Increasing Energy Efficiency of Bitcoin Infrastructure with Reinforcement Learning and One-shot Path Planning for the Lightning Network

Danila Valko
L3S Research Center, Leibniz University Hannover
Hannover, Germany
d.v.valko@gmail.com

Daniel Kudenko
L3S Research Center, Leibniz University Hannover
Hannover, Germany
kudenko@l3s.de

ABSTRACT

The Lightning Network (LN) is a technological solution designed to solve the Bitcoin blockchain transaction speed problem by introducing off-chain transactions. Since LN is a sparse and highly distributed network with three predominant routing protocols, its native pathfinding algorithms can potentially find multi-hop payment paths similar from the payment sender's perspective, but the algorithms themselves have different performance, computational cost, energy consumption, and ultimately different CO_2 emissions per step in the pathfinding phase. Bitcoin itself generates approximately 61.4 million tonnes of CO_2 eq. per year. Since the LN is built on top of Bitcoin, every small change in its energy consumption can have a significant impact on overall pollution. In this paper, we show that the Reinforcement Learning (RL) approach can reduce these costs and achieve better performance in terms of energy consumption at each pathfinding step. We introduce one-shot path prediction and propose an RL solution for a network agent that learns its neighborhood and uses local knowledge to quickly solve the pathfinding problem. Based on a real LN snapshot, we empirically show that the RL solution can outperform native pathfinding algorithms in the case of an increasing number of transactions in relatively small neighborhoods.

KEYWORDS

Reinforcement Learning, Pathfinding, Green Computing, Lightning Network, Bitcoin Infrastructure

1 INTRODUCTION

New and emerging payment channel blockchain networks such as the Lightning Network (LN) [31] offer a potential solution to the scalability limitations of Proof-of-Work blockchains like Bitcoin and Ethereum by moving transactions off-chain. These networks also promise faster transaction times and lower fees compared to on-chain transactions. The LN operates under a unified specification called BOLT (Basis of Lightning Technology) [5], and had its first transaction on May 10, 2017. As of February 10, 2023, it had approximately 16,000 active nodes, 75,000 active channels, and held more than 5,300 bitcoins [6]. This node infrastructure produces approximately 1.4 million tonnes of CO_2, or to put it another way, 0.0033 grams of CO_2 per year per transaction [10], and thus has a significant negative impact on the environment.

There are three predominant routing protocols and related algorithms that LN uses for routing multi-hop payments – LND [4], c-Lightning (CLN) [1] and Eclair (ECL) [2], they determine the payment paths that should have low locktimes, low fees and high probabilities for successful payments. The LN is a sparse and highly distributed network [43] and these routing algorithms can potentially find paths that are similar from the payment sender's perspective, but the algorithms themselves have different performance, computational costs, energy consumption, and ultimately different CO_2 emissions per step in the pathfinding phase.

Our main contribution is a "one-shot" path planning Reinforcement Learning (RL) based approach (RLA), which is able to predict a possible payment path of limited length with better performance. We empirically evaluate this approach on a real snapshot of the LN and estimate running times, energy consumption, and CO_2 emissions.

1.1 The pathfinding task in the Lightning Network

The LN consists of nodes and each of them can ask any other node to open a bidirectional payment channel by sending a specially formed funding transaction to the Bitcoin blockchain [31] (Figure 1). The funding transaction should specify the amount of money that each node locks in the channel and the information needed to identify participating nodes.

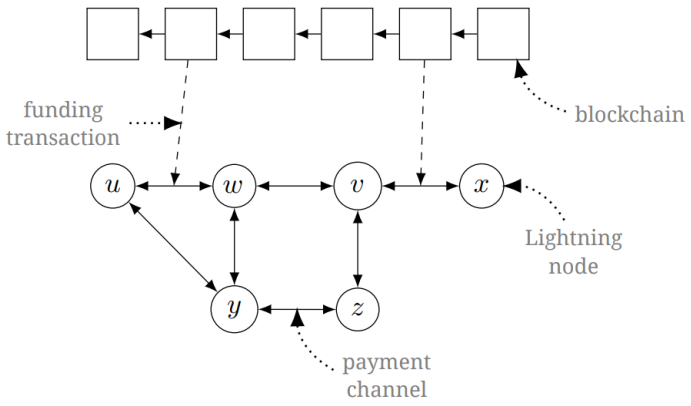

Figure 1: Payment architecture of the Lightning Network [32]

A node can send a payment to another party in the LN by transferring value through the route of channels. The sender then decides on the payment route/path to the recipient. To do this, they need to

Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), Cruz, Hayes, Wang, Yates (eds.), May 29-30, 2023, London, UK, https://alaworkshop2023.github.io/. 2023.

know about all public nodes and channels, known as the graph [3]. Nodes would take forwarding fees to transfer payments through their channels, which are expected to be quite small in most cases.

All nodes in the network would be able to act as senders or recipients of payments and as hubs that route payments between other nodes at the same time. At the moment, some members of the community believe [29] that LN could only exist as a hub-and-spoke network [7, 23], as channels created by small nodes would be unlikely to provide enough liquidity for LN to operate efficiently [32].

The pathfinding algorithms are not part of the BOLT specification, and it is up to each implementation/node to use what they see fit in order to find the shortest and most efficient path, with the information provided by the gossip protocol. Deterministic pathfinding algorithms typically treat the network like a graph, with each route/edge between nodes having a unique cost, instead of a distance.

In addition to two separate fee structures for each channel (base fee and fee rate), pathfinding in the LN is further complicated by the channels' capacity and their liquidity. In practice, this means each node will create multiple possible paths to its destination, and try them successively. For instance, LND ranks nodes it has successfully sent payments through to improve its pathfinding [3].

1.2 The Lightning Network topology and dynamic

The LN has a small diameter (< 10) with few connected components (< 10) and an average shortest path of 3 [11, 37]. The Adopting Bitcoin Conference and LNBIG.com reported that in October 2021, 43% of nodes might be reached by two hops, 93% of nodes by 3 hops, 98.8% of nodes by 4 hops, 99.9% of nodes after 5 hops [38]. The simulation [11] and empirical tests [17] also showed that, on average, 96-98% of payments are shorter than those with 5 hops.

General statistics show that, on average, LN adds 4-5% of new nodes and 10-15% of new channels per month, with most of these channels replacing old ones [6]. The average lifetime of a node exceeds 100 days, the average lifetime of a channel exceeds 38 days [11, 43]. Field studies show that the percentage of cut nodes has trended downwards since August 2020, is currently at approximately 8% [17]. The percentage of cut channels has also been on a downward trend since May 2021 and currently stands at 10% [17].

Arcane Research estimated that in September 2021, LN processed about 663,000 transactions with only commonly used wallet platforms [17]. They also noted that payments often rely on efficient routing nodes - hubs that are stable and integral for the network, allowing users to transfer funds and interact with the ecosystem without opening unique channels for each recipient. If we agree that the number of such hubs is about 10-20% [23] of the total number of nodes, then some of them might perform more than 200 transactions per month. This estimate does not take into account other payment sources, developer activity, and most b2b transactions. Interestingly, Arcane Research predicts that in the next 7 years, LN will add at least 700 million users paying for games, video and audio, which will lead to an increase in the number of transactions.

Thus, we can assume that an efficient network agent performs most transactions in its neighborhood of relatively small size L and is able to find a path of length $m \ll 10$, which is a case for a one-shot solution based on RL.

2 BACKGROUND AND PREREQUISITES

2.1 Reinforcement Learning and Plan-Based Reward Shaping

A task in RL is modeled as a Markov Decision Process (MDP). An MDP M is defined as a tuple $M = (S, A, T, R, \gamma)$, where S and A are the state and action space respectively. $T(s, a, s') : S \times A \times S \to [0, 1]$ is the probability of reaching state s' from state s after executing action a. The reward function $R(s, a, s') : S \times A \times S \to R$ assigns a numerical reward to a state transition from s to s' with respect to the executed action a. A policy $\pi(s, a) : S \times A \to [0, 1]$ defines how the agents should act in the environment through the probability distribution over all actions in every state. The decay factor $\gamma \leq 1$ is used to define the expected discounted return $D_t = \sum_{\infty\, k=0}^{\infty} \gamma^k R_{t+k}$ and the value function $V(s) = E_{A_t \sim \pi(St)}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s]$. The RL agent learns a policy that maximizes the expected discounted return, where the optimal policy has the maximum expected discounted return [40].

Long sequence, sparse reward RL tasks terminate with a reward of one once a sequence of specific high-level actions has been fulfilled. For all other transitions the agent gets a reward of zero. This also refers to the pathfinding task, where the agent receives no reward until he reaches his destination. This lack of reward feedback creates very challenging tasks for RL agents due to the difficulties of assigning the reward correctly to the important parts within a long sequence of actions.

Reward shaping can help solve sparse reward tasks by utilizing external knowledge to add intermediate feedback into the reward function. Reward shaping creates a new shaped reward function $R'(s, a, s') = R(s, a, s') + F(s, a, s')$ [16, 28].

The shaping function should modify the reward function to simultaneously incentivize reaching a new subgoal and penalize remaining at the same step in the plan. Additionally the total reward for reaching the goal is required to stay positive. The step count is greater or equal to zero in any non-terminal state for plan-based reward shaping. The exponential plan-based reward allows for both the negative on-step rewards and the positive on-subgoal rewards to increase proportionally with the number of fulfilled steps in the plan [16]:

$$F(s, a, s') = \begin{cases} 0, & \text{if } s \text{ is a terminal state,} \\ R_{goal} \frac{b^{step(s)}}{b^{|P|}}, & \text{otherwise} \end{cases}$$

where R_{goal} – the final reward, and $b > 1$ is the constant base parameter, the step function $step()$ is used to track the number of steps in the plan achieved at the current state, and $|P|$ is the total number of steps in the plan P. Whenever a new landmark (an appropriate step) is reached, the agent gets an exponentially larger reward.

2.2 Proximal Policy Optimization

The Proximal Policy Optimization (PPO) is a family of policy optimization methods that use multiple epochs of stochastic gradient ascent to perform each policy update. These methods have the

stability and reliability of trust-region methods but have better overall performance and applicable in more general settings [36]. There is strong evidence that PPO based agents are able to solve pathfinding tasks and better at adjusting such a heuristic [9, 39]. So, we employed Stable-Baselines3 PPO [8] and the OpenAI Gym goal-based environment [12] with mostly default hyperparameters (see § 4.5).

3 RELATED WORK

As for LN, there have been several recent studies that consider its topological properties [24, 33, 37, 41, 43] as well as the pathfinding task using native routing algorithms [21]. Recently, various deterministic algorithms for LN and their enhancements have been proposed: Flare [32], SilentWhispers [26], SpeedyMurmurs [34] and Flash [42], etc. They take into account network dynamics, privacy and anonymity, success rate, and execution performance.

Reinforcement learning has also been applied to the pathfinding/routing problem in various classes of networks, from wired and static networks to very dynamic wireless and ad hoc networks. Dozens of RL-based protocols have been proposed in a quarter century (see review [27]). In some cases, they surpass Dijkstra's algorithms [13].

To our knowledge, only one paper has introduced an RL-based solution for LN, which proposes a learning environment for making simultaneous multi-channel payments and achieving efficient fee setting [9]. They proposed an environment and trained an agent to find a combination of channels around some node in order to maximize the profit. However, this solution does not solve the path finding task itself, but use the built-in Dijkstra' algorithm for the cheapest path selection and transaction simulation. Thus, this is not suitable for an empirical comparison to our proposed method in terms of energy efficiency during the pathfinding phase.

The general idea of one-shot path planning for 2D and 3D small environments using a convolutional neural network was recently presented in [19]. However, this cannot be used directly for higher-dimensional environments, such as a network with many neighbors per node. The average degree of LN is rarely below 6 [11], in our snapshot it is 20.

Thus, this paper presents to the best of our knowledge the first attempt focusing on RL performance and CO_2 emissions in the case of blockchain-related networks. In addition, it introduces the novel approach of RL one-shot path prediction.

4 DATA AND METHOD

4.1 Environment and reward function

In our implementation, each episode is one transaction from u to v that may be successfully performed using any path P that belongs to LN, represented as a graph G. The initial state $S_{initial}$ is a vector with size of n, which encodes the state of the neighborhood. In a very simple implementation, $n = L$ and the initial state is simply a binary encoding that has 1 for the position associated respectively with u and v, 0 otherwise. Similarly, the final state S_{final} can represent any possible path $P \subset G$ from u to v as a binary encoding of the neighborhood. A complicated but probably more practical way is to use some fast graph embeddings that can handle large LN-graphs and representations with large neighborhood size.

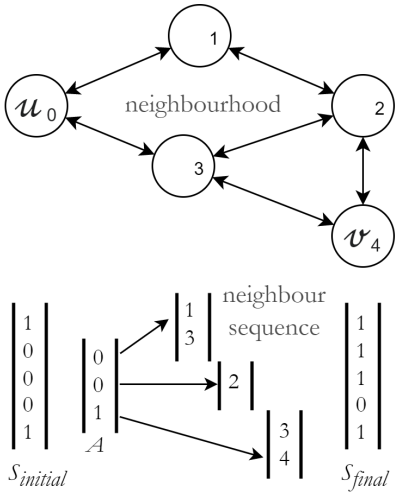

Figure 2: The state and the action mapping example

The action A represents any path P of length m as a sequence of a_i, that can be mapped to the subset of nearest nodes in the u neighborhood of size L:

$$A = \{a_i | a_i \in [0, 1], 0 \leq i < m\},$$

$$map(A, G|L) = \{p_i | p_i = a_i \rightarrow \{u, ..., v\} \subseteq G|L, a_i \in A\} = P, |P| = m.$$

In short, the agent performs action A, which is then sequentially mapped $map()$ to the next available neighbor, starting with u (Figure 2). If the path composed in this way contains a destination node v (a final state), the agent receives a positive reward.

As a baseline we implemented a goal based environment with the final reward R_{goal}, which means an agent gets +10 when the path P it predicts is an applicable path from node u to v, and 0 otherwise (any inapplicable path is also a terminal state). Since the agent is able to predict paths with maximum length m, the final path is obtained by pruning immediately after it reaches the destination v.

Then this base reward was then adjusted with the total path cost C_{total} to induce the agent to find an effective path in terms of the LN cost function. We used normalized LND cost function that has a bias against channels with known recent failures:

$$cost[u, v] = \frac{amount[u, v] \cdot delay[u, v] \cdot \rho + fee[u, v] + bias[u, v]}{c},$$

$$C_{total} = \sum_{[u,v] \in P} cost[u, v],$$

here ρ – is a risk factor set to $15 \cdot 10^{-9}$ by default [21] and $bias[u, v]$ accounts for previous payment failures caused by the channel $[u, v]$; c – the basis of normalization, $c = 1000$ [21]. The value of $bias[u, v]$ is extremely large during the first hour after failure and decreases exponentially with every hour elapsed after the last failure.

Finally, we added an immediate exponential reward bonus $k_{bonus} \cdot R_{goal}$ as a reward shaping procedure, (see § 2.1) that encourages the agent for each step that lie on the Dijkstra's planned trajectory $P_{planned}$. This modification allows the agent to quickly discover the shortest path, if it exists. This trick was observed to be more effective than a simple additive bonus in the case of agent training for sufficiently large and sparse networks. Thus,

$$R(s, s') = \begin{cases} \gamma k_{bonus} \cdot R_{goal}, & \text{if } s' \text{ is not a final state,} \\ R_{goal} - C_{total} + \gamma k_{bonus} \cdot R_{goal}, & \text{otherwise} \end{cases}$$

$$k_{bonus} = \frac{b^{|\{p|p \in P \wedge p \in P_{planned}\}|}}{b^{|P_{planned}|}}$$

here γ – the decay factor, and b – the scale parameter that were set 0.99 and 32, respectively, based on [18].

4.2 Energy consumption and emissions measurement

To calculate the relative energy consumption and corresponding CO_2 emissions for native algorithms and our solution, we use an approach that automatically measures CPU/GPU/RAM consumed power, and compares the energy mixes used in the power grids of different countries [25]. In the case of an unknown location, it uses the default world average of energy mix and carbon dioxide emissions. This global average energy mix consists of 28.7% coal, 22.9% oil, 33.9% natural gas, and 14.4% low-carbon fuels with the resulting global average CO_2 emissions of electricity of 726 kg CO_2 per MWh.

While this solution combines the most commonly used techniques (e.g., RAPL) [30], the advantage over other solutions is that this way of coding is simpler [15].

4.3 Data and preparation

We utilized data snapshots [14] dated between 14/10/2020 and 23/08/2022 that were widely described in [43]. Such snapshots contain network messages in the LN collected over a certain period and generated by the gossiping communication mechanism. Based on this data and with the Python NetworkX library, we replayed over 320,000 channel updates and reconstructed a real network topology around the last available timestamp. Nodes either without established or with deprecated channels were excluded because such nodes technically cannot participate in transactions. For simplicity, we also aggregated channels into undirected edges.

To test the approach we sampled six random nodes and selected its neighborhoods of size $L \in [50, 100]$ by using the ForestFire sampling method [35]. This is a stochastic snowball sampling method, which better represents both static properties and evolutionary patterns of the whole network [22]. According to § 1.2, there is a reliable guarantee that a sampled neighborhood with representative connectivity and a radius of less than five is able to cover possible payment recipients from neighborhood centroid with 99.9% probability, especially in the case of a payment hub. In the unlikely case that the sample does not include a path to the destination node, a re-sampling is triggered.

To compare native algorithms with our approach we sequentially generated a set of transactions from these six centroid nodes to each node in their own neighborhood and then executed the pathfinding.

4.4 Implementation of native LN pathfinding algorithms

As mentioned above, there are three main LN clients with built-in solutions based on Dijkstra' or Yen' algorithms (LND, CLN, and ECL). We performed our experiments on its Python implementation, which was introduced in [20, 21] and adapted to the Python NetworkX library.

4.5 Experiment details and hyperparameters

We used the same setup for the additional training and the experiments and employed default hyperparameters for the Stable-Baselines3 PPO implementation [8]: policy type: actor-critic; timesteps per epoch: 100,000 for general training, 10,000 for experiments and additional training; learning rate: 0.0003; discount factor (γ): 0.99; GAE parameter (λ): 0.95; clipping parameter: 0.2; value function coefficient: 0.5; maximum value for the gradient clipping: 0.5. Figure 3 shows that this approach is able to converge.

Hardware setup for general training. CPU: AMD EPYC 7662 64-Core Processor, 256 CPUs; GPU: 2 x A100-PCIE-80GB; 1 TB RAM, 1007.764 GB available; Platform system: Linux-5.10.0-15-amd64-x86_64-with-glibc2.31; Python version: 3.10.6.

Hardware setup for experiments and additional training. CPU: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 8 CPUs; GPU: 1 x NVIDIA GeForce MX350; 8 GB RAM, 7.675 GB available; Platform system: Windows-10-10.0.22621-SP0; Python version: 3.9.13.

5 EXPERIMENTS AND RESULTS

In this section we empirically evaluate our approach in comparison to native LN pathfinding algorithms and discuss the obtained results. Through our experiments, we want to show that one-shot RL path prediction can be performed quickly and with a lower energy consumption in the case of a relatively small neighborhood. The following results were averaged over six agents, each of which was trained on a different transaction set, i.e. on different neighborhoods. The results for each agent were averaged over L runs, where L is the size of the neighborhood.

5.1 Performance and emissions

Intuitively, since an agent generates a complete path by taking just one action, it should take constant execution time in an environment of any size. Whereas algorithms based on the Dijkstra method take increasingly longer to execute because of their complexity. The average runtime of the algorithms that were executed on sets of transactions in all experimental neighborhoods is shown in Figure 4.

A large neighborhood makes deterministic algorithms spend more time searching for the optimal path, and the longer the path, the longer it takes, see Figure 5. A well-trained agent is able to predict the applicable path in a fixed time, but in some cases it may be sub-optimal in terms of length (Figure 6).

Since these algorithms can use the CPU/GPU and RAM in different ways, they consume different amounts of energy and produce

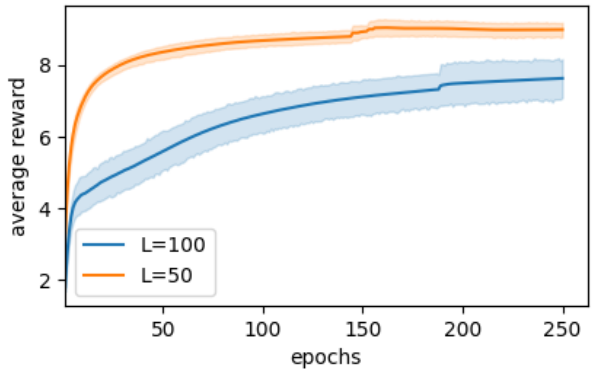

Figure 3: The ability to converge. The average reward per episode was averaged over a set of transactions that were introduced randomly during training. These results were averaged over 6 agents we trained on their neighborhoods separately. The results for each agent are an average over L runs. The shaded area under the curves represents the 95% confidence intervals for the average value.

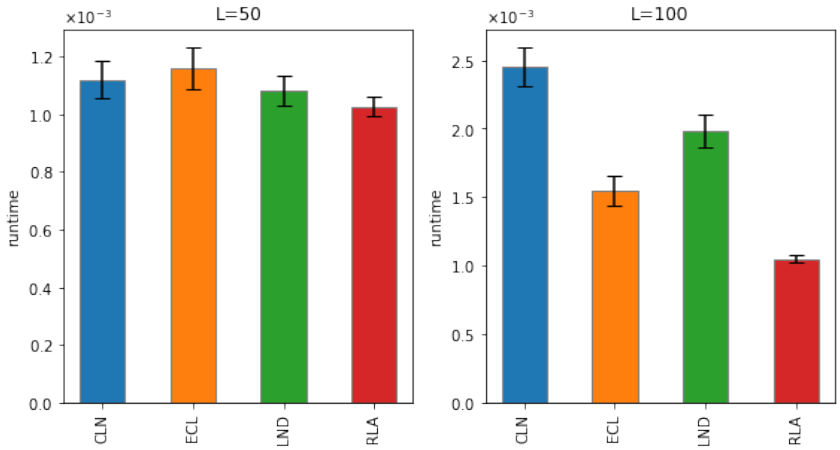

Figure 4: Pathfinding performance estimates per transaction, [sec]. The average runtime of the algorithms that were executed on sets of transactions over all chosen neighborhoods. The RLA represents average results of 6 agents we trained on their neighborhoods separately. The results for each agent are an average over L runs.

different amounts of CO_2 pollution accordingly, see Figure 7. There is a fairly strong correlation between execution time and the total CO_2 emission that can be produced, thus RLA also achieves good performance.

5.2 Sensitivity and robustness

In practice, when the environment changes significantly, the agent becomes confused and predicts the path with less confidence. To test the sensitivity of RLA to a certain number of changes in the graph, we randomly remove a certain number of edges from the graph and estimate the loss of the success rate and the need for additional training at a time (Figure 8). The success rate shows the relative number of transactions that were successfully routed by the agent in their neighborhood. In our implementation, the agent is able to achieve a 96% success rate if there is no significant change in its neighborhood.

Since most significant changes in LN do not practically exceed 10% (see § 1.2), we believe that rapid additional training is an appropriate solution. We did some light training with average epoch duration 11.2 sec, average emissions per epoch of $5.84 \cdot 10^{-6}$ kg CO_2 eq. and this seems to be enough to practically overcome the changes in the neighborhood. It can be seen that with small changes, an agent can return over 90% of its success rate after 100 epochs of additional training (Figure 8). Note that Dijkstra's routing algorithms can find a route for 98% of transactions, based on our tests, but the percentage of successful payments in LN cannot be accurately estimated [38].

6 CONCLUSION AND FUTURE WORK

In this paper we present a one-shot path prediction and propose an RL solution for a pathfinding agent in LN that is able to converge

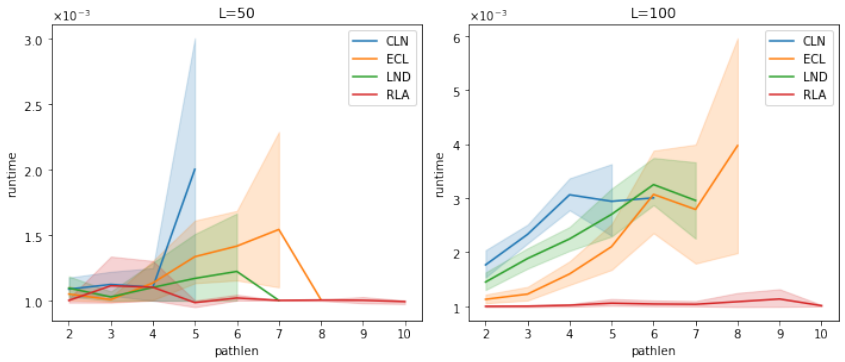

Figure 5: Pathfinding performance estimates by path-length, [sec]. The average runtime of the algorithms that were executed on sets of transactions over all chosen neighborhoods. The RLA represents average results of 6 agents we trained on their neighborhoods separately. The results for each agent are an average over L runs. The shaded area under the curves represents the 95% confidence intervals for the average value.

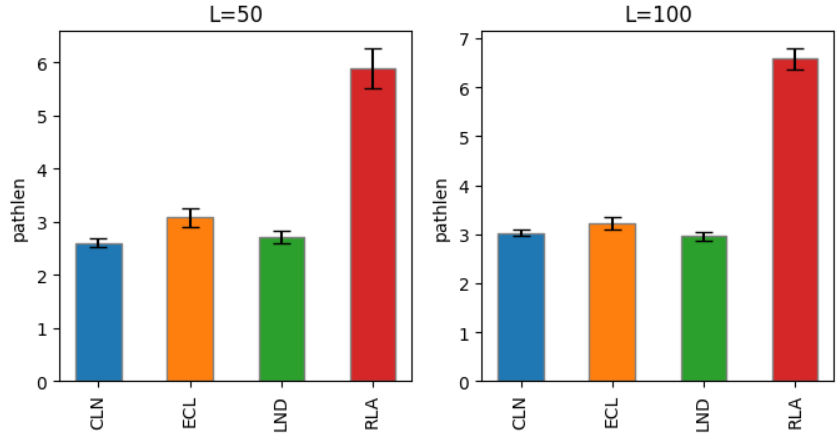

Figure 6: Average path-length. Averaged over the algorithms that were executed on sets of transactions over all chosen neighborhoods. The RLA represents average results of 6 agents we trained on their neighborhoods separately. The results for each agent are an average over L runs.

and learn its neighborhood. In our experiments, we have empirically shown that RLA:

(1) achieves good performance in terms of energy consumption at each pathfinding step;
(2) can outperform Dijkstra's routing algorithms in the case of an increasing number of transactions in relatively small neighborhoods, which is sufficient for the case of LN;
(3) can update the heuristic with less execution cost and CO_2 emissions in case of limited neighborhood change compared to full retraining from scratch.

Since the learning time and associated CO_2 emissions depend on the size of the state and the available maximum neighborhood size, a sparse state representation based on binary encoding can lead to a potential scalability limitation. This does not affect the idea of our approach or its practical implications, and we believe that using some fast graph embeddings will overcome this, allow one to handle larger LN-graphs and representations with larger neighborhood size more efficiently.

For future work, we also consider looking at this problem through the lens of multi-objective optimization: designing path planners that can balance several tradeoffs such as lock times, low fees, high probability of successful payments, and low energy consumption.

ACKNOWLEDGMENTS

The authors would like to thank Dr.-Ing. Manfred Veenker and the Veenker Foundation for its support, and Ildar Baimuratov, Konstantin Glonin, and Yuan Xue for their constructive and valuable discussions and guidance during the planning and development of this research.

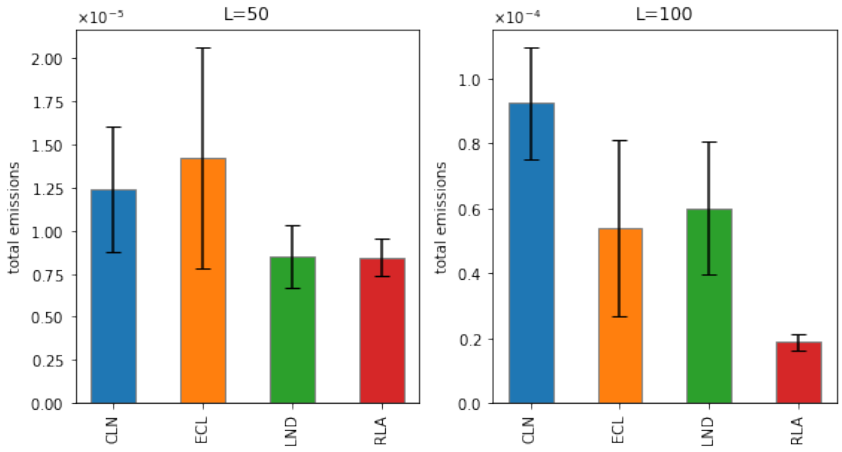

Figure 7: Average total emissions per transaction, [kg CO_2 eq.]. The average total emissions of the algorithms that were executed on sets of transactions over all chosen neighborhoods. The RLA represents average results of 6 agents we trained on their neighborhoods separately. The results for each agent are an average over L runs.

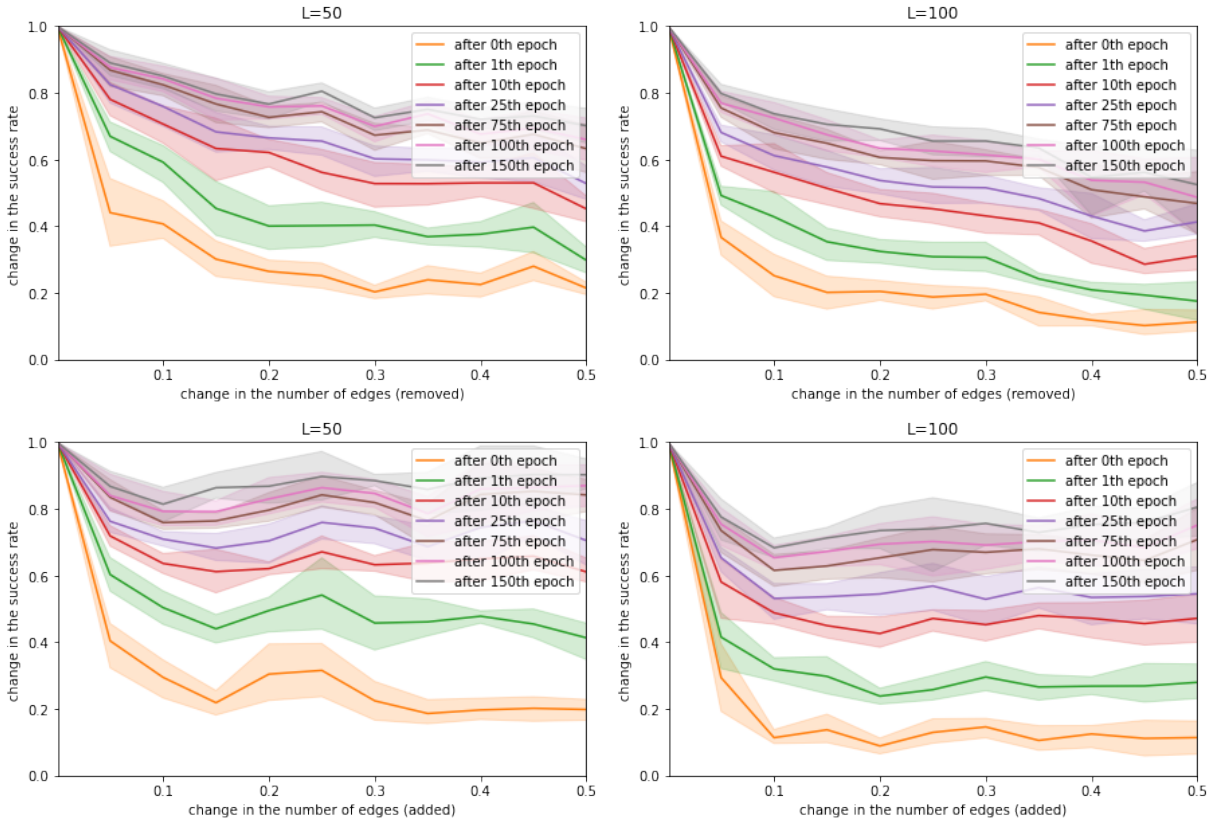

Figure 8: Comparative need for additional training when changing the network. The X-axis shows the relative number of edges that have been randomly removed/added from/in each neighborhood. The success rate means the relative number of transactions that were successfully routed by the agent in the neighborhood at a given level of change and after a certain epoch of additional training. The Y-axis shows the relative change in the success rate at a given level of change. It shows average results of 6 agents we trained on their neighborhoods separately. The results for each agent are an average over L runs. The shaded area under the curves represents the 95% confidence intervals for the average value.

REFERENCES

[1] [n.d.]. Core Lightning (CLN): A specification compliant Lightning Network implementation in C. GitHub. Retrieved February 22, 2023 from https://github.com/ElementsProject/lightning

[2] [n.d.]. Eclair (French for Lightning) is a Scala implementation of the Lightning Network. GitHub. Retrieved February 22, 2023 from https://github.com/ACINQ/eclair

[3] [n.d.]. Finding routes in the Lightning Network. Builder's Guide. Retrieved February 22, 2023 from https://docs.lightning.engineering/the-lightning-network/pathfinding/finding-routes-in-the-lightning-network

[4] [n.d.]. The Lightning Network Daemon. GitHub. Retrieved February 22, 2023 from https://github.com/lightningnetwork/lnd

[5] [n.d.]. Lightning Network In-Progress Specifications. GitHub. Retrieved February 22, 2023 from https://github.com/lightning/bolts

[6] [n.d.]. Real-Time Lightning Network Statistics. JSON. Retrieved February 22, 2023 from https://1ml.com/statistics

[7] [n.d.]. Spoke-hub distribution paradigm. Wikipedia. Retrieved February 22, 2023 from https://en.wikipedia.org/wiki/Spoke%E2%80%93hub_distribution_paradigm

[8] 2021. The Proximal Policy Optimization algorithm. Stable Baselines3 Library. Retrieved February 22, 2023 from https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

[9] Kiana Asgari, Aida Afshar Mohammadian, and Mojtaba Tefagh. 2022. DyFEn: Agent-Based Fee Setting in Payment Channel Networks. https://doi.org/10.48550/ARXIV.2210.08197

[10] Oliver Barratt and Danny Scott. 2021. Comparing Bitcoin Lightning energy usage to the real world. Retrieved February 22, 2023 from https://blog.coincorner.com/comparing-bitcoin-lightning-energy-usage-to-the-real-world-2d64c62b1783

[11] Ferenc Beres, Istvan Andras Seres, and Andras A. Benczur. 2019. A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network. https://doi.org/10.48550/ARXIV.1911.09432

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[13] Haoran Miao Babar Shah Bashir Hayat Imran Khan Tae-Eung Sung Ki-Il Kim Daniel Godfrey, Beom-Su Kim. 2021. Q-Learning Based Routing Protocol for Congestion Avoidance. *Computers, Materials & Continua* 68, 3 (2021), 3671–3692. https://doi.org/10.32604/cmc.2021.017475

[14] Christian Decker. [n.d.]. Lightning Network Research ; Topology Datasets. https://github.com/lnresearch/topology. https://doi.org/10.5281/zenodo.4088530

[15] Ralf Gitzel. 2022. Software Tools to Determine the Carbon Footprint of AI Code. Retrieved February 22, 2023 from https://www.linkedin.com/pulse/software-tools-determine-carbon-footprint-ai-code-ralf-gitzel/

[16] Marek Grzes and Daniel Kudenko. 2008. Plan-based reward shaping for reinforcement learning. *2008 4th International IEEE Conference Intelligent Systems* 2 (2008), 10–22–10–29.

[17] Anders Helseth. 2021. The State of Lightning. Arcane Research Report. Retrieved February 22, 2023 from https://arcane.no/research/the-growth-of-the-lightning-network

[18] Lukas Berg Henrik Müller and Daniel Kudenko. 2023. Using Incomplete and Incorrect Plans to Shape Reinforcement Learning in Long-Sequence Sparse-Reward Tasks. In *Proc. of the Adaptive and Learning Agents Workshop (ALA 2023) at AAMAS 2023, May 29-30* (London, UK) *(ALA 2023)*. Cruz, Hayes, Wang, Yates (eds.), London, UK. https://alaworkshop2023.github.io/

[19] Tomas Kulvicius, Sebastian Herzog, Timo Lüddecke, Minija Tamosiunaite, and Florentin Wörgötter. 2020. One-shot path planning for multi-agent systems using fully convolutional neural network. (2020). https://doi.org/10.48550/ARXIV.2004.00568

[20] Satwik Prabhu Kumble, Dick Epema, and Stefanie Roos. 2021. How Lightning's Routing Diminishes Its Anonymity. In *Proceedings of the 16th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES 21)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/3465481.3465761

[21] Satwik Prabhu Kumble and Stefanie Roos. 2021. Comparative Analysis of Lightning's Routing Clients. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. 79–84. https://doi.org/10.1109/DAPPS52256.2021.00014

[22] Jure Leskovec and Christos Faloutsos. 2006. Sampling from Large Graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Philadelphia, PA, USA) *(KDD '06)*. Association for Computing Machinery, New York, NY, USA, 631–636. https://doi.org/10.1145/1150402.1150479

[23] Jian-Hong Lin, Emiliano Marchese, Claudio J. Tessone, and Tiziano Squartini. 2022. The weighted Bitcoin Lightning Network. *Chaos, Solitons &amp Fractals* 164 (nov 2022), 112620. https://doi.org/10.1016/j.chaos.2022.112620

[24] Jian-Hong Lin, Kevin Primicerio, Tiziano Squartini, Christian Decker, and Claudio J Tessone. 2020. Lightning network: a second path towards centralisation of the Bitcoin economy. *New Journal of Physics* 22, 8 (aug 2020), 083022. https://doi.org/10.1088/1367-2630/aba062

[25] Kadan Lottick, Silvia Susai, Sorelle A. Friedler, and Jonathan P. Wilson. 2019. Energy Usage Reports: Environmental awareness as part of algorithmic accountability. https://doi.org/10.48550/ARXIV.1911.08354

[26] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2016. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks. Cryptology ePrint Archive, Paper 2016/1054. https://eprint.iacr.org/2016/1054 https://eprint.iacr.org/2016/1054.

[27] Zoubir Mammeri. 2019. Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches. *IEEE Access* 7 (2019), 55916–55950. https://doi.org/10.1109/ACCESS.2019.2913776

[28] A. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *International Conference on Machine Learning*.

[29] Chris Pacia. 2015. Lightning Network skepticism. Retrieved February 22, 2023 from https://chrispacia.wordpress.com/2015/12/23/lightning-network-skepticism

[30] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon Emissions and Large Neural Network Training. https://doi.org/10.48550/ARXIV.2104.10350

[31] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Retrieved February 22, 2023 from https://lightning.network/lightning-network-paper.pdf

[32] Pavel Prihodko, S. N. Zhigulin, Mykola Sahno, A B Ostrovskiy, and Olaoluwa Osuntokun. 2016. Flare : An Approach to Routing in Lightning Network White Paper. Retrieved February 22, 2023 from https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf

[33] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 2019. Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks. https://doi.org/10.48550/ARXIV.1904.10253

[34] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2017. Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions. https://doi.org/10.48550/ARXIV.1709.05748

[35] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Little Ball of Fur: A Python Library for Graph Sampling. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 3133–3140.

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. https://doi.org/10.48550/ARXIV.1707.06347

[37] István András Seres, László Gulyás, Dániel A. Nagy, and Péter Burcsi. 2019. Topological Analysis of Bitcoin's Lightning Network. https://doi.org/10.48550/ARXIV.1901.04972

[38] Bobby Shell. 2022. How many transactions can the Lightning Network handle? Retrieved February 22, 2023 from https://voltage.cloud/blog/bitcoin-education/how-many-transactions-can-the-lightning-network-handle

[39] Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr Panov. 2022. Pathfinding in stochastic environments: learning vs planning. *PeerJ Computer Science* 8 (2022), e1056.

[40] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

[41] Saar Tochner, Stefan Schmid, and Aviv Zohar. 2019. Hijacking Routes in Payment Channel Networks: A Predictability Tradeoff. https://doi.org/10.48550/ARXIV.1909.06890

[42] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: Efficient Dynamic Routing for Offchain Networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (Orlando, Florida) *(CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 370–381. https://doi.org/10.1145/3359989.3365411

[43] Philipp Zabka, Klaus-T. Foerster, Stefan Schmid, and Christian Decker. 2022. Empirical evaluation of nodes and channels of the lightning network. *Pervasive and Mobile Computing* 83 (2022), 101584. https://doi.org/10.1016/j.pmcj.2022.101584