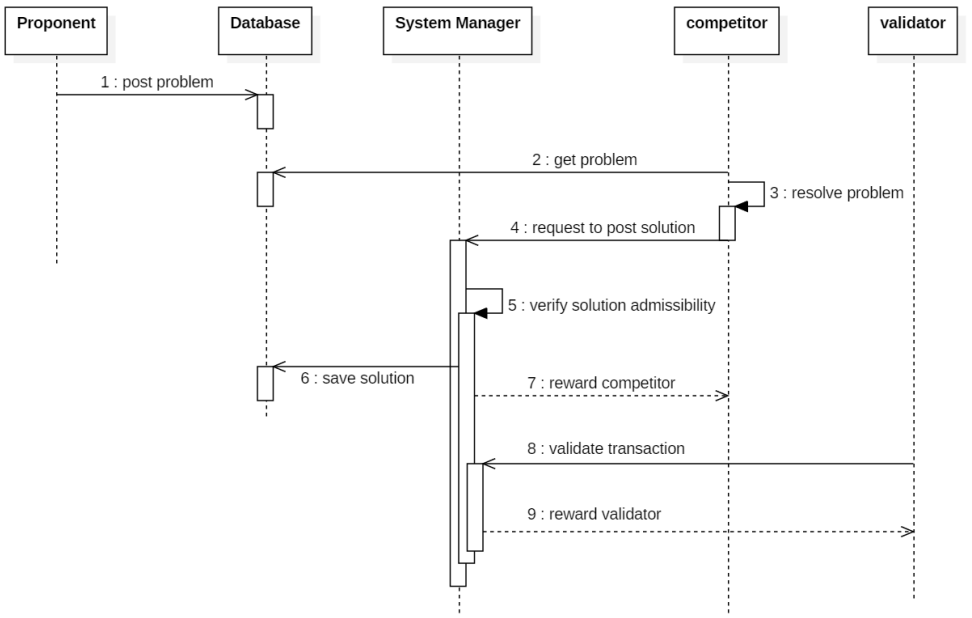

Figure 2. Gamified blockchain-based marketplaces: a workflow – notation loosely based on UML sequence diagrams.

i	1	2	3	4	5	…	*n*-1	*n*
S[i]	0	1	0	0	1	…	0	1

Figure 3. Binary representation of a KP solution.

in the collection. In this example we consider the 0-1 Knapsack problem, therefore $x_i \in \{0, 1\}$ is a binary value, meaning that each item can be included zero or one times in the collection.

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme, i.e., a way to represent individuals in the GA population. The standard GA 0-1 binary representation is an obvious choice for the one-dimensional Knapsack Problem since it represents the underlying 0-1 integer variables. Hence, in this representation, we used a n-bit binary string, where n is the number of variables in the KP. A value of 0 or 1 at the i-th bit implies that $x_i = 0$ or 1 in the solution, respectively. This binary representation of an individual's chromosome (solution) for the KP is illustrated in Figure 3. Note that a bit string $S \in \{0, 1\}^n$ might represent an infeasible solution. An infeasible solution is one for which the constraint (2) is violated, i.e., $\sum_{i=1}^{n} w_i x_i > W$.

3.2. Initial Population

To achieve sufficient diversification one can randomly generate an initial population with a high size being fixed (e.g. $n = 100$) and construct each of the initial feasible solutions by a primitive primal heuristic that repeatedly randomly selects a variable and sets it to one if the solution is feasible. Another competitor can generate the initial population changing its size and using a simple constructive greedy or a clever heuristic to construct initial solutions rather than randomly generate them. Competitors can initialize their population as they want as long as the final posted solution is feasible for the problem.

3.3. Parent Selection

Parent selection is the task of assigning reproductive opportunities to each individual in the population. Typically in a GA we need to generate two parents who will have (one or more) children. The tournament selection method works by forming two pools of individuals, each consisting of t individuals drawn from the population randomly. Using a larger value for t has the effect of increasing selection pressure on the more fit individuals. A competitor can adopt the standard binary tournament selection method (i.e., $t = 2$) as the method for parent selection because it can be implemented very efficiently. Another competitor can base its selection on other criteria such as Roulette Wheel Selection where the probability for an individual to be selected is proportioned to its fintess; Rank Selection; Random Selection, etc.

3.4. Crossover and Mutation

The binary, problem-independent, representation adopted for the KP in this example allows a wide range of the standard GA crossover and mutation operators

to be adopted. Competitors can choose among several crossover operators. In a *one-point crossover*, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs. *Multi-point crossover* is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs. In *uniform crossover* the chromosome is not divided into segments, rather it treats each gene separately. Two parents have a single child. Each bit in the child solution is created by copying the corresponding bit from one or the other parent, chosen according to a binary random number generator [0, 1]. If the random number is a 0, the bit is copied from the first parent, if it is a 1, the bit is copied from the second parent. There exist a lot of other crossovers like Partially Mapped Crossover (PMX), Order based crossover (OX2), Shuffle Crossover, Ring Crossover, etc. Once a child solution has been generated through crossover, a mutation procedure is performed that mutates some randomly selected bits in the child solution, i.e., causes these chosen bits to change from 0 to 1 or vice versa. The rate of mutation is generally set to be a small value (in the order of 1 or 2 bits per string). However, each competitor can define the mutation value she desires.

3.5. Repair operator

Clearly, the child solution generated by the crossover and mutation procedures may not be feasible because the Knapsack constraints may not all be satisfied. In order to guarantee feasibility, competitors can apply several heuristics, such as a simple greedy algorithm.

3.6. Stopping Criterion

The following three kinds of termination conditions have been traditionally employed for GAs [16]: (i) an upper limit on the number of generations is reached; (ii) an upper limit on the number of evaluations of the fitness function is reached; or (iii) the chance of achieving significant changes in the next generations is excessively low. However, there are a lot of other criteria defined in the literature, all with their pros and cons. Competitors can choose among them or define their own stopping criteria. A simple stopping criterion could be based simply on the execution time a competitor is willing to spent.

3.7. Algorithmic outline

Settings of the GA heuristic a competitor can use for the KP are:

- the binary tournament selection method;
- the uniform crossover operator;
- a mutation rate equal to 2 bits per child string;
- to discard any duplicate children (i.e., discard any child which is the same as a member of the population);
- the steady-state replacement method based on eliminating the individual with the lowest fitness value.

This settings can be included as solution information when a competitor posts the solution generated by this algorithm set up. Of course, competitors can use simpler heuristics than GAs and also obtain better results, depending on the problem instance and search technique used.

Let's consider the data in Table 1 and a weight limit of 20. In this case there exist two optimal solutions that can be found with different algorithms. The optimal solutions consist of selecting the 2nd and 7th item or the 7th and 8th item, and are generated following the binary representation showed before: (0,1,0,0,0,0,1,0) and (0,0,0,0,0,0,1,1) both with a weight of 19 and a total value of 8. The solutions are then evaluated by the validation algorithm provided by the proponent, with the purpose to check their feasibility and execute the objective function to measure their quality. Once accepted the system provides to insert them into the database, together with the algorithm information, and register them on the blockchain whose transactions are validated by other peers to guarantee the integrity and the ownership of those operations as mentioned in section 2.5. At this point, people on the network can buy, sell or exchange those solutions and the algorithm set up through smart contracts that establish and guarantee the conditions to be respected for their use, payment and their eventual ownership transfer. For example, competitors could have generated intermediate solutions such as (0,1,1,0,0,0,0,0) or (0,0,0,0,0,1,1,0) of value 7 and weight 18 and 20, respectively. Those solutions could be acquired, through smart contracts, by other peers to solve similar problems or, for example, to use them as part of the initial population in a evolutionary algorithm. Finally, instead of competing, competitors could also cooperate starting from a initial population and share computational efforts to create new generations (i.e., children) in a distributed fashion and split the rewards. Smart contracts guarantee that each competitor will be rewarded by the effort spent to find the final solution and all the information will be stored on the blockchain.

4. Discussion

The problems we want people to solve on our marketplace often apply in areas where there is no

item	1	2	3	4	5	6	7	8
value	5	3	4	6	3	1	5	3
weight	15	8	10	14	11	9	11	8

Table 1. Example of data for the KP.

existing best solution. This is particularly true in the engineering. It is natural in the resolution of new problems of engineering fields that there are, at first, moments characterized by enthusiasm, early important results and the excitement that goes with them. However, this excitement is often struck down by the fact that no optimum solution is known, and the actual results could be not the best ones. Clearly, this is the case of search-based optimization problems, that involve search techniques to find the best solution in a huge space of possible solutions. There are three key ingredients for the application of search-based optimization to a widely number of applications: (i) the choice of the representation of the problem; (ii) the definition of the objective function; and (iii) a set of manipulation operators. Typically, a proponent will have a suitable representation for her problem, so the first of the pre-requisites is easily satisfied. Even the objective function is defined by the proponent, which is posted together with the problem representation and its description. However, the objective function can be proposed by competitors and eventually accepted by the proponent, for example through a stacking mechanism. Competitors, for their parts, can define manipulation operators or using existing ones. Different search techniques use different operators. As a minimum requirement, it will be necessary to mutate an individual representation of a candidate solution to produce a representation of a different candidate solution. It will make it possible to apply hill climbing approaches and certain forms of evolutionary computation. If it is possible to determine the set of "near neighbors" of a candidate solution (in term of its representation) then simulated annealing and tabu search can be applied. If, instead (or in addition), it is possible to sensibly cross-over two individuals (to produce a "child" which retain characteristics of both "parents") then genetic algorithms will be applicable. With these three ingredients it becomes possible to implement search algorithms. The results of the search algorithms can be compared, for example using random search or other algorithms provided by the competitors or by the proponent itself, to provide as baseline data. Naturally, the aim is to beat them, though in some areas even a purely simple, unsophisticated algorithm, such as random search, has been found to be not without value, even beating human-directed search in some cases. This fact is supported by the "no free lunch" (NFL) theorems that establish that for any algorithms any elevated performance over one class of problems is exactly paid for in performance over another class [20]. Of course, the goal is to find the best solution for a given problem, regardless from the used algorithm. However, having different algorithms to compare – and solutions generated by these algorithms – allows researchers and people to understand and balance the results with performance, with the possibility to create an ontology of problems and related algorithms together with their best solutions and information.

There exist a lot of search-based optimization algorithms and techniques. Although competitors can use precise optimization algorithms such as linear programming[6] to solve problems – and we encourage them to do so, whenever possible –, those are straightforward deterministic algorithms. Even though modern solvers can deal with thousand of variable and millions of clauses, these deterministic optimization algorithms are often inapplicable because the problems have objectives that cannot be characterized by a set of linear equations. Often there are multiple criteria and complex objective functions. Many of the optimization problems are augmented versions of known NP-complete problems and, as such, they are well suited to the application of metaheuristic search techniques. A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines of strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework [18]. A metaheuristic is not an algorithm. Rather, it is a consistent set of ideas, concepts, and *operators* that can be used to design heuristic optimization algorithms. This acknowledgment brings with it all of issues that are associated with the application of metaheuristic search techniques: (i) global optimum; (ii) predictability; and (iii) computational expense.

Global Optimum. There is no guarantee that the global optimum will be found. In many applications, there is a threshold, above which a solution will be "good enough" for purpose. Furthermore, optimization may not seek to find an optimal solution to a problem, but rather, it may seek to improve upon the current situation.

Predictability. Each execution will potentially yield different results. It is true that each execution of a metaheuristic search algorithm can yield different results, but all search algorithms are formulated in such a way that repeated executions can only improve on a

[6]Linear programming is a mathematical optimization technique that is guaranteed to locate the global optimum solution subject to some linear expression in the decision variables given as input to the linear programming model.

"best so far" result, rather than overturning a previous result. The algorithms may be terminated at any time and also after any number of executions to yield results which are the "best so far".

Computational Expense. Many individual candidate solutions may need to be considered before an acceptable quality solution is found. The kinds of problems to which search techniques seem to be readily applicable, are those where the solution is highly complex. While speedy answers may be attractive, they are not essential in many applications. To overcome computational expense, competitors can solve problems in a distributed fashion, sharing their computational power and split the rewards, as nowadays happen with the famous "mining pools". Several techniques can be used that involve parallelism (e.g., Multiple Threads, Island Models, Master-Slave Fitness Assessment) [12] and different reward approaches exist (e.g., slush's pool, pay-per-share, p2pools, etc.).

An attractive field to which search-based optimization techniques have been widely applied is software engineering [8, 9]. Many of the problems faced by software engineers turn out to have natural counterparts as "standard" optimization problems. Indeed, from its formal definition to this date a huge number of software engineering problems have been mathematically formulated as optimization problems and tackled with a considerable variety of search-based techniques. Often, of course, there are some modifications and enhancements that are required and suitable representations and objective functions must be formulated for each problem; therein lies interesting and exciting research.

5. Conclusions and Future Work

The role of blockchain technology to structure effective, trust-based and cooperative software engineering solutions is not clear yet but offers ample potential and great opportunity. The solution we outlined in the previous pages combines state of the art blockchain technologies in a new way and includes in the midst all the elements of gamification in a purposeful way such that a business-savvy software-based solution can be structured. Although a proof-of-concept of our proposal is currently under way of prototyping, we are looking to formalize our proposal using more structured approaches to software and requirements engineering, properly exploring the solution space defined by our proposed design pattern and understanding its factual feasibility beyond the theoretical illustration and discussion we currently offered. In the future we plan to carry out such formalization, addressing in particular the business benefits behind the solution possibly by means of industrial empirical software engineering research. From a technical perspective, we aim at eliciting the different technical implementation options currently existing to support blockchains (e.g., Ethereum, Hyperledger) and comparatively analyze their fitness for purpose towards designing for, implementing, and operating prototypes for the proposed idea. We plan to propose a software-centric business model behind the proposed solution such that the proposed design pattern may be accompanied by a sound and well-thought business plan template around which companies and practitioners interested in blockchain technology may base their own solutions.

Finally, although this framework promotes sharing open-source solutions, others rules can apply. We also aim at creating a platform that allows proponents to create contests and define their own rules, in accordance with the design pattern presented, defining a distributed ledger model that marks the progress of problem solving as intellectual capital.

Acknowledgements

The authors are grateful to Hervé Gallaire for many useful feedbacks and insights for the previous versions of this paper.

References

[1] Bell, R.M. and Koren, Y. (2007) Lessons from the netflix prize challenge. *SIGKDD Explorations 9* **2**: 75–79.

[2] Bengtsson, M. and Kock, S. (2000) âĂİ coopetitionâĂİ in business networksâĂŤto cooperate and compete simultaneously. *Industrial marketing management* **29**(5): 411–426.

[3] Burke, E.K., Curtois, T., Kendall, G., Hyde, M., Ochoa, G. and Vazquez-Rodriguez, J.A. (2009) Towards the decathlon challenge of search heuristics. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* (ACM): 2205–2208.

[4] Buterin, V. *et al.* (2014) A next-generation smart contract and decentralized application platform. *white paper* .

[5] Caddy, T. (2005) Tamper resistance. *In Encyclopedia of Cryptography and Security, edited by Henk C. A. van Tilborg: Springer* .

[6] Chu, P.C. and Beasley, J.E. (1998) A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics* **4**(1): 63–86.

[7] Dalpiaz, Fabiano; Ali, R. and Brinkkemper, S. (2018) Special section on gamification and software engineering. *Information & Software Technology* **95**: 177–178.

[8] Harman, M. and Jones, B.F. (2001) Search-based software engineering. *Information and software Technology* **43**(14): 833–839.

[9] Harman, M., Mansouri, S.A. and Zhang, Y. (2012) Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* **45**(1): 11.

[10] Katz, D. (2014) Transitive credit as a means to address social and technological concerns stemming

from citation and attribution of digital products. *Journal of Open Research Software* **2.1**.

[11] LaToza, T.D. and van der Hoek, A. (2016) Crowdsourcing in software engineering: Models, motivations, and challenges. *IEEE Software 33* **1**: 74–80.

[12] Luke, S. (2013) *Essentials of Metaheuristics* (Lulu), 2nd ed., 99–107. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[13] Morschheuser, B., Hamari, J. and Koivisto, J. (2016) Gamification in crowdsourcing: A review .

[14] Nakamoto, S. (2008) Bitcoin: A peer-to-peer electronic cash system .

[15] Olague, G. (2006) Gecco-2006 highlights. human competitive awards the humies. synthesis of interest point detectors through genetic programming. *SIGEVOlution 1* **3**: 28–29.

[16] Safe, M., Carballido, J., Ponzoni, I. and Brignole, N. (2004) On stopping criteria for genetic algorithms. In *Brazilian Symposium on Artificial Intelligence* (Springer): 405–413.

[17] Scalabrino, S., Geremia, S., Pareschi, R., Bogetti, M. and Oliveto, R. (2018) Freelancing in the economy 4.0: How information technology can (really) help. *Social Media for Knowledge Management Application in Modern Organizations* : 290–314.

[18] Sörensen, K. and Glover, F.W. (2013) Metaheuristics. In *Encyclopedia of operations research and management science* (Springer), 960–970.

[19] Tamburri, D.A., Lago, P. and van Vliet, H. (2013) Uncovering latent social communities in software development. *IEEE Software 30* **1**: 29–36.

[20] Wolpert, D.H. and Macready, W.G. (1997) No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* **1**(1): 67–82.

[21] Zheng, Z., Xie, S., Dai, H., Chen, X. and Wang, H. (2017) An overview of blockchain technology: Architecture, consensus, and future trends. *Big Data (BigData Congress), 2017 IEEE International Congress on* : 557–564.