A Probabilistic Logic Model of Lightning Network

Damiano Azzolini[2], Fabrizio Riguzzi[2], Elena Bellodi[1], and Evelina Lamma[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,fabrizio.riguzzi,
elena.bellodi,evelina.lamma}@unife.it

Abstract. One of the main limitations of blockchain systems based on Proof of Work is scalability, making them unsuitable for e-commerce and small payments. Currently, one of the principal directions to overcome the scalability issue is to use the so-called "layer two" solutions, like Lightning Network, where users can open channels and send payments through them. In this paper, we propose a Probabilistic Logic model of Lightning Network, and we show how it can be adopted to compute several properties of it. We conduct some experiments to prove the applicability of the model, rather than providing a comprehensive analysis of the network.

Keywords: Probabilistic Logic Programming · Blockchain · Lightning Network.

1 Introduction

The scalability trilemma states: "Scalability, decentralization, security: you can have only two of the three". This is the main issue all blockchains must face. Proof of Work (PoW) based blockchains, such as Bitcoin [14] and Ethereum [24], are secure and (theoretically) decentralized, but not very scalable [7]. This is because, currently, the PoW consensus algorithm requires solving a computationally hard problem. At the moment of writing, in the case of Bitcoin, the average number of transactions per second is 7, for Ethereum 15, making them unusable for every day small payments. Blockchains based on Proof of Stake (PoS) or Delegated Proof of Stake (DPos) can support a higher number of transactions but at the cost of less decentralization.

Among all the various proposals to increase the number of processed transactions, such as Sharding [11] and sidechains [6], the so-called "layer two" solutions remain the most adopted. One of the most famous is Lightning Network [16] (LN). In Lightning Network users can open, through a transaction on the main chain, a bidirectional Payment Channel, and then use the funds locked in this channel to issue transactions, without utilizing the main chain. The capacity distribution in a channel is unknown, to preserve privacy. An important feature of LN is that it also allows the routing of payments between two users not directly connected by a channel.

Probabilistic Logic Programming (PLP) is a powerful language to represent scenarios where probability has a central role: it combines the expressivity of Logic Programming with uncertainty over facts, and it has been already used to model several

blockchain-related scenarios [3,5]. Routing in the LN can be seen as uncertain, in the sense that users may not be active for some reasons or may refuse to forward the payment. Furthermore, there is uncertainty on the funds distribution on a channel.

In [2] the authors proposed a logic model of the Bitcoin LN; here, we extend it by proposing a *probabilistic* logic model of Bitcoin LN, and we show how this model can represent the uncertain scenario cited above, allowing, for instance, to compute the probability of a successful payment routing. We focus on Lightning Network built upon Bitcoin. However, our model can be extended to a general blockchain.

The paper is structured as follows: in Section 2 we describe the general structure and features of the Bitcoin Lightning Network. Section 3 introduces the PLP base concepts needed to understand the LN model developed and discussed in Section 4. Section 5 presents a possible application of the model to compute routing probabilities, and Section 6 concludes the paper.

2 Blockchain, Bitcoin and Lightning Network

Bitcoin was designed by Nakamoto in 2008 [14] with the goal to create a decentralized payment system where users can issue transactions without the need of a centralized authority. Bitcoin blockchain offers several features such as immutability, auditability, and transaction atomicity. These features make it theoretically suitable for payments and micro payments. However, scalability is still one of the main limitations of the system and does not allow an increasing number of transactions.

A blockchain is based on a linear sequence of blocks, linked together by cryptographic functions. To append a block to the chain, a miner must execute, in the case of Bitcoin, a Proof of Work (PoW) algorithm which involves finding a solution to a hard puzzle. This mechanism ensures that blocks, once discovered, cannot be tampered with. On the other hand, this is a huge bottleneck for scalability. In fact, as computing power increases and technology evolves, the difficulty of PoW increases, keeping the average discovery time fixed to approximately one block every ten minutes, and thus limiting the number of transactions that can be processed by the system since blocks have a fixed size.

Another feature of Bitcoin that restricts the scalability is the block size limit (1Mb), a very controversial topic[3]. An increase of this limit will allow the system to process more transactions. However, this would require more computation power to store and manage the blockchain, reducing the decentralization of the system. Furthermore, forks would be more likely to happen, due to the slower propagation time. Finally, a change in block size can be done only with a hard fork, an update that would be backward incompatible: this can cause a consensus failure, making the system completely unreliable.

At the moment, the scalability problem is not solved. In the past, several improvements were proposed (called Bitcoin Improvement Proposal[4]) that slowly increased the number of manageable transactions. One of the first, advanced at the end of 2015

[3] https://en.bitcoin.it/wiki/Block_size_limit_controversy
[4] https://en.bitcoin.it/wiki/Bitcoin_Improvement_Proposals

and accepted few years later, is Segregated Witness (SegWit)[5]. In a nutshell, SegWit increases the capacity of a block by removing signature data from a transaction and introducing the definition of Virtual Size of a block and block weight, measured in weight unit instead of bytes.

Another related improvement proposal is the one that suggests the usage of Schnorr signatures [12,21]. Currently, to send transactions, signatures are needed. In the case a user wants to send a transaction from multiple addresses to one, every transaction requires its own signature, increasing the transaction size, making it more expensive. After the implementation of Schnorr signatures, if users control multiple addresses, they can move the funds from those addresses to a single address using only one signature, making the transaction lighter. These two solutions combined would increase the number of manageable transactions, but the Bitcoin system would still be limited.

Another approach consists in the so-called "layer two" solutions, based on an underlying blockchain: the main idea is to create a layer of channels on top of it where users can interact without facing the scalability problem of the blockchain. Lightning Network [16] (LN) is currently one of the most promising "layer two" solutions. It consists of a peer-to-peer network based on a blockchain, such as Bitcoin, where users can send payments and micro payments through bidirectional payment channels, without having to pay high transaction fees, and without the need of long confirmation times. To open a channel, users must broadcast an initial funding transaction on the underlying blockchain. To update the state of the channel, they can create a commitment transaction that is not published on the main chain. To close the channel, they must agree on the state of the channel and then publish a closing transaction.

One of the main features of LN is the possibility to send payments also to not directly connected users, through multi hop payments, assuming that the source and the destination are linked through intermediate connections. Moreover, thanks to Hashed Timelock Contracts (HTLC)[6], there is no need for trust between users.

However, the capacity of a channel between two users A and B is known, but the distribution of the capacity in each direction is unknown, since it is a feature introduced to increase the security of the system[7]. Due to this characteristic, routing is a complicated task in Lightning Network, and can be considered probabilistic.

3 Probabilistic Logic Programming

Logic Programming (LP) is a powerful language that allows one to express complex models with few lines of codes. One of the main limitations of Logic Programming is that it cannot manage uncertainty. Probabilistic Logic Programming [9,18] extends LP by allowing the definition of probabilistic facts that can follow several probability distributions. Initially, only Bernoulli distributions [20] (or generalized Bernoulli distributions [22]) were proposed.

[5] https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki

[6] https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts

[7] https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md

The distribution semantics [19] gives a precise meaning to PLP without function symbols (with finite grounding). An *atomic choice* indicates whether a grounding (i.e., a substitution term/variable) for a probabilistic fact is selected. A set of atomic choices is *consistent* if it does not contain two different alternatives (selected and not selected) for the same probabilistic fact. A consistent set of atomic choices forms a *composite choice*: in the case it contains an atomic choice for every grounding of every probabilistic fact it is named a *selection*. A selection identifies a ground logic program called *world*, and its probability can be computed as the product of the probabilities of the atomic choices. The probability P of a query q can then be computed as the sum of the probabilities of the worlds w in which the query is true:

$$P(q) = \sum_{w \models q} P(w)$$

Here, we consider the PLP language ProbLog [10]. A probabilistic fact f_i has the following syntax:

$$p_i :: f_i$$

meaning that f_i is true with probability $p_i \in]0,1]$, and false otherwise. If $p_i = 1$ the fact is deterministic, i.e., it is always true. An example of a Probabilistic Logic Program is shown below:

```
0.7::no_sleep.
0.8::too_much_work.
tired:- no_sleep.
tired:- too_much_work.
```

The first two lines are *probabilistic facts*, while the third and fourth lines are *clauses*. A clause is composed of a *head* and a *body* separated by the neck operator (`:-`). The head is true if the body is true. Here, for the first clause, the head is `tired` and the body is `no_sleep`. Both these clauses are *ground* since they do not contain variables (denoted with the first letter uppercase). In other words, this program models a person who is `tired` if he/she does not sleep enough or if he/she works too much. With probability 0.7, the person does not sleep enough (`no_sleep`), and with probability 0.8 he/she works too much (`too_much_work`). We can then ask the probability that the person is tired (`tired`) obtaining 0.94 ($0.7 \cdot 0.8 + 0.7 \cdot 0.2 + 0.3 \cdot 0.8$).

One of the main issues in PLP (and LP in general) is that the grounding of a program may be huge, so managing it can be very difficult. To tackle this, usually probabilistic logic programs are compiled into a more compact form through a process called knowledge compilation [8].

In the last few years, Hybrid Probabilistic Logic Programs arose [4,13], with the possibility to define continuous random variables and constraints among them. To consider continuous probability distributions, we introduce the syntax used by `cplint` [1] and its module MCINTYRE [17]. Continuous random variables can be encoded with:

$$f : Density$$

where f is an atom (a predicate symbol followed by a number of arguments) with a continuous variable as argument and *Density* is a special atom that models a probability density on the argument of the atom. For example,

```
f(X) : gaussian(X,0,2).
```

indicates that `X` in `f(X)` follows a Gaussian distribution with mean 0 and variance 2. Inference in these types of programs can be done, for example, by sampling [17].

4 Network Model

The LN can be represented as a graph where users (nodes) are linked by connections (edges) with capacity distributed according to some probability distribution, since the real distribution is unknown (except for the two users that opened the channel). Let us start with a deterministic model, that will be later extended to a probabilistic one. A connection between two nodes is represented with a fact `edge(A,B,Capacity)`, where `A` and `B` are the two nodes and `Capacity` is the capacity of the connection. The whole LN is represented as a list of `edge/3` facts (where `/3` indicates the arity, i.e., the number of arguments). For these experiments, we do not consider fee base and fee rate. However, they can be straightforwardly included in the analysis by simply adding more arguments. Connections are undirected, meaning that payments can go in both directions, and are represented with the following predicate `connected/3`:

```
connected(A,B,C):- edge(A,B,C) ; edge(B,A,C).
```

`A` and `B` are connected with a channel of capacity `C` if there is an edge from `A` to `B` or (`;`) from `B` to `A` with capacity `C`. The degree of a node is the number of edges incident to the node. We can search for a path between two nodes with a standard Prolog predicate. An example written using SWI-Prolog [23] is reported here for the sake of clarity:

```
connected_test(Source,Next,Size):-
    connected(Source,Next,Cap),
    Size < Cap.

path(Dest,Dest,_,_,Path,Path).
path(Source,Dest,Size,NSteps,Visited,Path) :-
    length(Visited,N),
    N < NSteps,
    connected_test(Source,Next,Size),
    \+ memberchk(Next,Visited),
    path(Next,Dest,Size,NSteps,[Next|Visited],Path).
```

The predicate `path/6` states that there is a path from `Source` to `Dest` that can route a payment of size `Size` if the two nodes are connected by intermediate nodes that have not been already visited (condition checked with `\+memberchk`, a deterministic version of `member/2`). Ensuring that a node has not been already visited is fundamental, otherwise we may get stuck in a loop where we move an infinite number of times between a pair of nodes. The length `N` of the path is bounded using the standard Prolog comparison predicate `</2`, that compares the length of the list containing already visited nodes (`Visited`) with a user defined threshold (`NSteps`). In a similar way, the capacity `Cap` is checked against the size `Size` of the payment in a second predicate called `connected_test`. A sample call to `path/6` is `path(a,c,10,3,[],P)`,

were we look for a path `P` from `a` to `c` of at most `3` edges that can route a payment of size `10`, starting with an empty list (`[]`) of visited nodes.

The previous code does not consider the capacity distribution in a channel. Let us now extend it with a probabilistic fact to represent it. Using `cplint` on swish [1], we can define continuous random variables (say for example uniformly distributed in [L,U]) with:

```
distr(X,L,U) : uniform_dens(X,L,U).
```

Here, `X` has a uniform distribution between `L` and `U`, where `L` is the minimum value and `U` the maximum value. The predicate `connected_test` can be modified as:

```
connected_test(Source,Next,Size):-
    connected(Source,Next,Cap),
    distr(C,0,Cap),
    Size < C.
```

In other words, we collect the capacity of the channel between two nodes, and we state that the capacity from `Source` to `Next` is given by a random variable uniformly distributed between `0` and `Cap`. With this definition, at each sampling iteration, the distribution for every channel is fixed and does not change. Clearly, to successfully route a payment, its size must be smaller than the capacity in the considered direction (`Size < C`).

We can further extend the previous model by considering also intermittent edges. This situation may arise when a node disconnects from the network or when declines to forward a payment. To model it, we can define a Bernoulli random variable that is true with a certain probability (set to 0.95 in the following code snippet):

```
0.95::active(_).
```

The predicate is then extended as:

```
connected_test(Source,Next,Size):-
    connected(Source,Next,Cap),
    active(Next),
    distr(C,0,Cap),
    Size < C.
```

In the experiments, we modelled the capacity of a channel using a uniform distribution, and the probability that a node is active with a Bernoulli distribution. However, several extensions and variations can be made: changing the type of distribution for the channel capacity (Gaussian with mean equal to half of the capacity, for example) or even setting the probability that a node declines to route a payment proportional to the payment size, to the fees or a combination of the two.

5 Routing Analysis

We used a snapshot of the LN taken from `https://ln.bigsun.xyz/` on the 12th of April 2021. We considered only the open channels, resulting in a network with 14734

Capacity (sat)	Occurrences	Degrees	Occurrences	Highest Capacities (sat)	Occurrences
100000	3942	1	6958	500000000	4
1000000	3523	2	2575	477184791	1
500000	2794	3	1372	354000000	1
2000000	1533	4	802	300000000	2
16777215	1522	5	549	250000000	1
200000	1446	6	397	238135604	1
5000000	1359	7	263	225118006	1
20000	1297	8	209	200000000	28
10000000	1131	9	182	179792707	1
50000	1079	10	145	154222260	1

Table 1. Information about channel capacities and node degrees.

nodes, 44349 channels, and with a total capacity of 125819660675 satoshi (1258.19 bitcoin). Table 1 shows a recap of the most common channel capacities and node degrees (considering also duplicated edges that connect the same pair of nodes).

To demonstrate how PLP can be used to model the LN, we conducted some experiments. For all of them, we fixed the maximum length of the path. Moreover, we suppose that the distribution of the capacity in a channel is uniform (i.e., if the channel that connects A and B has capacity 1, we can route from A to B a uniform distributed value between 0 and 1, and from B to A the remaining), since we do not have further information.

To select the range of the payment size, we first compute the average of the capacities of all the connections. We obtained approximately 2837035 satoshi but with a huge standard deviation ($> 10^7$). So, we counted the number of connections that have less than the average capacity, less than half of the average capacity, and less than a quarter of the average capacity, obtaining respectively 35555 ($\approx 80\%$), 31902 ($\approx 72\%$), and 26077 ($\approx 59\%$). Figure 1 shows a more detailed graph, where the X axis indicates the percentage of the average capacity and the Y axis indicates both the number of connections (edges) with less than that value, and the relative percentage of the total connections. To further analyse the distribution of the capacity, we removed the nodes with the highest capacities and plotted the variation of the total capacity. The results are shown in Fig 2.

In a first test, we want to compute the probability to successfully route a payment of varying size between two different random nodes of the same degree at the first attempt, given that nodes may not be active. One of the main requisites to route a payment is that all the nodes along the path are active, otherwise it cannot be sent. Moreover, as said before, a node may refuse to forward the payment for some reasons. We fix the length of the path (number of intermediate edges) to 2 and the node degree of source and destination to 2, 5, and 10. We then plot how the probability varies when intermediate nodes may not be active. Probability values are computed with the predicate `mc_sample(+Query:atom,+N:int,-Prob:float)`, provided by the MCINTYRE module [17], that samples the query `Query` `N` times and returns the ratio between the number of successes and the number of samples (`Prob`). We set the number of samples to 1000.

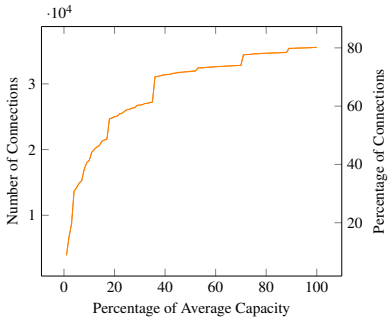

Fig. 1. Number and percentage of connections with less than a certain percentage of the average capacity (2837035).

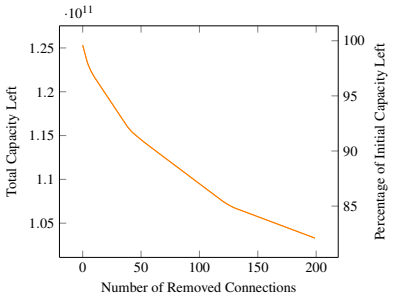

Fig. 2. Value and percentage of capacity left after removing the top n (X axis) connections in terms of capacity.

Figure 3 shows the results for the first experiment: nodes with higher degrees have, as expected, a higher probability to successfully route a payment on the first attempt, even if this gap reduces as the payment size increases. Figure 4 shows the results for the experiments with a varying probability of intermediate nodes to be active: the probability of a successful routing decreases, but not so drastically. If the source and the destination are the same node, and the length of the path is greater than 2 (at least the source node and another one, if these two nodes are connected by at least 2 edges, otherwise the path should be composed of at least 2 additional nodes plus the source), we can compute the probability of a successful *rebalance*. This is a common situation since, if users want to refill a channel, currently one of the main solutions is to send a circular payment to themselves.

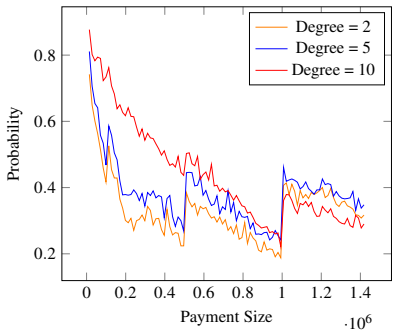

Fig. 3. Probability of a successful payment of varying size between random connected nodes of degree 2, 5, and 10.

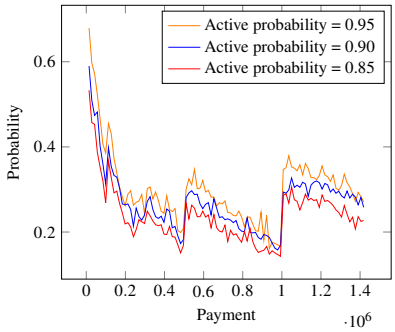

Fig. 4. Probability of a successful payment of varying size between random connected nodes of degree 2, with different active probabilities for intermediate nodes.

In another experiment, we want to know the probability of a successful payment between two random nodes of variable degrees, given that the payment is split in various equal parts. That is, we split a payment into N parts and compute the probability that all these payments succeed at the first attempt. This is a common scenario in LN [15], because, when the size of the payment increases, the probability of success decreases, due to the channel capacity limitation. However, increasing the number of payments also increases the fees required to route the payment, due to the necessity to issue multiple transactions. The graphs in Figures 5, 6, and 7 show how the probability of a successful payment varies when the payment is split in 2, 3 or 4 parts and intermediate nodes are always active. In Figures 8, 9, and 10 we performed the same experiment but we fixed the degree of source and destination to 2, and we varied both the probability that a node is active and the number of parts of a payment.

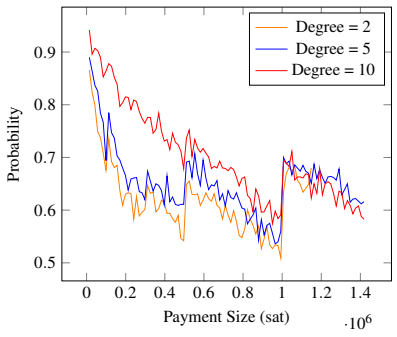

Fig. 5. Probability of a successful payment of varying size split into 2 equal parts, between connected nodes of various degrees.

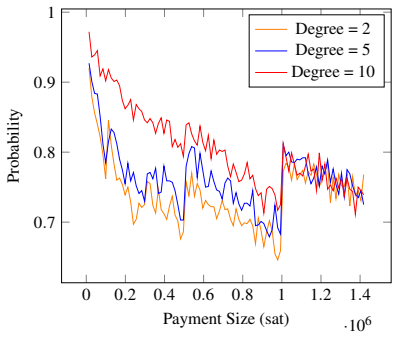

Fig. 6. Probability of a successful payment of varying size split in 3 equal parts, between connected nodes of various degrees.

We can see that all the plots present two sudden jumps in probability, around 500000 ($5 \cdot 10^5$) and 1000000 (10^6): this is in accordance with the distribution of edges with these two values of capacity. In fact, 500000 and 1000000 are the second and third most common capacities (see Table 1). In our implementation, to route a payment, we randomly check if one of the edges between the current node and another node has enough capacity. If it has, it is selected. However, if the value of the payment is close to the total capacity of the selected edge, the routing will likely fail, since we suppose that the distribution is uniform. For example, if we try to route a payment of size $4.5 \cdot 10^5$ into a channel of capacity $5 \cdot 10^5$, we have only approximately 10% of chances to succeed (with a distribution supposed uniform). Since a lot of channels have capacity $5 \cdot 10^5$, when the payment approaches this value, the success probability of routing at the first attempt reduces. With a payment slightly greater than $5 \cdot 10^5$, these channels are no longer considered, since they do not have enough capacity, so the probability suddenly increases. Similarly happens with the value 10^6. Another, but less noticeable jump, can be found around 100000 (10^5), which is the most common value for the capacity of a channel. This happens for the same reasons explained before.

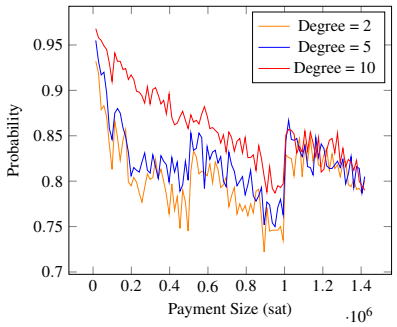

Fig. 7. Probability of a successful payment of varying size split into 4 equal parts, between connected nodes of various degrees.

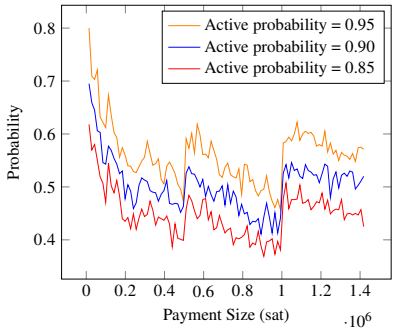

Fig. 8. Probability of a successful payment of varying size split into 2 equal parts, between connected nodes of degree 2 with varying active probability.

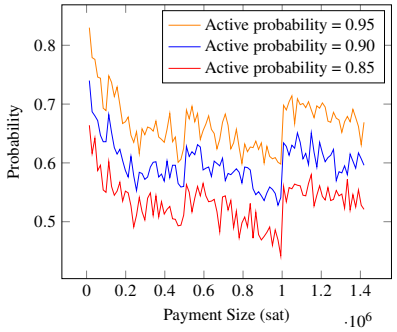

Fig. 9. Probability of a successful payment of varying size split into 3 equal parts, between connected nodes of degree 2 with varying active probability.

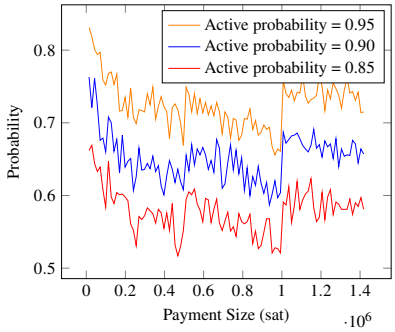

Fig. 10. Probability of a successful payment of varying size split into 4 equal parts, between connected nodes of degree 2 with varying active probability.

6 Conclusions

In this paper, we propose to analyse (Bitcoin) Lightning Network with Probabilistic Logic Programming. The usage of PLP allows representing the network with a highly expressive language. We described the network as an undirected graph with the capacity of a channel following a uniform distribution. Furthermore, we considered also the possibility to have intermittent nodes, a situation that may arise also when a node refuses to forward a payment. The goal of this paper is to prove the feasibility of a Probabilistic Logic model of the network, rather than provide and analysis of it, since the network continuously changes (and thus an analysis will be obsolete in a few weeks, even days or hours), and routing mechanism are typically more sophisticated than randomized routing. To test the applicability of a probabilistic logic model, we ran some experiments

on a real snapshot of the network, obtaining that a Probabilistic Logic analysis can be useful to model several uncertain scenarios.

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. Azzolini, D., Bellodi, E., Brancaleoni, A., Riguzzi, F., Lamma, E.: Modeling bitcoin lightning network by logic programming. Proceedings 36th International Conference on Logic Programming (Technical Communications) **325**, 258–260 (2020). https://doi.org/10.4204/EPTCS.325.30
3. Azzolini, D., Riguzzi, F., Lamma, E.: Studying transaction fees in the bitcoin blockchain with probabilistic logic programming. Information **10**(11), 335 (2019)
4. Azzolini, D., Riguzzi, F., Lamma, E.: A semantics for hybrid probabilistic logic programs with function symbols. Artif. Intell. **294**, 103452 (2021). https://doi.org/10.1016/j.artint.2021.103452
5. 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)
6. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains (2014)
7. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Gün Sirer, E., Song, D., Wattenhofer, R.: On scaling decentralized blockchains. In: Clark, J., Meiklejohn, S., Ryan, P.Y., Wallach, D., Brenner, M., Rohloff, K. (eds.) Financial Cryptography and Data Security. pp. 106–125. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
8. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002)
9. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming, LNCS, vol. 4911. Springer (2008)
10. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) IJCAI 2007. vol. 7, pp. 2462–2467. AAAI Press/IJCAI (2007)
11. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 17–30. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978389
12. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Designs, Codes and Cryptography **87**(9), 2139–2164 (Sep 2019). https://doi.org/10.1007/s10623-019-00608-x
13. Michels, S., Hommersom, A., Lucas, P.J.F., Velikova, M.: A new probabilistic constraint logic programming language based on a generalised distribution semantics. Artif. Intell. **228**, 1–44 (2015). https://doi.org/10.1016/j.artint.2015.06.008
14. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)

15. Piatkivskyi, D., Nowostawski, M.: Split payments in payment networks. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 67–75. Springer (2018)
16. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
17. 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
18. Riguzzi, F.: Foundations of Probabilistic Logic Programming. River Publishers, Gistrup, Denmark (2018)
19. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995. pp. 715–729. MIT Press (1995)
20. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: IJCAI 1997. vol. 97, pp. 1330–1339 (1997)
21. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
22. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3131, pp. 195–209. Springer, Berlin (2004)
23. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: Swi-prolog. Theory and Practice of Logic Programming **12**(1-2), 67–96 (2012)
24. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)