Analyzing Transaction Fees with Probabilistic Logic Programming

Damiano Azzolini[1], Fabrizio Riguzzi[1], and Evelina Lamma[1]

University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy
{damiano.azzolini,fabrizio.riguzzi,evelina.lamma}@unife.it

Abstract. Fees are used in Bitcoin to prioritize transactions. Transactions with high associated fee are usually included in a block faster than those with lower fees. Users would like to pay just the minimum amount to make the transaction confirmed in the desired time. Fees are collected as a reward when transactions are included in a block so, on the other perspective, miners usually process first the most profitable transactions, i.e. the one with higher fee rate. Bitcoin is a dynamic system influenced by several variables, such as transaction arrival time and block discovery time making the prediction of the confirmation time a hard task. In this paper we use probabilistic logic programming to model how fees influence the confirmation time and how much fees affect miner's revenue.

Keywords: Bitcoin · Blockchain · Probabilistic Logic Programming.

1 Introduction

In the last year, the terms *blockchain* and *bitcoin* started to gain more and more popularity. The absence of a centralized third party, the security and the opportunity to develop new cryptocurrencies where all transactions are stored in a distributed ledger are only a few of the features of blockchain systems. Research on blockchain involves several different research areas, among them: distributed systems to maximize and improve the connections between peers, cryptography to ensure data consistency, economy to study the behaviour of the cryptocurrencies and game theory to model the interaction between interacting parties.

According to [32], blockchains have evolved over time: starting from version 1.0, where, thanks to Bitcoin [19], people were allowed to trade monetary value, Ethereum [7] extended the use cases, allowing users to define the so-called *smart contracts*. Nowadays, we are witnessing the birth of blockchains 3.0 with solutions like Lighting Network [23], that increases substantially the number of processed transactions.

Despite the availability of many different blockchains such as Ethereum [7,34], EOS.IO [11], Hyperledger [14] and Cardano [8], Bitcoin still has the highest market capitalization of all[1].

[1] https://coinmarketcap.com/

A probabilistic analysis can be particularly useful to determine how miners, peers and users interact, even when non determinism and randomization are not allowed by the various blockchain protocols. The intrinsic uncertainty of these processes requires a probabilistic analysis to be fully understood and predicted.

Probabilistic (Logic) Programming [10] has been applied to model several real world domains [20] including the Bitcoin protocol [3].

In particular starting from measures of average block size, average number of transactions in a block and average fee rate, we created two probability models: one for computing how transaction fees affect the average profit of a miner and one to analyze how fee rates in Bitcoin affect the confirmation time. For both experiments we used *likelihood weighting* to see how the observation of a certain event, such as the confirmation of a transaction with a certain fee rate or an increase of the average fee rate, modifies the probability of confirmation of the following transactions.

The paper is structured as follows: in Section 2 we give a brief overview of blockchain in general, Bitcoin and fees. Section 3 shows basic concepts of probabilistic logic programming. Section 4 explains how we conducted the experiments shown in Section 5. Section 6 concludes the paper with a discussion about the existing literature and some future works.

2 Blockchain, Bitcoin and Fees

The first idea to use cryptography to secure timestamping digital data goes back to 1991 [13]. In 2008 Satoshi Nakamoto published his paper [19] and shortly after Bitcoin and Blockchain were born. In brief, a blockchain is a sequence of blocks linked together using cryptography functions in order to guarantee data integrity and data consistency. The whole blockchain is maintained by a set of peers. All the peers can see the same data, in particular, all the blocks in the same order, thanks to a so-called *consensus* algorithm: in the case of Bitcoin, this involve solving a computationally hard problem that, once solved, can be easily checked by anyone in the network. This algorithm, called *proof-of-work*, allows also the system to function without a centralized third party. To increase the probability of success, peer usually group themselves into mining pools to share the computing power and split the revenues in case of success. Find a solution to the PoW allows the solver to append a new block to the blockchain. After that he will receive a reward in bitcoin for his work. Users in the system can send transactions, transfers of value (bitcoin) among two or more users. Each block is composed by a set of transactions. Each transaction is also attached to an amount of bitcoin as a reward for the miner who includes it into a block. One of the most interesting features is the possibility, starting from block number 0 (called *genesis block*), to reconstruct in a fully deterministic way the whole history of blocks and transactions, allowing everyone to have access to the same data.

The miner who solves the PoW receives, in addition to an amount of bitcoin, the sum of all the fees of the transactions in a block trough a special transaction called *coinbase* transaction. Thus, the miner is incentivised to include the most profitable transactions into a block. However, due to the max block size limit (1Mb, a very discussed threshold[2]), and the difficulty of the PoW puzzle (set to be solved in 10 minutes of average), transactions usually wait several minutes in the so-called *mempool*, waiting to be included in a block and confirmed.

To reduce the time spent in the mempool, users can attach a high fee to a transaction. This option, however, triggers a high competition situation where peers keep increasing the fees to prioritize transactions. On the other hand, increasing the average fees can potentially reduce the number of users, since they may be unwilling to pay such a high amount of fees for the transfer of a little amount. Users are therefore incentivised to find an equilibrium between the priority of the transaction, its size, its associated fees and the *fee rate* (amount of fees, usually measured in *satoshi*, per byte where 1 satoshi $= 10^{-8}$ bitcoin).

A common scenario which complicates the optimal fee rate estimation is the presence of dependent transactions in the same block. According to the Bitcoin consensus rule, all peers must see all blocks and all transactions in a block in the same sequential order. Moreover, bitcoin cannot be spent before being received. This means that, if A sends an amount X to B and B wants to send X to C, the transaction where A sends X to B must appear earlier in the sequence than the one used by B to send X to C. With this constraint, a miner cannot simply order the transaction in a descending fee rate value order but he needs to take into account dependencies: if B has a high associated fee rate and spends the output of A, A must be included before B even if A has a low associated fee rate. This situation con be used to force a transaction confirmation and it is known as *Child Pay for Parent*[3].

For all the previous reasons, fee estimation is a hard task. In addition, the number of transactions received by the network during a certain time span is unpredictable as well as is block discovery time.

There are several methods to estimate the optimal fee rate. One of the most used Bitcoin client, Bitcoin Core[4], offers a command called *estimatesmartfee* to estimate the optimal fee rate to attach to a transaction in order to have it confirmed with high probability in N blocks, where N is chosen by the user and can be up to 1008. The algorithm, as described in the Bitcoin Core source code[5], works as follow: instead of tracking every single fee rate, which is too expensive both for storage and computation, Bitcoin Core groups transactions into exponentially spaced *buckets*. Transactions in the same bucket have similar fee rate. The algorithm then tracks the number of transactions that enters in each bucket and the number of transactions successfully included into the blockchain within the target. Moreover, to make the prediction more accurate, the algorithm gives more importance to recent blocks than to older blocks.

[2] https://en.bitcoin.it/wiki/Block_size_limit_controversy
[3] https://en.bitcoin.it/wiki/Miner_fees
[4] https://bitcoin.org/en/download
[5] https://github.com/bitcoin/bitcoin/blob/master/src/policy/fees.h

3 Probabilistic Logic Programming

In this paper we consider Probabilistic Logic Programming under the distribution semantics, as proposed in [26,31], which is capable of representing several domains [1,2,27]. A probabilistic logic program defines a probability distribution over logic programs called *worlds*. To define the probability of a query, this distribution is extended to a joint distribution of the query and the worlds. Consequently, the probability of the query is obtained from the joint distribution by summing out the worlds in a process called *marginalization*.

Each sentence of a logic program is called *clause* composed by a *head* and a *body*. An example of clause is: $tails(Coin) :- toss(Coin)$, where $tails(Coin)$ is called head and $toss(Coin)$ body. The previous clause can be read as: "if a $Coin$ is tossed then the $Coin$ lands tails". In these experiments we consider Logic Programs with Annotated Disjunctions (LPADs) [33] with no function symbols (if function symbols are allowed see [25]). Alternatives are expressed with disjunctive heads of clause where each atom is annotated with probability. An LPAD is composed by one or more clauses C_i. The general form of a clause is: $h_{i1} : \Pi_{i1}; \ldots ; h_{iv_i} : \Pi_{iv_i} :- b_{i1}, \ldots, b_{iu_i}$, 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 $body(C_i)$. Clauses where $\sum_{k=1}^{v_i} \Pi_{ik} < 1$ are also allowed: 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 $1 - \sum_{k=1}^{v_i} \Pi_{ik}$. An example of LPAD can be:

$$
\begin{aligned}
&heads(Coin) : 0.5; tails(Coin) : 0.5 :- toss(Coin), \backslash + biased(Coin).\\
&heads(Coin) : 0.6; tails(Coin) : 0.4 :- toss(Coin), biased(Coin).\\
&fair(Coin) : 0.9; biased(Coin) : 0.1.\\
&toss(coin).
\end{aligned}
$$

This program can be read as: if we toss a $Coin$ that is not (\+) biased then it lands heads with probability 0.5 and tails with probability 0.5. If we toss a $Coin$ that is biased then it lands heads with probability 0.6 and tails with probability 0.4. The third clause states that a $Coin$ is fair with probability 0.9 and biased with probability 0.1. The last clause assert that a $coin$ is certainly tossed.

Evaluating the probability of a query, a task called *inference*, is one of the main challenges in probabilistic (logic) programming. There are two types of inference: approximate inference and exact inference. Exact inference is used when the problem has to be solved exactly. Several tools that performs exact inference have been presented, such as PITA [28,29]. The main disadvantage of exact inference is that it is, in general, #P-complete [16] so it is not usable for large domains. A possible alternative to exact inference is *approximate* inference. Both types of inferences are implemented in *cplint* [27], accessible also online[6].

[6] http://cplint.eu/

3.1 Conditional Approximate Inference

Approximate inference in cplint is performed using Monte Carlo algorithms [6, 24]. Each algorithm is usually composed by the following steps: 1) sampling a world by sampling each ground probabilistic fact, 2) checking if the query is true in the world, 3) compute the probability p of the query as the fraction of samples where the query is true and 4) repeat the process for a fixed number of times or until convergence. This process is still very expensive for large programs because the generation of a world requires sampling many probabilistic facts. To reduce the number of calculations, usually samples are evaluated lazily, i.e., the sampling of probabilistic facts is performed only when required by a proof [26].

Using Monte Carlo methods, it is also possible to compute the probability of a query given a certain evidence, using algorithms such as rejection sampling or Metropolis-Hastings Markov Chain Monte Carlo (MCMC). In the case that the evidence is on atoms that have continuous values as argument, *likelihood weighting* must be used [21]. In likelihood weighting, each sample has an associated weight based on the evidence. The total probability of the query is then computed summing all the weights of the samples where the query is true and then dividing this value by the total sum of the weights of the samples.

In cplint, (conditional) approximate inference can be done using the module MCINTYRE [24]. cplint also allows the definitions of continuous random variables using the syntax `A:Density:- Body`. In particular, `g(X):gaussian(X,0, 1)` states that argument X of $g(X)$ follows a Gaussian distribution with mean 0 and variance 1. The following example shows how to model a mixture of two Gaussians: a biased coin is toss. With probability 0.6 it lands heads, with probability 0.4 it lands tails. If it lands heads, X in $mix(X)$ is sampled from a Gaussian with mean 0 and variance 1. If it lands tails, X is sampled from a Gaussian with mean 5 and variance 2.

$$\begin{aligned}
&heads : 0.6; tails : 0.4.\\
&g(X) : gaussian(X, 0, 1).\\
&h(X) : gaussian(X, 5, 2).\\
&mix(X) :- \ heads, g(X).\\
&mix(X) :- \ tails, h(X).
\end{aligned}$$

Using cplint, we can take N samples of X in $mix(X)$ by querying `mc_sample_arg (mix(X),N,X,L0)` or we can take N samples of X in $mix(X)$ given that *heads* was true by querying `mc_mh_sample_arg(mix(X),heads,N,X,L0)`.

4 Modelling Transaction Fee with Probabilistic Logic Programming

Transaction fee are a hot topic in Bitcoin. As said above, miners are interested in selecting only the most profitable transactions while users are interested in minimizing the cost for a transaction. There are several sources of uncertainty that makes the computation of the optimal value a complicated task. One of them is block discovery time [5]. Its probability distribution can be described with a Poisson distribution with rate (usually indicated with λ) 10 since all the events are independent i.e., the discovery time of a block does not give information about the next block and blocks are discovered every 10 minutes on average. To keep the block production rate constant, the *target* value, that conditions the block discovery time, is dynamically updated every 2016 blocks, based on the time it took to find the last 2016 blocks.

Other sources of uncertainty, just to name a few, are: the number of transactions broadcast every minute, the average size of them and the average size of a block. All of them can be modelled with a Normal (also known as Gaussian) distribution. This distribution is characterized by two parameters, *mean* (μ) and *variance* (σ^2) and is well suited to model data that tends to be around a central value. Moreover, thanks to the Central Limit Theorem, the Poisson distribution with mean λ can be approximated with a Gaussian distribution with mean and variance λ, i.e., $Possion(\lambda) \approx Gaussian(\lambda, \lambda)$.

5 Experiments

In this paper we use probabilistic programming to model two real world scenarios: computing the amount of fees collected by a miner over time and computing the Bitcoin transaction fees trend.

In the first experiment, we are interested in computing how transaction fees affect the average profit of a miner. Nowadays most of the revenues of miners come from blocks reward: each miner that appends a block to the blockchain receives a certain amount of bitcoin. However, the Bitcoin block mining reward halves every 210,000 blocks so, the more blocks will be appended to the main chain, the less will the miner's revenue be. Currently, the revenue is 12.5 bitcoin but approximately by the end of 2020 it will be halved. Therefore, in the future, transaction fees will have a central role in supporting the miners activity.

In this experiment we modelled the number of transactions in a block (N_{tx}), and the transaction reward R as Gaussian distributions. For both, the mean of the distribution is sampled from another Gaussian distribution. The obtained fees are $N_{tx} * R$. The model is shown in Listing 5-1:

```
mean_r(M):gaussian(M,18,2).
mean_b(M):gaussian(M,700,25).
revenue(_,M,R):gaussian(R,M,2).
block_size(_,M,S):gaussian(S,M,25).

val_r(I,V):- mean_r(M), revenue(I,M,V).
val_b(I,V):- mean_b(M), block_size(I,M,V).

obtained_fees(I,O):- val_r(I,R), val_b(I,B), O is R*B/100000.
```

Listing 5-1. Example of a model.

The predicates `val_b/2` and `val_r/2` compute the average size of a block and the average fee rate. Finally, `obtained_fees/2` computes the amount of fees received for creating one block. The output value is divided by 10^5 to get the value in bitcoin, since the average fee rate is in satoshi/byte and the block size in kilobyte. To compute the results, we used the predicates `mc_expectation/4` and `mc_lw_expectation/5` from the cplint package. The signature of the first predicate is the following: `mc_expectation(+Query:atom,+N:int,?Arg:var,-Exp: float)`. It takes `N` samples of `Query` and sums up the value of `Arg` for each sample. The overall sum is divided by `N` to give `Exp`. The second predicate has one more argument, the evidence. The difference with respect to the previous one is that each sample is weighted by the likelihood of evidence in the sample, according to likelihood weighting. The results are shown in Fig. 1, where we observed `val_r/2` with `V` variable according to the legend and in Fig. 2 where we observed `val_b/2` with `V` variable according to the legend. For both experiments we used 1000 samples.

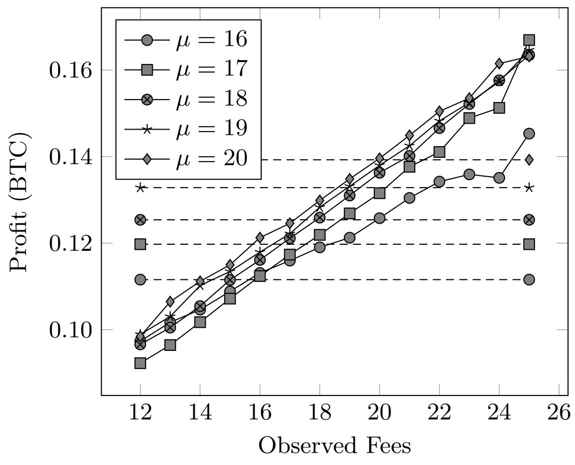

Fig. 1. The graph shows how a variation in the average bitcoin fee rate can influence the miner profit. The data are computed by setting the parameters for the Gaussian distribution for block size as $\mu = 700$ and $\sigma^2 = 25$ and for rewards as $\sigma^2 = 5$ and μ according to the legend. The straight lines represent the values computed without observations (obtained with `mc_expectation/3`).

In the second experiment, we want to understand how transaction fees may vary over time. In particular, we want to know what is the probability that a transaction with a certain fee rate is confirmed after a given number of blocks. We start by defining the probability distributions of the involved variables. Looking at Bitcoin data from blockchain.com, we retrieved the average number of transactions in a block, the average block discovery time and the average number of transactions added to the mempool per second. All these variables can

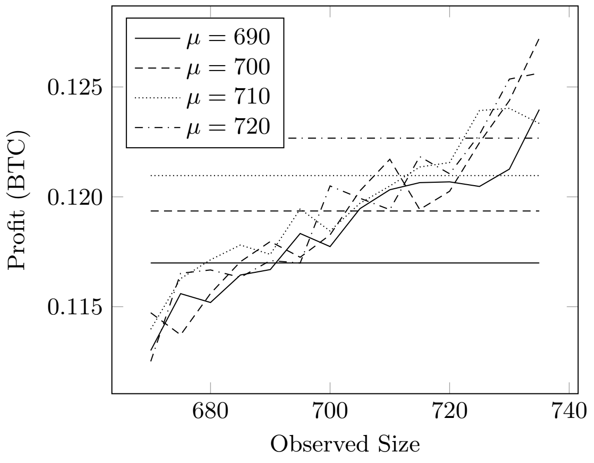

Fig. 2. The graph relates the block size with the average profit obtained from fees. The parameters for the distribution for block size are $\sigma^2 = 25$ and μ variable and for the reward $\mu = 17$ and $\sigma^2 = 2$. Straight lines represent the values computed without observations.

be assumed to follow a Poisson distribution that can be approximated with a Gaussian distribution. The average transaction fee rate is also modelled with a Gaussian distribution. However, because this value often varies over time, we re-sample the mean of the distribution of the fees for every iteration. The model is shown in Listing 5-2.

```
average_fee(_,M):uniform(M,15,25).
compute_fee(_,M,F):gaussian(F,M,4).

fee(I,F):- average_fee(I,M), compute_fee(I,M,F).

compute_time(F,M,V):gaussian(F,M,V).
number_of_tx_in_block(_,N):gaussian(N,1600,1600).
block_discovery_time(_,N):gaussian(N,500,500).
tx_per_second(_,N):poisson(N,5).

generate_pool(N,N,[]):-!.
generate_pool(I,N,[F|T]):- I < N, fee(I,F), I1 is I+1,
    generate_pool(I1,N,T).

get_len(A,B,B):- A >= B, !.
get_len(A,B,A1):- A < B, A1 is A-1.

loop_pool(FeeRate,I,NBlocks,Pool):- I =< NBlocks,!,
    number_of_tx_in_block(I,N), N11 is round(N),
    length(Pool,LP), get_len(LP,N11,N1),
    length(L,N1), append(L,RemPool,Pool),
    loop_pool_check(FeeRate,I,RemPool,NBlocks).

loop_pool_check(_,_,[],_):- !.
loop_pool_check(FeeRate,_,[H|_],_):- H < FeeRate,!.
loop_pool_check(FeeRate,I,RemPool,NBlocks):- !,
    I1 is I+1, block_discovery_time(I,Time),
    tx_per_second(I,T), NNewTx is T*Time,
    NT1 is round(NNewTx),
    generate_pool(0,NT1,NewArrived),
    append(NewArrived,RemPool,NewPool),
    sort(0, @>=, NewPool, PoolSorted),
    loop_pool(FeeRate,I1,NBlocks,PoolSorted).

included(_I,FeeRate,NBlocks):-
    loop_pool_check(FeeRate,0,[FeeRate],NBlocks).
```

Listing 5-2. Example of model.

The program creates an initial pool of N transactions by sampling N times a value from a Gaussian distribution using the predicates `generate_pool/3` and `fee/2`. To compute how many blocks we need to wait to confirm a transaction with associated fee rate F, we sort the pool, compute the average number N_b of transactions in a block, the average block discovery time T and the average number of transactions per second N_{txs} (predicates `loop_pool/4` and `loop_pool_check/4`). We then compute $N_{txs} * T = N_{ta}$, the number of transactions arrived during the last block creation. To simulate the inclusion of N_{ta} transactions in a block we removed the best N_b transactions from the mempool (we suppose the miner acts as expected, i.e., he includes only the most profitable transactions). If the transaction with the best fee rate in the remaining mempool has a value less than F, this means that the transaction we consider has been successfully included in a block and the iteration stops. Otherwise, we simulate the arrival of N_{ta} new transactions to the mempool with `generate_pool/3` and repeat the process. In this case, we compute the results using both `mc_sample/3` and `mc_lw_sample/4` provided by the cplint package. The first one samples the goal a certain number of times and computes the probability of success. The second one works in a similar way but, in addition, performs likelihood weighting: each sample is weighted by the likelihood of the evidence in the sample. Results are shown in Fig. 3. The used parameters are shown in Listing 5-2 and $NBlocks$ was set to 1 (next block). For instance, if $\phi = 16$ the query is: `?- mc_lw_sample(included(1,16,1),included(0,ObservedFees,1), NSamples, Probability)`. As expected, the confirmation probability decreases as the observed fees increase. ϕ represents the fees associated to a transaction. The experiments were executed computing 250 samples. Value of observed fees less than ϕ gives probability $= 1$ and so are not reported in the graph.

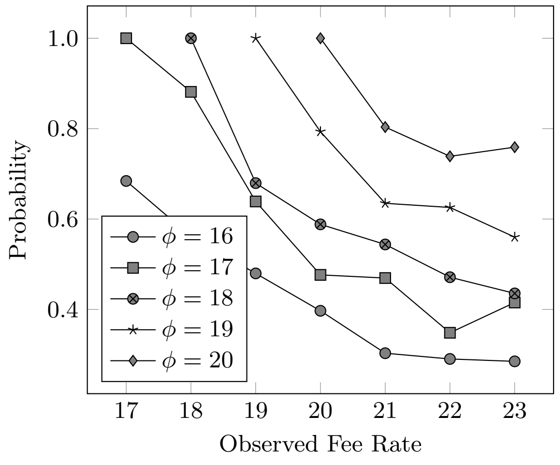

Fig. 3. The graph shows how transaction fees influence the probability of confirmation in N blocks. We selected a value of fee rate (ϕ) for the transaction under consideration and then computed how probability changes according observed fee rate.

6 Conclusion

In this paper we show how to model blockchain fees using probabilistic logic programming. Starting from real world data, we try to model how transaction fees influence the confirmation time and how transaction fees contribute to miners revenue. Despite being a relatively new technology, Bitcoin has attracted a lot of interest in research. There are several papers that study Bitcoin behaviour such as [3, 22, 30] where the authors study the so-called double spending attack. A game theoretic analysis can be found in [18] where Bitcoin is analyzed in a situation where all the participants behave according to their incentives. However, there are only few works in the literature concerning Bitcoin fees. In particular, in [15, 17] the authors proposed a method based on queuing theory to model how fees effect the confirmation time. To avoid price fluctuation, authors in [4] proposed a new method to computes fees. An analysis on how block reward, transaction fees and their ratio influences the Bitcoin ecosystem can be found in [9].

To extend our work, decision theory and game theory models can be used to deeply analyze miner's behaviour and understand how they can optimally select transactions based on several profit variables. Another interesting direction could be the usage of deep learning models [12] to analyse historical data and predict the evolution of the system.

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).

https://doi.org/10.3233/IA-170105
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). https://doi.org/10.1007/978-3-319-49130-1_26
3. Azzolini, D., Riguzzi, F., Lamma, E., Bellodi, E., Zese, R.: Modeling bitcoin protocols with probabilistic logic programming. In: Bellodi, E., Schrijvers, T. (eds.) Proceedings of the 5th International Workshop on Probabilistic Logic Programming, PLP 2018, co-located with the 28th International Conference on Inductive Logic Programming (ILP 2018), Ferrara, Italy, September 1, 2018. CEUR Workshop Proceedings, vol. 2219, pp. 49–61. CEUR-WS.org (2018), http://ceur-ws.org/Vol-2219/paper6.pdf
4. Basu, S., Easley, D., O'Hara, M., Sirer, E.G.: Towards a functional fee market for cryptocurrencies. CoRR **abs/1901.06830** (2019), http://arxiv.org/abs/1901.06830
5. Bowden, R., Keeler, H.P., Krzesinski, A.E., Taylor, P.G.: Block arrivals in the bitcoin blockchain. CoRR **abs/1801.07447** (2018), http://arxiv.org/abs/1801.07447
6. Bragaglia, S., Riguzzi, F.: Approximate inference for logic programs with annotated disjunctions. In: ILP 2011. LNAI, vol. 6489, pp. 30–37. Springer, Florence, Italy (27-30 June 2011)
7. Buterin, V.: A next-generation smart contract and decentralized application platform (2014), https://github.com/ethereum/wiki/wiki/White-Paper, accessed February 14, 2019
8. Cardano., https://whycardano.com/
9. Carlsten, M., Kalodner, H., Weinberg, S.M., Narayanan, A.: On the instability of bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 154–167. ACM (2016)
10. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Mach. Learn. **100**(1), 5–47 (2015)
11. Eosio - an introduction by ian grigg, https://eos.io/introduction
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning, vol. 1. MIT Press (2016)
13. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. In: Conference on the Theory and Application of Cryptography. pp. 437–455. Springer (1990)
14. Hyperledger., https://www.hyperledger.org/
15. Kasahara, S., Kawahara, J.: Priority mechanism of bitcoin and its effect on transaction-confirmation process. CoRR **abs/1604.00103** (2016), http://arxiv.org/abs/1604.00103
16. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. Adaptive computation and machine learning, MIT Press, Cambridge, MA (2009)
17. Koops, D.T.: Predicting the confirmation time of bitcoin transactions. CoRR **abs/1809.10596** (2018), http://arxiv.org/abs/1809.10596
18. Kroll, J.A., Davey, I.C., Felten, E.W.: The economics of bitcoin mining, or bitcoin in the presence of adversaries. In: Proceedings of WEIS. vol. 2013, p. 11 (2013)
19. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
20. Nguembang Fadja, A., Riguzzi, F.: Probabilistic logic programming in action. In: Holzinger, A., Goebel, R., Ferri, M., Palade, V. (eds.) Towards Integrative Machine Learning and Knowledge Extraction, LNCS, vol. 10344. Springer (2017). https://doi.org/10.1007/978-3-319-69775-8_5
21. Nitti, D.: Hybrid Probabilistic Logic Programming. Ph.D. thesis, KU Leuven (2106)

22. Pinzón, C., Rocha, C.: Double-spend attack models with time advantange for bitcoin. Electr. Notes Theor. Comput. Sci. **329**, 79–103 (2016). https://doi.org/10.1016/j.entcs.2016.12.006, https://doi.org/10.1016/j.entcs.2016.12.006
23. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016), https://lightning.network/lightning-network-paper.pdf
24. Riguzzi, F.: MCINTYRE: A Monte Carlo system for probabilistic logic programming. Fund. Inform. **124**(4), 521–541 (2013). https://doi.org/10.3233/FI-2013-847
25. Riguzzi, F.: The distribution semantics for normal programs with function symbols. Int. J. Approx. Reason. **77**, 1 – 19 (October 2016). https://doi.org/10.1016/j.ijar.2016.05.005
26. Riguzzi, F.: Foundations of Probabilistic Logic Programming. River Publishers, Gistrup,Denmark (2018), http://www.riverpublishers.com/book_details.php?book_id=660
27. 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). https://doi.org/10.1002/spe.2386
28. Riguzzi, F., Swift, T.: Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions. In: ICLP TC 2010. LIPIcs, vol. 7, pp. 162–171. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010). https://doi.org/10.4230/LIPIcs.ICLP.2010.162
29. Riguzzi, F., Swift, T.: The PITA system: Tabling and answer subsumption for reasoning under uncertainty. Theor. Pract. Log. Prog. **11**(4–5), 433–449 (2011). https://doi.org/10.1017/S147106841100010X
30. Rosenfeld, M.: Analysis of hashrate-based double spending. CoRR **abs/1402.2009** (2014), http://arxiv.org/abs/1402.2009
31. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995. pp. 715–729. MIT Press (1995)
32. Swan, M.: Blockchain: Blueprint for a new economy. "O'Reilly Media, Inc." (2015)
33. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic Programs With Annotated Disjunctions. In: ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer (2004)
34. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)