Modeling Bitcoin Protocols with Probabilistic Logic Programming

Damiano Azzolini[1], Fabrizio Riguzzi[2], Evelina Lamma[1], Elena Bellodi[2], and Riccardo Zese[1]

[1] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
damiano.azzolini@student.unife.it,
[fabrizio.riguzzi,evelina.lamma,elena.bellodi,riccardo.zese]@unife.it

Abstract. Bitcoin is one of the first decentralized, peer to peer, payment systems based on the so-called Proof-of-Work (PoW). PoW is an algorithm that requires the computation of a hard function in order to gain access to a resource but, at the same time, the correctness of the computed result should be easily checked. The use of a PoW removes the necessity of a centralized third party and so the consistency of the network may be altered directly by the involved users. Peers, to solve the PoW more efficiently, usually organize themselves into mining pools, to increase the overall computational power: this situation, unfortunately, leads to a network centralization. In this paper we consider two typical scenarios of a Bitcoin network and we model them by probabilistic logic programming (PLP): the centralization of the hashing power by large pools and the "double spending attack". In the first one, we verify the effectiveness of a protocol that attempts to discourage the formation of large pools. In the second one, we compute the probability of success of an attacker. Both scenarios are modeled using the PLP package *cplint*.

Keywords: Bitcoin, Blockchain, Probabilistic Logic Programming.

1 Introduction

Bitcoin is a payment system based on a decentralized protocol that has captured the attention of many users and developers. The key of Bitcoin (and more in general, blockchain) success is the possibility to create new currencies and new methods of payment without the need of a centralized trusted third party. Trust between peers is obtained through a process called Proof-of-Work. Involved peers usually group themselves in pools in order to reduce the variability of rewards. Pool formation, on the other hand, decreases the decentralization of the system and centralizes the computational power in the hands of only few miners (currently, in Bitcoin network, the first three pools hold more than 50% of the total hash power and the first 10 more than 90% [15]). This situation can eventually

allow few people to control the whole network and lead to the so-called 51% or majority attack. In this paper we analyze a protocol proposed in [8] to tackle hashing power centralization, and we evaluate the probability of a double spending attack. The analysis is based on an encoding of the models by Probabilistic Logic Programming. Results obtained are similar to the ones described in [13] and [20]. A similar scenario is analyzed both in [11] and [14].

The paper is organized as follows: in Section 2 we introduce Probabilistic Logic Programming. In Section 3 we briefly present the Bitcoin protocol. In Section 4 we analyze a solution to large pools formation proposed in [8] and model the probability of a double spending attack. Section 5 concludes the paper.

2 Probabilistic Logic Programming

We consider Probabilistic Logic Programming (PLP) languages under the distribution semantics [21], that have been shown expressive enough to represent a wide variety of domains [2,19,1]. A program in a language adopting the distribution semantics defines a probability distribution over normal logic programs called *instances* or *worlds*. Each normal program is assumed to have a total well-founded model [23]. Then, the distribution is extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs.

A PLP language under the distribution semantics with a general syntax is that of *Logic Programs with Annotated Disjunctions* (LPADs) [24]. We present here the semantics of LPADs for the case of no function symbols, if function symbols are allowed see [18].

In LPADs, heads of clauses are disjunctions in which each atom is annotated with a probability. Let us consider an LPAD T with n clauses: $T = \{C_1, \ldots, C_n\}$. Each clause C_i takes the form: $h_{i1} : \Pi_{i1}; \ldots; h_{iv_i} : \Pi_{iv_i} :- b_{i1}, \ldots, b_{iu_i}$, where $h_{i1}, \ldots, h_{iv_i}$ are logical atoms, $b_{i1}, \ldots, b_{iu_i}$ are logical literals and $\Pi_{i1}, \ldots, \Pi_{iv_i}$ are real numbers in the interval $[0, 1]$ that sum to 1. $b_{i1}, \ldots, b_{iu_i}$ is indicated with $body(C_i)$. Note that if $v_i = 1$ the clause corresponds to a non-disjunctive clause. We also allow clauses where $\sum_{k=1}^{v_i} \Pi_{ik} < 1$: in this case the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{v_i} \Pi_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

Each grounding $C_i\theta_j$ of a clause C_i corresponds to a random variable X_{ij} with values $\{1, \ldots, v_i\}$ where v_i is the number of head atoms of C_i. The random variables X_{ij} are independent of each other. An *atomic choice* [16] is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, v_i\}$ identifies one of the head atoms. In practice (C_i, θ_j, k) corresponds to an assignment $X_{ij} = k$.

A *selection* σ is a set of atomic choices that contains an atomic choice (C_i, θ_j, k) for each clause $C_i\theta_j$ in $ground(T)$. Let us indicate with S_T the set of all selections. A selection σ identifies a normal logic program l_σ defined as $l_\sigma = \{(h_{ik} :- body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. l_σ is called an *instance*, *possible world*

or simply *world* of T. Since the random variables associated to ground clauses are independent, we can assign a probability to instances $P(l_\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$.

We consider only *sound* LPADs where, for each selection σ in S_T, the well-founded model of the program l_σ chosen by σ is two-valued. We write $l_\sigma \models q$ to mean that the query q is true in the well-founded model of the program l_σ. Since the well-founded model of each world is two-valued, q can only be true or false in l_σ.

We denote the set of all instances by L_T. Let $P(L_T)$ be the distribution over instances. The probability of a query q given an instance l is $P(q|l) = 1$ if $l \models q$ and 0 otherwise. The probability of a query q is given by

$$P(q) = \sum_{l\in L_T} P(q,l) = \sum_{l\in L_T} P(q|l)P(l) = \sum_{l\in L_T : l\models q} P(l) \tag{1}$$

Example 1. Let us consider the UW-CSE domain [12] where one of the objectives is to predict the "advised by" relation between students and professors. In this case a program for $advisedby/2$ may be

$$\begin{aligned} &advisedby(A,B) : 0.3 :- \\ &\quad student(A), professor(B), project(C,A), project(C,B). \\ &advisedby(A,B) : 0.6 :- \\ &\quad student(A), professor(B), ta(C,A), taughtby(C,B). \end{aligned}$$

where $project(C,A)$ means that C is a project with participant A, $ta(C,A)$ means that A is a teaching assistant (TA) for course C and $taughtby(C,B)$ means that course C is taught by B. The probability that a student is advised by a professor depends on the number of joint projects and the number of courses the professor teaches where the student is a TA: the higher these numbers the higher the probability.

Suppose we want to compute the probability of $q = advisedby(harry, ben)$ where *harry* is a student, *ben* is a professor, they have one joint project and *ben* teaches one course where *harry* is a TA. Then the first clause has one grounding with head q where the body is true and the second clause has one grounding with head q where the body is true. The worlds where q is true are those that contain at least one of these ground clauses independently of the presence of other groundings, so $P(advisedby(harry, ben)) = 0.3 \cdot 0.6 + 0.3 \cdot 0.4 + 0.7 \cdot 0.6 = 0.72$.

3 Bitcoin

Bitcoin is one of the first decentralized, peer to peer, payment systems based on Proof-of-Work (PoW). It was proposed in 2008 by Satoshi Nakamoto [13]. Due to its decentralization, there is no need to have a trusted centralized third party. Trust between peers is obtained with a distributed public ledger, called *blockchain*: each peer in the network has a copy of this ledger and can see all the transactions that took place since the beginning of the network.

Peers can send each other coins, issuing a *transaction*. A transaction consists of one or multiple inputs (which are references to outputs of previous transactions) and one or multiple outputs (specifying the address of the receiver, the value that needs to be transferred and the fees for the miner). Once the transaction is set, if it is valid, it will be moved into a pool of unconfirmed transactions.

To be confirmed, the transaction needs to be included into a block and then stored into the blockchain. Adding a block to the chain can be done by solving a cryptographic puzzle, the so called Proof-of-Work. This process is best known as *mining*. PoW consists in finding a value called nonce (that can be changed arbitrarily during this brute-force process) such that, when appended to the current block header (composed of meta-data), SHA256(SHA256(blockHeader)) $\leq T$ where T is a value called *target* and SHA is the hash function "Secure Hash Algorithm" [4,17]. The *target* value is dynamically changed every 2016 blocks found to ensure that new blocks will be generated at regular intervals, one every 10 minutes on average. The goal is to produce exactly 2016 blocks in two weeks: if this value is reached in a different time, every Bitcoin client compares the actual time it took to generate these blocks with the two weeks goal and modifies the target accordingly, making PoW more or less difficult [22], i.e., if the goal has been reached in less than two weeks, the target value will decrease and so mining will become more difficult and vice versa. If a peer is able to find the correct hash, it is rewarded with a certain number of bitcoins plus the fees of the transactions included in the blocks. The number of bitcoins which can be mined is limited to 21 millions so, once all available bitcoins will be mined, transaction fees will become the only mining income.

Once the correct hash is found, the block is added to the chain. All the blocks are linked together by adding the hash of the last validated block inside the header of the block which has to be inserted into the chain. The chain can have different branches, and the miners add the block to the longest branch. Blockchain is spread over a huge network, so two valid blocks can be broadcast at the same time but be received in different order. In this case, peers will continue to build the chain on top of the received block (i.e. someone will build upon the first block, someone else on the second one). This double branch is deleted when a new block is broadcast: the new block will have one of the two previous blocks as ancestor and so will make one of the two branches longer than the other one. This behaviour suggests that each participant receiving bitcoins in a transaction waits for some blocks (number of confirmations) before considering the transaction fully valid. Generally, a transaction remains unconfirmed until it has at least 6 blocks over it, since another chain could be the valid one. Even if 6 is the suggested number to wait for the authors of [5], according to [6] a recent attack on Bitcoin Gold (one of the many forks of Bitcoin) has reverted 22 blocks. This scenario opens up several possibilities for attack. One of the best known and studied attacks is the so called "double spending attack" [7], which will be presented in detail in Section 4.2.

4 Protocol Models

4.1 Preventing Large Pools Formation

PoW is a hard problem and the probability of success (i.e. finding the correct hash of a block) is proportional to the hash rate owned by each miner. Nowadays, the probability that a single miner can find a correct block is close to zero [10]. To increase the probability of success, miners usually organize themselves into a mining pool, to work together on each block and share the revenues in case of success. The pool is typically maintained by a pool operator who may take a fee, such as a fixed percentage of the block reward, for his services. The formation of large mining pools, on the other hand, can generate a super pool that holds more than 50% of the total hash power. This situation happened in July 2014 when *GHash.io* held more than 51% of the hashing power [9] and therefore was theoretically capable of performing a successful double spending attack publishing fraudulent transactions.

To prevent the centralization of the hashing power, the authors in [8] and [3] proposed a 2-phase PoW protocol not yet adopted in practice. This new method consists of a PoW organized in 2 steps: the first step is to find a double hash of the header SHA256(SHA256(header)) that is smaller than a certain value X (this is the current PoW). The second one consists in signing the header with the private key of the address that will receive the mining reward and then finding a new hash SHA256(SIG(header,privateKey)) smaller than a value Y. The ratio between X and Y is crucial to avoid the creation of large pools but, at the same time, let small pools exist (and incentivize the participation of new miners). Unfortunately, the original paper does not show how to choose the correct rate between these two values. This solution disincentivizes the formation of large pools because the pool operator should share his private key with all the components of the pool, in order to complete the second phase. Doing that, he allows every peer to access and potentially steal the newly mined coins. Therefore, the pool operator will share his private key only to trusted peers and not to everyone who wants to join the pool, reducing the probability of large pool formation.

Considering the above-described situation, a peer can be in one of the following 4 states:

- *State 0 (a0)*: a miner generates a hash using a certain nonce. If this hash is correct, it will move to state 1, if it's not, it will stay in *a0*.
- *State 1 (a1)*: the miner has already solved the first hash puzzle (he has found a nonce value such that SHA256(SHA256(header)) is less than a difficulty parameter X) and now needs to solve the second one (find a hash value such that SHA256(SIG(header,privateKey)) is less than a difficulty parameter Y).
- *State 2 (a2)*: the miner has solved both hashes, is rewarded and now is ready to mine another block (back to state 0).
- *State 3 (a3)*: another miner has solved the second hash, so the first miner can now stop working on this hash and move to another one (back to state 0).

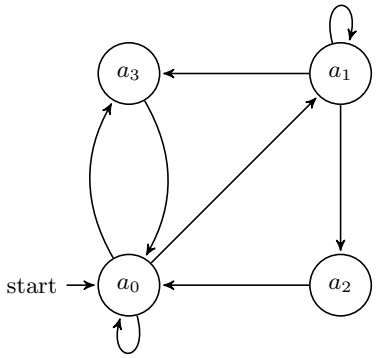

Fig. 1. Markov chain of the Model.

This scenario can be modeled using the Markov chain shown in Figure 1. The chain can be represented using the *cplint* package for Probabilistic Logic Programming [1].

In this example, we consider both X and Y having a finite value (in [8] and [3], X has a finite value while Y is infinite, i.e. each hash is good). The following LPAD models a race between two peers, a and b:

```
a_found_y(_):0.15.
b_found_y(_):0.25.
b_found_x(_):0.10.
found_y(S):- a_found_y(S); b_found_y(S).
```

As an example, we suppose that a will find the right Y hash with 15% probability and b with 25% probability. `found_y/1` is true when a or b have found the correct Y hash at state S.

```
trans(a0,S,a1):1.0/50; trans(a0,S,a0):1.0-1.0/50:- \+b_found_x(S).

trans(a1,S,a2):0.15;trans(a1,S,a1):1.0-0.15:- \+b_found_y(S).
```

Transition between $a0$ and $a1$ will happen with probability 1/50 and only if b has not found the correct hash for the current block (in this case, there's no need to keep searching for this hash). Similarly, a will move from $a1$ to $a2$ with 15% of probability, only if b has not found the hash.

```
trans(a0,S,a3):- b_found_x(S).
trans(a1,S,a3):- b_found_y(S).

trans(a2,S,a0):- found_y(S).
trans(a3,S,a0):- found_y(S).
```

In case b has found X or Y, a will reach state $a3$ if it was in $a0$ or $a1$. If a reaches node $a2$ or $a3$, it means that Y has been found, respectively by a or b. Then, a will reach $a0$ to start mining a new hash. In this LPAD, we have modelled only a. Peer b can be defined in a similar way.

Finally, transitions between states can be modelled with the predicate `reach/3`, starting at state S at instant I, state T is reachable if there is a transition from state S to state U at instant I and if T is reachable from U at next instant:

```
reach(S, I, T) :-
   trans(S, I, U),
   reach(U, next(I), T).
reach(S, _, S).
```

We can approximate the probability P to reach state $a2$ from state $a0$ by sampling:

```
?- mc_sample(reach(a0,0,a2),1000,P).
P = 0.068
```

The predicate `mc_sample/3` performs approximate inference by sampling 1000 times `reach/3` and returning the estimated probability that a sample is true.

4.2 Double Spending Attack

Probabilistic Logic Programming can also be used to model the scenario of an attacker trying to double spend, i.e. trying to spend the same bitcoin twice [7]. The process goes as follows: first the attacker creates a transaction $T1$ with an address A as output.

Then secretly, he starts mining his own fork of the chain: this new chain does not include $T1$ but a new transaction $T2$ whose output is not A, but the address of the attacker himself (Fig.2).

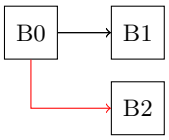

Fig. 2. Initial state of the double spending attack. Block $B1$ with transaction $T1$ is inserted in the chain after $B0$, while the attacker starts mining another block ($B2$) with $B0$ as ancestor, without $T1$ inside.

After a certain number of blocks (transaction confirmations) built on top of block $B1$, the recipient of the transaction is convinced that $T1$ is valid and can, ideally, send goods to the attacker (Fig.3).

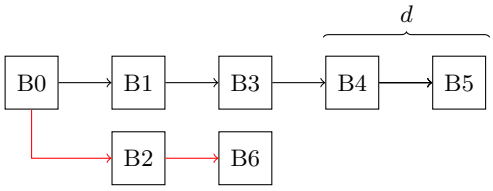

Fig. 3. General case. The "honest" chain has built 3 confirmation blocks on $B1$ ($B3, B4, B5$) while only one block ($B6$) has been built on top of $B2$ by the attacker. In this figure, d represents the distance between the honest and the secret chain and is used to evaluate the advantage of the honest chain over the attacker.

Now the attacker, to double spend, needs to make his own chain longer than the honest one. If it succeeds, he will publish all the mined blocks on his own fork. In this way, all the transactions stored in blocks between $B1$ (including transaction $T1$) and the last block of the honest chain will be invalidated (Fig.4). If this happens, if a user A sends some money to a user B in one of the invalidated transactions, B will no longer have this money and A will be able to spend it again, unbeknownst to B.

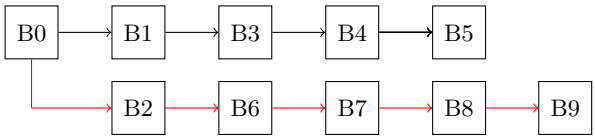

Fig. 4. Successful attack. The attacker has built a longer chain (marked in red). The attacker will now publish all blocks from $B2$ to $B9$ and so all blocks from $B1$ to $B5$ in the black chain will not be considered valid because they are part of a chain which is not the longest one.

In this model, some simplifications are made:

- The total hash rate of the whole network is constant.
- The mining difficulty is constant.
- There is only one attack going on: all the peers, except for the attacker, are mining over the honest chain.

The whole attack can be described by two functions: the attacker's potential progress function and the catch up function.

The attacker's potential progress function relates the number m of blocks that are mined by the attacker, while the honest chain has mined n blocks. In Nakamoto's paper [13], this function is described by a Poisson distribution:

$$P(m) = \frac{e^{-\lambda}\lambda^m}{m!}, \tag{2}$$

where $\lambda = nq/p$, q hash rate of the attacker, $p = 1 - q$ hash rate of the honest miners.

On the other hand, Rosenfield [20] describes this process with a binomial negative distribution (also known as Pascal distribution):

$$P(m) = \binom{m+n-1}{m} p^n q^m, \tag{3}$$

where q is the probability of success (in this case, a success represents a block found by the attacker), and $p = 1 - q$.

In both cases, even if the attacker has created a chain longer than the honest one, he cannot publish his secret chain until N confirmations are reached: the victim of the attack would be aware of the attack and so would not send him the goods. Once N confirmations are obtained, the race between the honest and the attacker's chain starts.

In this scenario, we are not interested in how long the attacker will take to build a chain longer than the honest one, but only if this will eventually happen. This process, according to [13] and [20], can be described as a binomial random walk, that can be modeled with a discrete time Markov chain.

If we define d as the difference between the length of the two chains (Fig. 3), an increase of d by 1 means that the honest chain has found a block before the attacker's chain.

We can define the probability that the attacker will catch up from Z blocks behind as P_{CZ}:

$$P_{CZ} = \begin{cases} 1 & \text{if } q \geq p \\ (q/p)^z & \text{if } q < p \end{cases} \tag{4}$$

Using PLP, this can be expressed with:

```
move(T,1):0.7; move(T,-1):0.3.

walk(InitialPosition):-
   walk(InitialPosition,0).

walk(0,_).
walk(X,T0):-
    X > 0,
    X < 1000, % threshold for not winning
    move(T0,Move),
    T1 is T0+1,
    X1 is X+Move,
    walk(X1,T1).
```

In this example, we suppose that, with 70% probability, the distance between the two chains increases by one. We model the binomial random walk described in function 4 using `walk/2`: in each step, we compute if the difference will increase or

decrease by one. If the two chains have the same length, the attack has succeeded and the secret chain can be published. Usually, due to the difference of the hash power, the probability for a successful attack is very small. To avoid an infinite loop, if the difference between the two chains is greater than 1000, we suppose that the attack has failed.

Nakamoto's [13] and Rosenfield's [20] proposals can be modeled as follows:

```
attacker_progress_poisson(X):poisson(X,Lambda).
attacker_progress_pascal(X):pascal(X,N,P).


success_poisson:-
    attacker_progress_poisson(A),
    V is NumberOfConfirmations - A,
    (   V = 0 ->
     true;
     walk(V)
    ).

success_pascal:-
    attacker_progress_pascal(A),
    V is NumberOfConfirmations - A,
    (   V = 0 ->
     true;
     walk(V)
    ).
```

with `attacker_progress_poisson/1` and `attacker_progress_pascal/1` we sample, respectively, a value from a Poisson probability distribution with parameter `Lambda` and a value from a Pascal probability distribution with parameters `N` and `P` where `N` is the number of failures and `P` the success probability. After that, we compute the difference `V` of length between the two chains (the d value from Fig. 3). If the difference is 0, the attack is successful, otherwise we start a binomial random walk with `V` as initial position.

To get the probability values, we can query, for instance:

```
?- mc_prob(success_poisson,P).
P = 0.036.

?- mc_sample(success_pascal,1000,P).
P = 0.049.
```

Predicate `mc_prob/2` returns in P the approximate probability of success according to a Poisson distribution (see formula 2), while predicate `mc_sample/3` samples 1000 times `success_pascal/0` and returns in P the probability of success according to the Pascal distribution. These values are obtained using `Lambda = 10(0.3/0.7)`, `N = 10`, `X = 0.3` and `NumberOfConfirmations = 10`.

Fig. 5 shows the success probability as a function of the number of transaction confirmations according to both probability distributions. The results are in accordance with those obtained in [13] and [20].

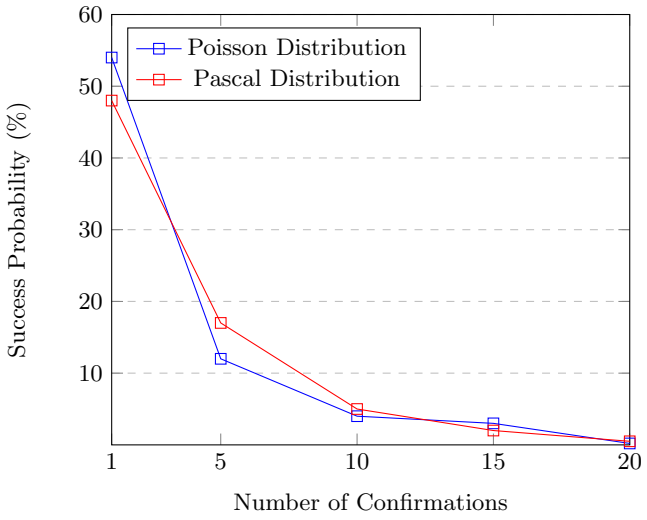

Fig. 5. Success probability of an attacker in a double spending attack according to the Poisson and Pascal probability distributions.

5 Conclusion

In this paper we have shown that Probabilistic Logic Programming can be used both to model the current Bitcoin protocol and to analyze some proposals for improvements. This analysis could be extended to blockchains in general and so, to other cryptocurrencies, such as Ethereum. The models could be also further specialized, considering a variable hash rate over time and a variable difficulty of PoW. Probabilistic Logic Programming could be also applied to model other validation types, such as Proof-of-Stake and Delegated-Proof-of-Stake, in order to compute, for instance, expected block reward or expected validation time.

References

1. Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., Zese, R.: `cplint` on SWISH: Probabilistic logical inference with a web browser. Intell. Artif. 11(1), 47–64 (2017)
2. Alberti, M., Cota, G., Riguzzi, F., Zese, R.: Probabilistic logical inference on the web. In: Adorni, G., Cagnoni, S., Gori, M., Maratea, M. (eds.) AI*IA 2016. LNCS, vol. 10037, pp. 351–363. Springer International Publishing (2016)

3. Bastiaan, M.: Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In: Available at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-astochastic-analysis-of-two-phase-proof-of-work-in-bitcoin. pdf (2015)
4. Bryson, P.J.: Secure hash standard (shs)(federal information processing standards publication 180-4) (2012)
5. Confirmation, `https://en.bitcoin.it/wiki/Confirmation`, accessed May 30, 2018
6. Double spend attack on exchanges (2018), `https://forum.bitcoingold.org/t/double-spend-attack-on-exchanges/1362`, accessed May 30, 2018
7. Double spending, `https://en.bitcoin.it/wiki/Irreversible_Transactions`, accessed May 30, 2018
8. Eyal, I., Sirer, E.G.: How to disincentivize large bitcoin mining pools. Blog post: http://hackingdistributed. com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools (2014)
9. Ghash.io, `https://en.bitcoinwiki.org/wiki/GHash.IO`, accessed May 30, 2018
10. Hash rate, `https://blockchain.info/charts/hash-rate`, accessed May 30, 2018
11. Karame, G., Androulaki, E., Capkun, S.: Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. IACR Cryptology ePrint Archive 2012(248) (2012)
12. Kok, S., Domingos, P.: Learning the structure of Markov Logic Networks. In: ICML 2005. pp. 441–448. ACM (2005)
13. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
14. Pinzón, C., Rocha, C.: Double-spend attack models with time advantange for bitcoin. Electronic Notes in Theoretical Computer Science 329, 79–103 (2016)
15. Pool distribution, `https://btc.com/stats/pool`, accessed May 30, 2018
16. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artif. Intell. 94, 7–56 (1997)
17. Proof of work, `https://en.bitcoin.it/wiki/Proof_of_work`, accessed May 30, 2018
18. Riguzzi, F.: The distribution semantics for normal programs with function symbols. Int. J. Approx. Reason. 77, 1 – 19 (October 2016)
19. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R., Cota, G.: Probabilistic logic programming on the web. Softw.-Pract. Exper. 46(10), 1381–1396 (10 2016)
20. Rosenfeld, M.: Analysis of hashrate-based double spending. arXiv preprint arXiv:1402.2009 (2014)
21. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995. pp. 715–729. MIT Press (1995)
22. Target, `https://en.bitcoin.it/wiki/Target`, accessed May 30, 2018
23. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
24. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic Programs With Annotated Disjunctions. In: ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer (2004)