

University of Kentucky

UKnowledge

Theses and Dissertations--Computer Science

Computer Science


2022

Learning a Scalable Algorithm for Improving Betweenness in the Lightning Network

Vincent Davis

University of Kentucky, vincentmdavis@protonmail.com

Author ORCID Identifier:

 <https://orcid.org/0000-0001-7336-6378>

Digital Object Identifier: <https://doi.org/10.13023/etd.2022.432>

[Right click to open a feedback form in a new tab to let us know how this document benefits you.](#)

Recommended Citation

Davis, Vincent, "Learning a Scalable Algorithm for Improving Betweenness in the Lightning Network" (2022). *Theses and Dissertations--Computer Science*. 123.
https://uknowledge.uky.edu/cs_etds/123

This Master's Thesis is brought to you for free and open access by the Computer Science at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Computer Science by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Vincent Davis, Student

Dr. Brent Harrison, Major Professor

Dr. Simone Silvestri, Director of Graduate Studies

LEARNING A SCALABLE ALGORITHM FOR IMPROVING BETWEENNESS
IN THE LIGHTNING NETWORK

THESIS

A thesis submitted in partial
fulfillment of the requirements for
the degree of Master of Science in
the College of Engineering at the
University of Kentucky

By
Vincent M. Davis
Lexington, Kentucky

Director: Dr. Brent Harrison, Professor of Computer Science
Lexington, Kentucky
2022

Copyright© Vincent M. Davis 2022

ABSTRACT OF THESIS

LEARNING A SCALABLE ALGORITHM FOR IMPROVING BETWEENNESS IN THE LIGHTNING NETWORK

This paper presents a scalable algorithm for solving the Maximum Betweenness Improvement Problem as it occurs in the Bitcoin Lightning Network. In this approach, each node is embedded with a feature vector whereby an Advantage Actor-Critic model identifies key nodes in the network that a joining node should open channels with to maximize its own expected routing opportunities. This model is trained using a custom built environment, *lightning-gym*, which can randomly generate small scale-free networks or import snapshots of the Lightning Network. After 100 training episodes on networks with 128 nodes, this A2C agent can recommend channels in the Lightning Network that perform competitively with recommendations from centrality based heuristics and in less time. This approach provides a fast, low resource, algorithm for nodes to increase their expected routing opportunities in the Lightning Network.

KEYWORDS: Bitcoin, Betweenness, Reinforcement Learning, Graph Networks

Vincent M. Davis

November 12, 2022

LEARNING A SCALABLE ALGORITHM FOR IMPROVING BETWEENNESS
IN THE LIGHTNING NETWORK

By
Vincent M. Davis

Dr. Brent Harrison

Director of Thesis

Dr. Simone Silvestri

Director of Graduate Studies

November 12, 2022

Date

TABLE OF CONTENTS

List of Tables	v
List of Figures	vi
Chapter 1 Introduction	1
Chapter 2 Background	4
2.1 The Lightning Network	4
2.1.1 Channels	4
2.1.2 Liquidity	5
2.1.3 Routing Nodes	5
2.1.4 Fee Policy	5
2.1.5 Hash Time Lock Contracts	6
2.1.6 Network Model	6
2.1.7 Attachment Strategies	6
2.2 Betweenness Centrality	7
2.3 Reinforcement Learning	8
Chapter 3 Related Works	9
3.1 Attachment Strategies	9
3.1.1 Greedy Algorithm	9
3.1.2 Centrality-based Selection	10
3.2 Graph Embedding Techniques	11
3.3 Lightning Network Simulators	13
3.3.1 CLoTH	13
3.3.2 LNTrafficSimulator	14
3.3.3 TimeMachine	14
Chapter 4 Methods	15
4.1 Graph Representation	15
4.2 Markov Decision Process Formulation	16
4.3 Actor-Critic Model	17
4.3.1 Policy Network	17
4.3.2 Value Network	17
4.4 Training Architecture	18
4.4.1 Network Training	19
Chapter 5 Experiments	20
5.1 Baselines	20
5.2 lightning-gym	20
5.3 Experiment 1 - Setup	21

5.4	Experiment 2 - Setup	21
5.5	Experiment 3 - Setup	21
Chapter 6	Results	23
6.1	Experiment 1 - Results	23
6.2	Experiment 2 - Results	25
6.3	Experiment 3 - Results	27
Chapter 7	Discussion	29
7.1	Performance	29
7.2	Scalability	30
7.3	Training	30
7.4	Future Work	31
Chapter 8	Conclusion	33
	Bibliography	34
	Vita	37

LIST OF TABLES

6.1	Comparison of performances on the Lightning Network.	24
6.2	Change in Number of Nodes and Edges after pruning	26
6.3	Comparison of runtimes for each algorithm.	27

LIST OF FIGURES

2.1	A fixed transaction size is applied to all fee policies in order to simplify the network.	7
4.1	Advantage Actor-Critic Architecture	17
4.2	Training architecture. A2C accepts as input a graph embedding from the GCN and then takes an action in the environment, producing a new state-reward pair.	18
6.1	Performance comparison on a synthetic network with 128 nodes.	23
6.2	Performance comparison on a synthetic network with 1024 nodes.	23
6.3	Comparison of the performances of A2C agent, Betweenness, Degree, and Random on instances of the Lightning Network. Each algorithm starts with an isolated node and a budget of 10 channels.	24
6.4	Change in number of nodes after removing bridges, smaller connected components, and low degree nodes.	25
6.5	Change in number of channels after removing private, inactive, and low capacity edges.	25
6.6	Comparison of the runtimes of the five algorithms for each month.	26
6.7	Performance of agent trained on different random scale free networks with 128 nodes and a budget $k = 10$	27
6.8	Performance of agent repeatedly trained on single instance of a random scale free network with 128 nodes and a budget $k = 10$	28

Chapter 1 Introduction

Bitcoin is a peer-to-peer electronic cash system whose defining characteristics are its decentralized ledger, proof-of-work consensus model, and fixed supply cap[2]. These features combined allow anyone to exchange value without the need for a trusted third party. There is no central server to coordinate Bitcoin transactions. Instead, transactions are added to the decentralized ledger through a process known as mining. Transactions are broadcast and picked up by miners who add them to an “ongoing chain of hash-based proof-of-work” [22]. This process involves solving a computationally difficult problem that can be easily verified by any participant in the network. The chain that exhibits the greatest amount of computational work is accepted as the “true” chain of events. To change this history of transactions would require an attacker to own a majority of the computational power of the network. Miners are incentivized to process transactions by earning block rewards and mining fees.

Since its inception in 2008, Bitcoin has reached a magnitude in both use and value that it has become more economical to defer finalization of exchange between individuals. The mining fees associated with transactions are calculated based on the volume of *data* consumed on-chain, not by the amount of funds involved, making small payments expensive. Furthermore, transaction throughput on the Bitcoin blockchain is limited by design. Bitcoin has a fixed blocksize and blocktime which limits the speed of the network to processing on average one 4MB block every ten minutes or a maximum of seven transactions per second [22]. The design decision to have a fixed blocktime and blocksize comes with a tradeoff: it keeps Bitcoin’s storage requirements accessible, but it also limits scalability of Bitcoin’s blockchain as a P2P electronic cash system. When considering the time, mining fees, and energy involved, common use cases such as micropayments, subscriptions, and streaming payments are expensive to perform on the base layer of the Bitcoin blockchain.

The Lightning Network is a Layer 2 payment protocol built on top of the Bitcoin blockchain meant to answer its scalability problem while still maintaining a trustless payment system. It is characterized as a peer-to-peer Payment Channel Network (PCN) consisting of nodes and channels; directed edges with both capacity and liquidity [23]. Nodes could be merchants, consumers, or routing nodes (liquidity service providers). It is not necessary that each node have a direct channel to every node with which they exchange value. Payments can be routed through other nodes as long as there exists a path with sufficient liquidity and capacity. The intermediate nodes can charge a fee for allowing others to leverage their liquidity. Payments are source-routed over cheapest paths with regard to these transfer fees. Therefore, routing nodes are incentivized to maximize the number of cheapest routes on which they lie. This problem is more formally known as the Maximum Betweenness Improvement Problem (MBI). Opening channels that improve betweenness have been shown to lead to higher expected routing opportunities and expected revenue[12][17].

Current approaches to MBI are either intractable or suboptimal. For example, implementations of the exact MBI algorithm [17] took between 30 to 40 minutes per

channel on snapshots of the Lightning Network in 2019. The network has grown considerably since then, from 2,400 nodes in February 2019 [25] to 18,000 nodes in December 2021. The network topology can update as often as every 10 minutes and so the suggestions of an exact algorithm will likely be obsolete by the time the algorithm completes. The Greedy algorithm [6] can approximate the exact solution within a factor of $1 - \frac{1}{2e}$ in directed networks, but its complexity grows polynomially with the size of the network. There is much time that can be saved by not investigating “low quality” nodes.

On the other hand, centrality based heuristics such as LightningNetworkDaemon’s *autopilot* prefer nodes with high betweenness centrality, but do not make use of the underlying structure of the network. The improvement this algorithm has on betweenness centrality has been shown to be mostly superficial as nodes usually place themselves in high competition/low revenue areas[17]. Nodes cannot increase their fees without pricing themselves out of participation. Furthermore, neither exact algorithms nor heuristics make use of previous work. Nodes can simulate opening channels and then observe the outcome. Based on that outcome, they can decide whether or not to actually open the channel. This manual process is not as intuitive when multiple channels will be opened. The individual benefit of each channel is impacted by the opening of the other channels. To try all possible channel combinations can quickly become computationally infeasible. Even after all that effort, a brute force approach would only solve the specific instance of the problem.

Rather, a reinforcement learning agent should be used. Reinforcement learning is a trial-and-error learning process by which an agent learns how to optimally interact with a complex environment. Observations from the environment turn into insight as the agent learns a relationship between its current state, its available actions, and the states into which those actions transition. By framing the MBI problem on the Lightning Network as a reinforcement learning problem, the agent can learn the impact that each channel opening has as it relates to other potential channels. The insights gained from this previous work can be exploited to produce better recommendations later on.

Arguably just as important as the solution method is the problem representation. If the problem is not represented in a way that can be generalized, then the reinforcement learning agent will only be able to solve that specific problem instance. Graph Convolutional Networks (GCNs) are a powerful method of representation that is both permutation invariant and inductive[29]. Combining this graph representation method with reinforcement learning methods allow the agent to apply previous insight to solving unseen instances of the problem through relational inductive bias[10][14][4].

This paper presents a scalable reinforcement learning approach to solving the MBI problem as it appears in the Bitcoin Lightning Network. In this approach, each node is embedded with network context using a GCN. An Advantage Actor-Critic (A2C) agent then identifies key nodes via the network embedding that a joining node should open channels with to maximize its own betweenness centrality. The performance of this method as well as its ability to generalize to unseen snapshots of the Lightning Network is evaluated.

The A2C model is trained using a custom built environment, *lightning-gym*. Un-

like other Lightning Network simulators, *lightning-gym* uses a common interface defined by OpenAI[7]. This interface is episodic in nature, which allows for the training of reinforcement learning agents. In general, an agent observes an initial state of the environment, and takes an action, which returns a new state and reward. This process is repeated until a terminal state is reached. *lightning-gym* can randomly generate small networks or import snapshots of the Lightning Network and simulate channel openings.

The *lightning-gym* simulator is able to test the A2C model against other baselines in a consistent way. After 100 training episodes on graphs with 128 nodes, the A2C agent can recommend channels in the Lightning Network that perform competitively with recommendations from centrality based heuristics and in less time. This has huge implications on the future development of the topology of the network as nodes will have access to a fast, low resource, algorithm to increase their expected routing opportunities.

The major contributions of this thesis include: a scalable reinforcement learning algorithm to the MBI problem, *lightning-gym*, an OpenAI gym environment for the Lightning Network capable of training and comparing multiple attachment strategies on random and real data, and a collection monthly snapshots of the Lightning Network from February 2021 to December 2021. This data is available for use within *lightning-gym*.

Chapter 2 Background

This chapter covers the necessary background information on the Lightning Network, betweenness centrality, and reinforcement learning. First, the Lightning Network is presented from a high level, followed by a more detailed explanation of its individual parts. The incentive model of the Lightning Network is also mentioned and how it encourages routing nodes to maximize their betweenness centrality. A summary of reinforcement learning and a high level description of the solution method to the MBI problem is included at the end.

2.1 The Lightning Network

The Lightning Network is a layer of abstraction on top the Bitcoin blockchain. At a high level, nodes open channels in the network by depositing funds into an address they share with another node. This transaction is recorded on the blockchain, indicating to other nodes in the network of the existence of the channel. The amount of funds each user owns is their liquidity. The initial liquidity balance is apparent on the blockchain. However, the current liquidity balance is maintained between the owners of the channel. This is because the balance is updated *without* broadcasting the update to the blockchain.

Payments are routed through the network in an atomic way via decrementing hash time lock contracts. Funds are only routed if there is a path with sufficient liquidity. Since users defer broadcasting their latest balance to the blockchain, the transfer of funds is not subject to a 10 minute blocktime. In addition, the cost to send funds is no longer related to mining fees. On the base blockchain, users compete to have their transactions included in a block sooner by paying higher fees. This paradigm is flipped on the Lightning Network. Routing nodes compete for routing opportunities by offering cheaper routes than other nodes.

2.1.1 Channels

Channels are the mechanism by which two parties exchange value over the Lightning Network. They are composed of two types of on-chain transactions: an initial funding transaction, and a commitment transaction. The funding transaction determines the *capacity* of the channel, while the commitment transaction determines the *liquidity* balance. Creating a channel is analogous to opening a joint bank account, wherein the total amount of funds does not change, only who owns how much. Updating this balance is free between adjacent nodes.

When creating a channel, funds of one or both parties are locked in a 2-of-2 multisignature address on-chain via a funding transaction. 2-of-2 multisignature means that updating the *liquidity* balance between parties requires a new commitment transaction signed by both parties. Dispersal of the funds occurs when the latest dually signed commitment transaction is broadcast on-chain. The funds are then moved

from the channel to on-chain addresses owned by either party according to the balance. Thus an unlimited number of feeless payments between the parties can be made by just two on-chain transactions: one to open the channel, and one to close it [23].

2.1.2 Liquidity

From a node’s point of view, capacity is made up of inbound liquidity, how much funds are available to be received via incident channels, and outbound liquidity, how much funds are available to be sent via incident channels. Sending funds across the Lightning Network reduces the outbound liquidity of the sender’s channel and increases the outbound liquidity of the recipient’s channel. If a node’s channels consists of only outbound liquidity, they are unable to receive funds and vice-versa. Channels with similar inbound and outbound liquidity are considered *balanced*. Keeping a balanced liquidity allows nodes to route the payments of others.

2.1.3 Routing Nodes

Routing nodes in the Lightning Network leverage their liquidity to route payments between peers that do not share a direct channel. In exchange for this liquidity provision, routing nodes charge a transfer fee. When making a payment, nodes will choose the cheapest path available with respect to these fees. Therefore, for a node to maximize its *expected* routing opportunities [12], it should open low-fee channels with other nodes such that it creates as many cheap paths as possible. There is a real-world cost associated with opening and closing channels. The capacity of a channel is determined by the amount of Bitcoin ‘locked’ into it by one or both parties. This work considers a scenario where a routing node can open a certain number of channels and wants to maximize their expected routing opportunity.

2.1.4 Fee Policy

The cost to send a payment between non-adjacent nodes is determined by the fee policy of each channel along the path. The fee policy contains information about the base fee and the fee rate, which are chosen by the node. The base fee is constant, but the fee rate scales with the volume of the payment. In the figure below, the fee f_b that B charges A to forward a transaction tx to C is $f_b(c, |tx|)$ where $f_b(c, |tx|) = f_b^B + f_b^P * |tx|$ (the base rate plus the fee rate multiplied by the payment size) [31]. Fees are also forward facing along the payment path. In other words, a routing node calculates their fee using the policy of its channel that is losing outbound liquidity. Note that how this fee is calculated is fundamentally different than how mining fees on the base blockchain are calculated. Since transaction fees on the Lightning Network are determined by the transaction size, rather than volume of data, smaller payments are more affordable. The disruption large payments cause to the liquidity balance of intermediary nodes is priced into the Lightning Network. As a result, larger payments may be better suited for the base Bitcoin blockchain.

2.1.5 Hash Time Lock Contracts

Atomic transfer of funds between nodes on the Lightning Network is facilitated by Hash Time Lock Contracts[1]. This contract uses a combination of two locks, a hash lock and a time lock, to ensure the atomic transfer of funds in a trustless way. The recipient will generate a time sensitive, data sensitive, invoice. Essentially, the sender commits to paying this invoice for a specific amount of time and the recipient reveals a secret value before that time to settle the invoice.

A hash lock locks the funds behind the output of a hash function, in this case RIPEMD160[1]. The funds can be unlocked by providing the input, also known as the secret preimage, to the hash function which produces the previously specified hash output. The hash lock allows the payment to be routed through the network because only those who know the secret input can redeem it. The recipient reveals this secret in order to claim the funds, but only after the sender commits their funds behind a time lock.

A time lock locks funds until a specified time. Time in this case is determined by the current blockheight of the Bitcoin blockchain. Time locks can also be used to “release” funds to a new spending condition. The time lock in a Lightning invoice is used to reverse the payment in case of uncooperative behavior. Thus, protecting the sender in case the recipient does not reveal the secret preimage. It also forces any intermediary nodes to reveal the secret in order to be reimbursed for their payment on the sender’s behalf.

2.1.6 Network Model

Altogether, the Lightning Network is a graph made up of nodes interconnected by channels containing capacity-respecting liquidity balances. $G = (V, E, F)$ where $E = \{(i, j, c, l_i, l_j) \text{ such that } i, j \in V, \text{ and } c, l_i, l_j \in \mathbb{N} \text{ and } c = l_i + l_j\}$. F is the set of fee policies associated with each channel. In simulation, the Lightning Network is typically represented as a symmetric directed weighted graph. Figure 2.1 below shows a simplification that is frequently made when evaluating the network in simulation[17][12][5][31]. A fixed size transaction is input into all of the fee policies and the returned value representing the cost to forward a payment one hop is assigned as the edge weight.

2.1.7 Attachment Strategies

An attachment strategy recommends which channels a node should open. The method for suggesting a set of channels can be anything ranging from random selection to centrality based heuristics or even the optimization of some node/network metric. In general, an attachment strategy $S(G, k, cap)$ takes as parameters:

- G - a lightning network snapshot,
- k - the number of channels to be opened, and
- cap - the capacity for each new channel

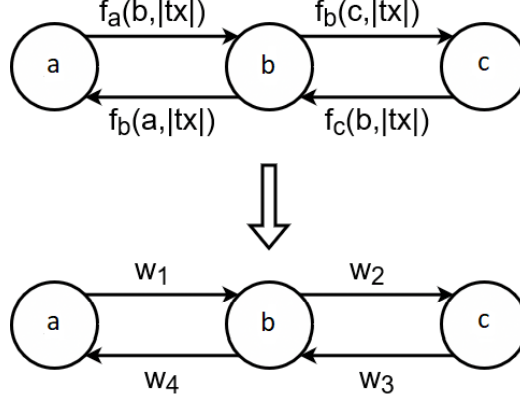


Figure 2.1: A fixed transaction size is applied to all fee policies in order to simplify the network.

and returns C a list of candidate channels [17]. Note that this definition relates to joining nodes only and does not consider nodes with preexisting channels.

2.2 Betweenness Centrality

Betweenness centrality indicates how many shortest paths make use of a given node or edge. It can be used to determine key nodes in the flow of information [3], resources[27], traffic[16], etc. In PCNs like the Bitcoin Lightning Network, flow occurs in the form of payments between individuals across paths of channels.

Definition 1 (Betweenness Centrality). The betweenness centrality of a vertex is the number of shortest paths that pass through that vertex relative to the total number of shortest paths[13], i.e.,

$$\mathbf{bc}(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

However, instead of measuring path cost by hop count, cost is calculated with respect to a channel's fee policy along the path and the payment amount being sent. The current implementation of the Lightning Network uses source routing to find the cheapest available path to route a payment. Fee-weighted betweenness centrality indicates how many cheapest paths make use of a given node or channel. Therefore, this study focuses on maximizing fee-weighted betweenness centrality.

Expected routing opportunity is closely correlated to betweenness centrality[12][25][5]. Thus increasing betweenness will also increase expected routing opportunity. This problem is formally known as the the Maximum Betweenness Improvement (MBI) problem. The MBI problem is concerned with adding edges between a joining node and other nodes in the network such that the betweenness of the joining node is maximized. As mentioned earlier, this work considers the budget constrained version of this problem.

Definition 2 (MBI Problem). Given a graph $G = (V, E)$, a vertex $v \in V$, and a budget $k \in \mathbb{N}$, add k edges incident to node v such that $\mathbf{bc}(v)$ is maximized [6].

MBI is an NP-Hard problem with no polynomial time approximation algorithm within $1 - \frac{1}{2e}$ (unless $P=NP$) [6][11]. This complexity results from the combinatorial nature of the problem and polynomial time complexity required to calculate betweenness centrality. As will be shown later on in this work, the time required to calculate the betweenness centrality of a node has increased significantly since the Lightning Network’s inception.

2.3 Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique where an agent learns how to make decisions in an environment to reach a goal. The problem the agent is attempting to solve is framed as a Markov Decision Process (MDP). By framing the problem as a MDP, an environment, or *gym*, can be created to “train” the agent. A MDP can be expressed as a tuple of $\langle S, A, T, R, \gamma \rangle$ where:

- S is the set of possible environment states.
- A is the set of actions an agent can take.
- $T : S \times A \rightarrow \Pi(S)$ is the state-transition function that describes how states transition to another state when an agent takes an action.
- $R : S \times A \rightarrow \mathcal{R}$ is the reward function that describes the reward of taking an action in a given state.
- $\gamma : 0 < \gamma < 1$ is the discount factor which determines how much an agent discounts future rewards [19].

The agent learns by interacting with the environment and observing the outcome. The agent’s behavior is defined using a policy $\pi : S \rightarrow A$ which is a function mapping states to actions. The goal of an agent in a MDP is to learn a policy that specifies the action that should be taken in each state that maximizes the agent’s expected cumulative reward.

This work explores solving the MBI problem by framing it as a MDP and applying an Advantage Actor-Critic (A2C) model. The A2C model consists of 3 parts: a policy (actor) network, a value (critic) network, and an advantage function. The policy network learns which action to take in a given state. The value network learns to approximate the expected reward, or quality, of a given state. The advantage function calculates the error between the predicted quality of state and its actual quality returned by the environment. The advantage function is used to teach the value network to predict the quality of a state which in turn teaches the policy network to suggest actions that lead to better states[21]. The A2C model was chosen for two main reasons. Firstly, the advantage function acts as a baseline, which leads to lower variance when estimating the policy gradient [30]. Secondly, actor-critic methods are well suited for large action spaces [26].

Chapter 3 Related Works

This chapter covers other’s previous work investigating attachment strategies, how graph embedding techniques and reinforcement learning can be combined to solve combinatorial problems on graphs, and state of the art Lightning Network simulators.

3.1 Attachment Strategies

This section details current approaches to the MBI problem as well as popular attachment strategies in the Lightning Network. Strategies have been divided into two categories: greedy and centrality-based. The former strategy selects channels based on immediate reward, in this case betweenness, while the latter strategies use network metrics to select nodes. There is a tradeoff between time and solution quality as well as the impact certain strategies have on the topology of the network.

3.1.1 Greedy Algorithm

The greedy MBI algorithm developed by Bergamini et al. iteratively suggests edges that lead to the greatest improvement in betweenness centrality[6]. The authors designed a dynamic algorithm to incrementally update the betweenness centrality of the joining node with each new edge. This method saves time as the algorithm only has to calculate the betweenness of the candidate edge at each iteration. The betweenness centrality of a node in a directed graph as it iteratively adds edges is a monotone non-decreasing function. This submodularity can be exploited by the greedy algorithm. After the first iteration of the greedy algorithm, node suggestions can be pruned from future iterations if the improvement caused by adding that edge is less than some observed lower bound. However, this is not the case for undirected networks.

The authors measured the runtime of the greedy MBI algorithm on several real world directed and undirected networks. The algorithms runtime on directed graphs of similar size to the Lightning Network are between ten and twenty minutes with a budget of ten edges. On undirected graphs of similar size to the network, the runtime of the greedy algorithm ranges from ten minutes to slightly less than one hour. These runtimes are more relevant as the Lightning Network is a symmetric directed graph. Most of the time is consumed during the first iteration of the algorithm, when the betweenness of the node is first being calculated. Afterwards, finding additional edges consume less time because the betweenness of the node is updated by the dynamic algorithm. The memory footprint of their dynamic update algorithm for calculating betweenness is polynomial, which makes it unsuitable for large networks. Unfortunately, the greedy algorithm’s runtime on weighted directed graphs was not evaluated.

3.1.2 Centrality-based Selection

The most popular Lightning Network protocol implementation, LightningNetworkDaemon (LND), includes an autopilot feature that will automatically open and manage channels on the user’s behalf. This feature uses the Betweenness heuristic first identified in [24] to suggest with which nodes to open channels. This attachment strategy suggests nodes with preference to high betweenness centrality. The intuition is that creating channels with well connected nodes will in turn make the joining node well connected also. Note that this implementation considers traditional betweenness centrality (shortest paths) rather than fee-weighted betweenness centrality (cheapest paths). The original implementation calculates betweenness centrality once and selects the top k vertices with respect to betweenness centrality (where k is the budget) whereas the Betweenness heuristic implemented in the LND autopilot calculates the betweenness centrality of every node in the network, selects the node with the highest betweenness centrality, and repeats this process k times.

Previous work [12] shows that the greedy algorithm outperforms centrality based attachment strategies like Betweenness in improving a joining node’s betweenness centrality. The authors show that expected *reward* improvement is also impacted by the channel attachment strategy. Reward improvement, like betweenness improvement, considers how to open channels to maximize the number of cheapest paths a node lies on but with the added complexity of deciding what fee policy to set also. The authors proved that, by assuming a constant fee policy for the joining node’s channels, that the Maximum Reward Improvement (MRI) problem reduces to MBI. Therefore the problem of MRI can be solved in stages, where the first stage is channel selection and the second stage is calculating the fee policy for each channel. This second stage is the Channel Fee Function (CFF).

In their experimental setup, the authors compare several different attachment strategies for selecting channels with and without the CFF and then compare the total revenue earned by each strategy in simulation. The attachment strategies include preferential attachment such as Betweenness, Degree, and Pagerank as well as a Random selection and the Greedy algorithm.

By including variations of the same attachment strategy but without the CFF, the authors were able to draw conclusions about how each strategy places the joining node into different levels of competition. Other factors kept equal, the authors showed that using an attachment strategy that improves betweenness centrality directly leads to higher expected reward improvement than strategies that open channels according to a heuristic. Furthermore, using the Betweenness attachment strategy lead to *worse* expected reward improvement than Random selection. Their explanation for this phenomena is that because of the nature by which the Betweenness heuristic suggests nodes, there is little opportunity to increase expected reward without decreasing expected routing opportunities [12].

This conjecture is further reinforced by the work of [17]. This work explores the impacts of different attachment strategies on the node’s ability to use the network and the development of the network’s topology. The authors evaluate a set of attachment strategies based of motivations for joining the network. They classified three types

of motivations for joining the network: users which are interested in improving local connectivity and cost to use the network, service providers which are interested in earning transaction fees and global connectivity, and an altruistic motivation which is interested in improving network metrics such as robustness and diameter.

They evaluated six different attachment strategies:

1. random selection (Random)
2. preference to highest degree (Degree)
3. preference to highest betweenness (Betweenness)
4. minimize the joining node's maximum distance from all other nodes (k-Center)
5. minimize the joining node's average distance from all other nodes (k-Median)
6. an exact algorithm for maximizing betweenness improvement (MBI)

Of the strategies evaluated in simulation, Betweenness resulted in less routed transactions than Random selection. On the other hand, they found that the MBI strategy leads to the greatest increase of expected routing opportunities and it is the second best strategy for improving one's cost to use the network. However, each additional edge in the budget added at least 30 minutes to the runtime of the MBI algorithm. In order to evaluate how the topology of the network would be affected, they simulated 5,000 nodes joining the network with a budget $k = 10$ under each strategy. However, the MBI attachment strategy was excluded from this experiment because of the intractability of finding exact solutions for thousands of nodes.

All of the attachment strategies with an objective of optimizing some node metric have a common issue: there is no specific way to select the first node. This is true for both the exact MBI algorithm and the greedy algorithm for maximizing betweenness. This is due to the fact that having a nonzero betweenness centrality requires that the node have at least two channels. When evaluating the greedy algorithm in [6], the algorithm starts from a random pivot node with at least 2 channels. In the betweenness improvement algorithms presented in [12, 17] the joining node will first open two channels with preference to nodes with the highest degree. Preferential attachment to nodes with high degree can lead to centralization of the network [17, 31].

3.2 Graph Embedding Techniques

The Lightning Network is dynamic and competitive; it updates as often as every ten minutes and users are financially motivated to continuously improve their position in the network. Therefore, using an exact algorithm for the MBI problem is not practical because by the time the solution is found it is most likely obsolete. Nor is it practical to trade time for perceived solution quality by using a centrality based heuristic. Furthermore, neither approach relies on its previous work if the network has changed by the time it completes. The problem of maximizing betweenness is a

problem being repeatedly solved on slightly different graphs. By using a reinforcement learning approach, every observation can be turned into a learning opportunity.

Previous research [10] has successfully learned greedy heuristics with low approximation ratios for solving hard problems by using graph embedding techniques and reinforcement learning. This approach is attractive as the agents can generalize their learning between graphs of different sizes and similar distributions. Their experimental setup considered three graph problems: minimum vertex cover (MVC), maximum cut (MAXCUT), and the traveling salesman problem (TSP). These problems were formulated as Markov Decision Processes where the state is the current partial solution, and each node that is not included the partial solution represents an action. The Q function learns to predict the quality of each available action in the current state. A greedy policy can be formulated from this Q function that selects the action with the greatest expected reward given the current state.

The initial challenge of this approach is constructing a representation of the nodes in such a way that it can be evaluated by the Q function. The authors used a graph embedding technique *structure2vec* to embed each node with context about its n -hop neighborhood. Each node in the graph is tagged with a feature vector including information about the node such as whether the node is included in the solution. Each node then aggregates the node feature vectors of their neighbors according to the topology and apply a nonlinear function. The output of this function is assigned as the node’s new feature vector. Successive iterations embed information from more distant nodes. These node embeddings are input into a Deep Q Network to predict expected reward. The *structure2vec* DQN model is incrementally trained using ϵ -Greedy on random Barabasi-Albert graphs generated from a similar distribution. The training graphs range from 50 to 500 nodes. In order to show scalability, the testing graphs range up to 1200 nodes. Against solvers like CPLEX, the learned greedy heuristic can achieve comparable approximation ratios ($\leq 1\%$).

Other embedding techniques such as Graph Convolutional Networks (GCN) have been explored [18]. Their experimental setup considers three graph problems: maximal independent set (MIS), minimum vertex cover (MVC), and maximal clique (MC). These are NP-Complete problems which the authors prove by reducing MVC and MC to MIS and then reducing MIS to the Boolean Satisfiability problem (SAT). The purpose of proving reducibility is so that the authors can focus on creating an algorithm for solving MIS which can then be modified for the other problems mentioned. This work explores using a GCN to label whether each node belongs to the solution set. The authors describe a spatial GCN with multiple hidden layers, the last of which is the sigmoid function. The training set for this approach is a set of graphs each paired with a binary vector that indicates which nodes belong to the solution. The GCN is then trained to predict this binary vector when given a graph as input. This is done by minimizing the binary cross-entropy loss for each training sample. The naive implementation would predict a real value within the continuous range $[0,1]$ for each of the nodes. Rounding this value produces a potential solution, although there is no guarantee that the constraints of the problem will not be violated. In situations where there exists more than one optimal solution for the given graph, the GCN may produce a labelling that does not differentiate between these solutions.

Instead, the authors use the predictions returned from the GCN as a probability map. The first algorithm presented in the work, BasicMIS, takes a graph as input and returns the best MIS solution found. The algorithm produces a probability mapping for the nodes in the graph, iterates through them descending order and labels unlabeled nodes with 1 and their neighbors with 0 until the graph is completely labelled or the algorithm encounters an already labelled node. If the graph is completely labelled, then it is compared to the current best solution, otherwise, the graph is reduced and the algorithm is recursively called with the new graph as input. Essentially, the probability mapping is guiding a depth first tree search for candidate solutions. Algorithm 2 is a modified version of BasicMIS that can generate a set of solutions by using multiple probability mappings and a queue to keep track of partial solutions. Using a queue and investigating diverse partial solutions adds breadth to the search. The combined approach results in the rapid generation of diverse solutions that can be further improved by using local search.

The model was trained using SAT problems from the SATLIB benchmark which had been converted into graphs. These graphs have about 1200 vertices each. The authors compared their approach against baselines including the S2VDQN model from [10], a state of the art SAT solver, Z3, a SOTA MIS solver, ReduMI, and a SOTA Integer Linear Program (ILP) solver, Gurobi. The algorithms were compared on 20 problems from the 2017 SAT competition with a time limit of 10 minutes. The authors approach was able to solve 100% of the problems faster than the purpose built solver, ReduMIS. On the other hand, S2VDQN was only able to solve 80% of the problems in the allotted time.

3.3 Lightning Network Simulators

There are a limited number of simulators that have been built specifically for the Lightning Network. The Lightning Network simulators listed below have either of two responsibilities. Either they are gossip simulators or they are traffic simulators.

3.3.1 CLoTH

Previous work [8], designed a network simulator, **CLoTH**, to simulate HTLC payments over snapshots of the Lightning Network and generate statistical reports. These reports are used to answer specific questions about routing, capital allocation, payment success, and the price of uncooperative behavior. The simulator processes a list of HTLC payments as discrete events and updates the state of the network. The process of simulating payments uses functions analogous to the actual functions in the Lightning Network. The authors reference the source code of the most popular Lightning Network protocol implementation, LightningNetworkDaemon, and how it guided the design of the simulator. At the end of each simulation, **CLoTH** returns a list of performance measures detailing payment success rate, payment failure due to insufficient capacity or liquidity, and payment route lengths. The authors detailed their findings in a later work [9].

3.3.2 LNTrafficSimulator

The simulator, LNTrafficSimulator, in [5] also simulates payment processing on the Lightning Network. However, the questions they sought to answer were economic in nature. The authors sought to investigate the price competition among routing nodes, optimal fee policies for central nodes, and the privacy implications of short payment paths. Furthermore, LNTrafficSimulator can simulate a bias in payment destination towards merchants. This feature stems from the assumption that most payments are made by users to online shops and liquidity service providers. The authors gathered a list of merchants from an online node directory 1ML.com and include it as input to the simulator. The authors also wanted to estimate the daily traffic and income of each node in the network. Since payments are handled peer to peer, daily traffic and income cannot be estimated without nodes volunteering information. The authors validated their parameters for the simulator by adjusting them to match the reported traffic and income of nodes owned and operated by LNBIG.com. At the time of writing, the authors estimated that 5,000 payments occurred on the Lightning Network each day, with an average size of 60,000 satoshis (0.0006 BTC), with 80% of them destined towards merchants. Running the simulation with these parameters reproduced revenue and traffic similar to that reported for LNBIG-owned nodes.

3.3.3 TimeMachine

Recent work [31] investigating the centrality of the Lightning Network used a custom-built simulator, TimeMachine. This simulator is capable of recreating snapshots of the Lightning Network by replaying gossip messages containing information about updates to the network. These gossip messages include updates about nodes, the creation and closure of channels, and updates to channel fee policies. Gossip messages were collected by several nodes deployed in the network over a duration of almost 2 years, April 2019 to January 2021. The team studied how the centralization of the network changes over time. In particular, the authors investigated the distribution of betweenness centrality across the network. An equal distribution of betweenness centrality is indicated by a 1:1 relationship between the share of nodes and the share of centrality that they possess. In other words, “X% of nodes possess X% of betweenness centrality”. The relative difference between the area under the equal distribution and the actual distribution indicates how centralized a network is. This value is known as the Gini Coefficient. During the time that these gossip messages were collected, the researchers found that centralization has increased from a Gini coefficient of 81% to a coefficient of 91%.

Chapter 4 Methods

The MBI problem can be framed as an MDP whereby an A2C agent learns to identify nodes in the network that will maximize the betweenness centrality of a joining node. In the process of training, the agent approximates two functions: the state-action function, or policy, and the state-value function. A greedy policy can be derived from the policy network by taking the action with the highest probability of being the “best” action.

4.1 Graph Representation

In order to facilitate the agent’s learning, a representation model is required. This work chose to use a Graph Convolutional Network to embed each node. Given an instance of the network, the GCN embeds each node with information about the node itself and its n -hop neighborhood where n is the number of layers in the GCN. From these embeddings, the agent learns to identify key nodes in the network.

Features such as a node’s location in the network or the policies of its channels can be exploited by a reinforcement learning agent. However, the challenge lies in designing a node representation that captures this information across different sized networks. In other words, similar nodes should have similar embeddings.

Graph Convolutional Networks are designed to address this challenge. In GCNs, each node collects the feature vectors of the nodes in its immediate neighbors. These feature vectors are aggregated by the node, thus fixing the dimensionality problem that arises between nodes with different degrees. The aggregated features are then passed to a linear layer, followed by a non-linear activation function [29]. The node then assigns the output as its new feature vector. Successive layers capture neighborhood information from nodes an increasing hop distance away. A single layer is shown below.

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} \Theta^{(l)})$$

where

- $H^{(0)} = X$ - the initial node feature tensor
- $\tilde{A} = A + I_N$ - adjacency matrix of G plus self loops
- $\tilde{D} = \sum_j \tilde{A}_{ij}$ - the out-degree of each node along the main diagonal
- Θ is a vector of learnable parameters
- σ - the nonlinear activation function

The node feature vector X contains the following information about each node:

- degree centrality - the ratio of a node’s degree to the total number of edges.

- inclusion - whether the node is included in the solution S .

The A2C agent is trained using random scale-free networks generated by the Barabasi-Albert Model. The weights were uniform in the training data. However, when testing on snapshots of the Lightning Network, performance is increased by using the reciprocal of the edge weights. The modified layer is shown below.

$$H^{(l+1)} = \sigma(\tilde{E} \times H^{(l)} \Theta^{(l)})$$

where

- $\tilde{E} = \tilde{D}^{-\frac{1}{2}} \times E \times \tilde{D}^{-\frac{1}{2}}$ - normalized edge weights
- $E = [e_{ij} = \frac{1}{f_{c_{ij}}}]$, the reciprocal of the cost of the channel, if it exists, otherwise $e_{ij} = 0$.

4.2 Markov Decision Process Formulation

The states, actions, and reward space of the Markov Decision Process are defined as:

1. *Actions*: Adding any node $v \in V$ which has not yet been added to the current state S is a legal action. Each node is represented by its embedding from the GCN.
2. *States*: A state S is a sequence of actions (nodes) on a graph G . To the value network, the state is represented by a column-wise average of all of the embeddings:

$$H_p^{(l)} = \sum_{v \in V} \mu_v$$

A terminal state is reached when the budget k has been exhausted.

3. *Transition*: Transitions are deterministic. When a node u is selected as an action, its feature indicating inclusion in the solution $x_v \in X$ is set equal to 1. In addition, S transitions to $S' = (S, u)$.
4. *Rewards*: Let the agent suggest opening a channel on behalf of node v to node u . The reward, $r(S, u)$, of taking action u in state S is the change in the betweenness centrality of v after connecting u and transitioning to the new state S' .

$$r(S, u) = bc_{S'}(v) - bc_S(v)$$

4.3 Actor-Critic Model

Actor-critic models separate the roles of evaluating the quality of a state and determining the action to take in given a state. These roles are assigned to the value network and policy network, respectively. This architecture was chosen because it is well suited for problems with large state and action spaces. The state space scales with both the budget and the number of nodes in the network. A node is either included in the solution (state) or not. For a budget, n , equal to the number of nodes, the number of possible states is 2^n .

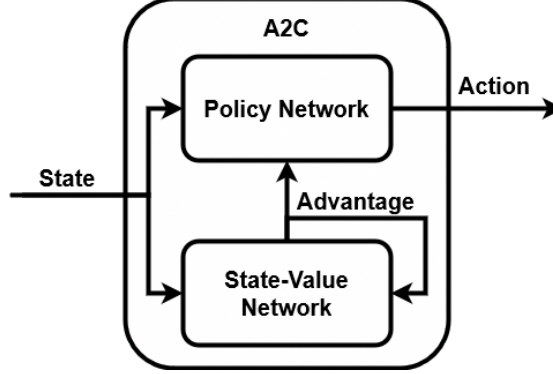


Figure 4.1: Advantage Actor-Critic Architecture

4.3.1 Policy Network

The policy network, P^π , takes as input the embedded network and produces a probability distribution. This is a vector of values, each representing an action's likeliness to be the best action. The policy network learns to increase the probability of selecting the node u in state S that improves the fee weighted betweenness centrality of the joining node v the most.

During training, the agent's policy samples from the probability distribution returned by the policy network, taking $H^{(l)}$ as input.

$$\pi(S) := \text{sample}_{u \in \bar{S}} P^\pi(H^{(l)})$$

During evaluation, the A2C agent's policy takes the mostly likely action according to the policy distribution.

$$\pi(S) := \text{argmax}_{u \in \bar{S}} P^\pi(H^{(l)})$$

4.3.2 Value Network

The value network is responsible for predicting the quality of the partial solution. The quality of the partial solution is equal to the expected reward of the episode. In other words, the value network predicts the total betweenness improvement of the isolated node assuming it has already created channels with nodes from the partial

solution. The value network takes as input a mean pooling of the output layer, $H_p^{(l)}$, multiplies it by a set of learnable parameters, applies a nonlinear activation function, and returns a scalar value. $H_p^{(l)}$, $H^{(l)}$, and X should be considered synonymous with S .

$$V^\pi(H_p^{(l)}) = \sigma(H_p^{(l)}\Theta) \approx \mathbb{E} \left[\sum_{i=1}^T \gamma^{i-1} r_i \right]$$

4.4 Training Architecture

A high-level description of how the A2C agent learns the greedy heuristic:

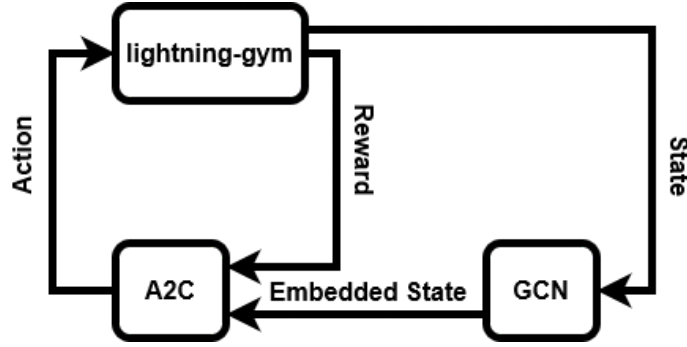


Figure 4.2: Training architecture. A2C accepts as input a graph embedding from the GCN and then takes an action in the environment, producing a new state-reward pair.

1. A random scale free network , $G = (V, E)$, is generated.
2. Partial solutions are represented as a set $S = \{v_1, v_2, \dots, v_{|S|}\}$ where $v_i \in V$ and $\bar{S} = V - S$. In practice, however, the structure of G is manipulated directly and the partial solution is represented using a 0-1 vector. For each element x_v in this vector, $x_v = 1$ if the corresponding node $v \in S$ and 0 otherwise. This vector is used in the feature tensor of the nodes.
3. Each node in S represents a new neighbor with the A2C agent's node. The quality of S is the resulting betweenness of the agent's node after having opened channels with all the nodes in S .
4. The agent selects action that iteratively maximizes the quality of S . It selects these actions based off its previous observation and the current embedded network.

4.4.1 Network Training

A custom environment, *lightning-gym*, is used to train the agent. This environment simulates channel openings on small, randomly generated, scale-free networks and returns the reward. This observation is used to train the A2C agent. During each iteration of an episode, the A2C agent collects the environment observations and output of the Value and Policy networks. The following policy gradient function is used in training the network:

$$\nabla_{\theta} J(\theta) \sim \left(\sum_{t=0}^{T-1} \log \pi(a|S_t) \right) A(S_t, a)$$

where $A(S_t, a)$ is the expected benefit of taking an action a in state S at time t :

$$A(S_t, a) = Q(S_t, a) - V^{\pi}(S_t)$$

and $Q(S_t, a)$ is the actual value of taking action a at time t .

The observations and outputs are collected by the A2C agent to be used as input to the policy gradient function which back-propagates corrections through the value and policy networks.

Chapter 5 Experiments

5.1 Baselines

The performance and scalability of the A2C agent is compared to six other attachment algorithms:

- **Random** - suggests k nodes sampled uniformly at random.
- **Degree** - suggests k nodes w.r.t. highest degree.
- **Betweenness** - suggests k nodes w.r.t. greatest betweenness centrality.
- **k -Center** - suggests k nodes that minimize the joining node's *maximum* distance to all other nodes.
- **Greedy** - suggests nodes that result in the greatest betweenness improvement after trying all available actions.
- **Trained Greedy** - like Greedy, except this algorithm suggests the best action out of five sampled from the policy network of the trained A2C agent.

The Trained Greedy algorithm is inspired by the approach taken [18]. However, Trained Greedy samples multiple actions from the trained policy network of the A2C agent, tries all of them, and then selects the one with the best immediate improvement. The approximation algorithm from [15] is used for the k -Center algorithm. Results from the Greedy Algorithm are excluded for graphs with $\geq 1,000$ nodes. The performance of the Random algorithm is the average result of 30 trials.

5.2 lightning-gym

A node was deployed to collect monthly snapshots of the Lightning Network from February 2021 to December 2021. These snapshots contain information about the network's nodes and channels, including fee policy and capacity. These snapshots are used in the custom OpenAI Gym environment, *lightning-gym*, to train the A2C agent and compare it against baselines.

Instances of the Lightning Network are pruned before being used in training and comparison. The network itself is a multidirected graph represented as an array of nodes and an array of channels. To be included in the network, a channel must be active, have a defined policy, and a minimum capacity. If a channel is not active or its policy is undefined, it cannot be used by the network, and so it is removed. Channels with low capacity are removed because they are easily made unbalanced with relatively low payments. Therefore, low capacity channels add little to the connectivity of a node and should be ignored. The minimum channel capacity is set to 0.01 Bitcoin. If a pair of nodes has more than one channel between them, the higher fee is retained and the capacities are combined.

Afterwards, all bridges are removed from the network, along with all of the resulting smaller connected components. This guarantees that every remaining node has a degree of at least 2 and every remaining channel is capable of undergoing a circular rebalance. The remaining network consists of well connected nodes and public channels with good probability of liquidity.

The environment generates random scale-free directed graphs and adds an isolated node. These graphs are generated using the Barabasi-Albert method. The A2C agent trains on this environment selecting nodes for the joining node to open channels with. The environment simulates the channel opening and returns the betweenness improvement of the joining node as the reward and the new embedded graph as the next state. This process is repeated until the A2C agent’s budget is exhausted. Afterwards, the agent updates its policy and value networks with regard to the new state-action-reward observations. The cumulative reward of an episode represents the total betweenness improvement the agent was able to gain.

5.3 Experiment 1 - Setup

The objective of **Experiment 1** is to compare the performance of each algorithm in real and synthetic environments. In the real environment scenario, each algorithm is compared on monthly snapshots of the Lightning Network. Each algorithm is given a budget of 10 channels, and identical fee policies. The cumulative reward of each algorithm is collected at the end of each episode. In the synthetic environment, performance is compared on two random BA graphs: one with 128 nodes and one with 1024 nodes. For these two graphs, performance is compared as the budget increases from 1 to 15. The algorithms are evaluated on graphs of two different sizes to determine whether there is a drop off in performance as the size of the graph increases.

5.4 Experiment 2 - Setup

In order to determine the scalability of this approach, **Experiment 2** includes an analysis of the size of the Lightning Network over time before and after pruning. This is followed by a comparison of the runtimes for each algorithm to return channel recommendations for each monthly snapshot of the Lightning Network. In this experiment, results are evaluated with consideration to the time required to discover them. The algorithms and environment were implemented in Python. The runtime of the algorithms are measured using Python’s built-in library *timeit*. The machine used was an MSI GE76 Raider with 16GB DDR4 RAM, 11th Gen Intel i7 processor, and GeForce RTX 3060 with 6GB VRAM.

5.5 Experiment 3 - Setup

In **Experiment 3** the training performance of the A2C agent is evaluated under two scenarios. The environment has been designed so that the agent can be trained on

a new random instance every episode or repeatedly on a single instance. The former scenario should lead to better performance on the general MBI problem, but the latter scenario is more analogous to repeatedly solving the MBI problem in a specific context, such as the Lightning Network. Repeated training demonstrates whether the A2C agent is able to exploit previous work. The cumulative reward after each episode is plotted as a percentage of the performance of the greedy algorithm. i.e. A score greater than or equal to 1 means that the agent found a solution as good as or better than the solution found by greedy search. The agent is trained for 100 episodes on graphs with 128 nodes. It is given a budget of 10 channel openings. The table below includes the parameters used when training the value and policy networks.

Training Parameter	Value
GCN Dimensions	2x128x128
Actor Network Dimensions	128x128
Policy Network Dimensions	128x1
Learning Rate	1e-2
Decay Rate	0.999
Activation	ReLU
Optimizer	ADAM

Chapter 6 Results

6.1 Experiment 1 - Results

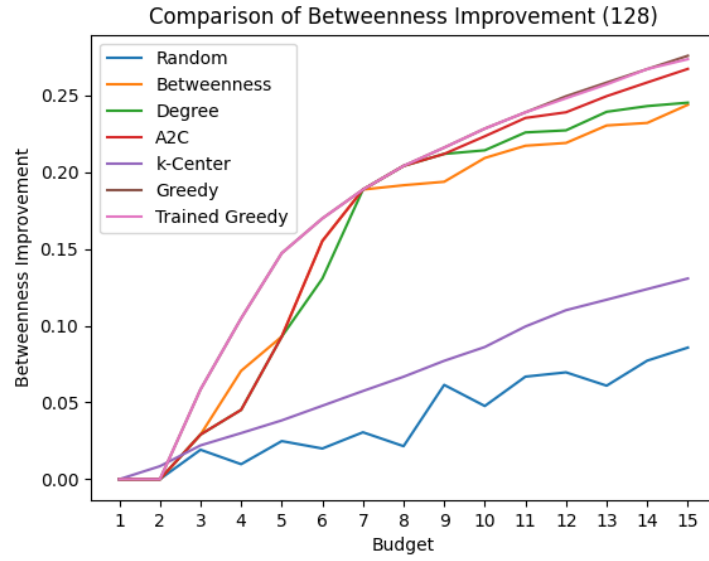


Figure 6.1: Performance comparison on a synthetic network with 128 nodes.

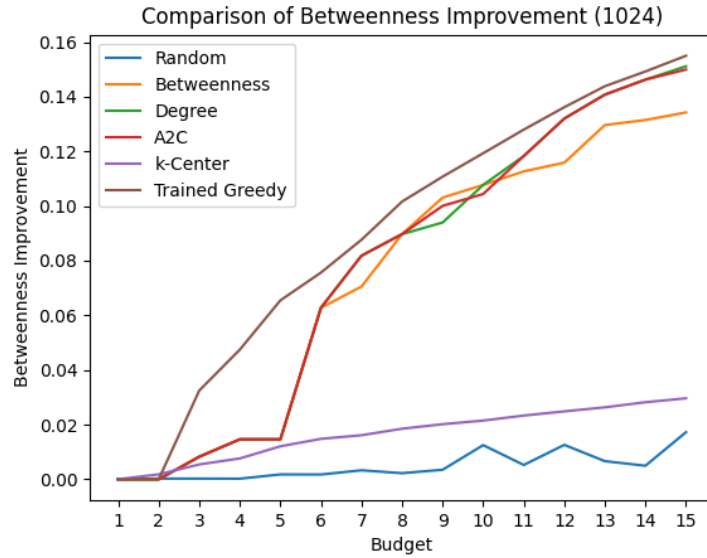


Figure 6.2: Performance comparison on a synthetic network with 1024 nodes.

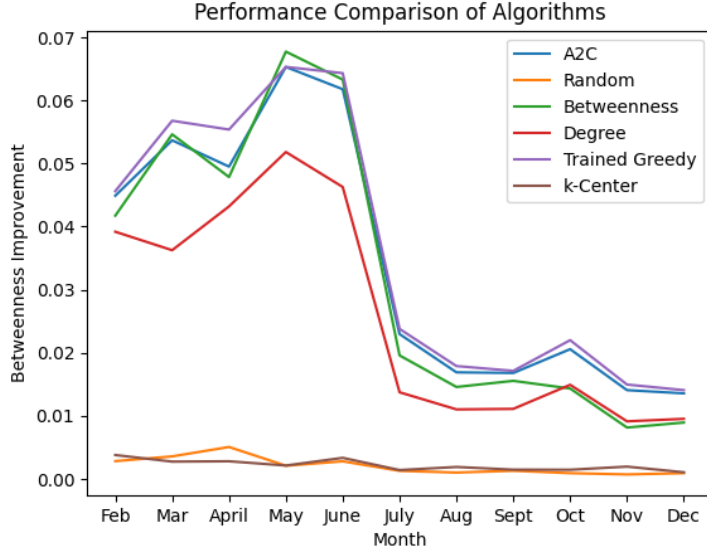


Figure 6.3: Comparison of the performances of A2C agent, Betweenness, Degree, and Random on instances of the Lightning Network. Each algorithm starts with an isolated node and a budget of 10 channels.

Table 6.1: Comparison of performances on the Lightning Network.

Months	A2C	Random	Betw.	Degree	Trained	kCenter
Feb.	0.0449	0.0028	0.0417	0.0392	0.0456	0.0038
March	0.0537	0.0036	0.0546	0.0363	0.0568	0.0027
April	0.0495	0.0051	0.0479	0.0432	0.0554	0.0028
May	0.0653	0.0021	0.0677	0.0518	0.0653	0.0021
June	0.0618	0.0028	0.0633	0.0463	0.0643	0.0033
July	0.0230	0.0013	0.0196	0.0137	0.0238	0.0014
Aug.	0.0169	0.0010	0.0146	0.0110	0.0179	0.0019
Sept.	0.0168	0.0013	0.0156	0.0111	0.0171	0.0015
Oct.	0.0206	0.0009	0.0143	0.0149	0.0220	0.0015
Nov.	0.0141	0.0007	0.0082	0.0091	0.0150	0.0020
Dec.	0.0136	0.0009	0.0089	0.0095	0.0141	0.0011

Figures 6.1 and 6.2 show a comparison of the agent’s performance as budget increases on two differently sized synthetic networks. Overall, behavior is similar between the two figures. Both show an increase in betweenness as budget increases, although there is less improvement gained in Figure 6.2. Centrality based heuristics show a clear advantage over Random and k -Center. There is more variation in performance on the network in Figure 6.1. On the larger network in Figure 6.2, the performance is less varied.

6.2 Experiment 2 - Results

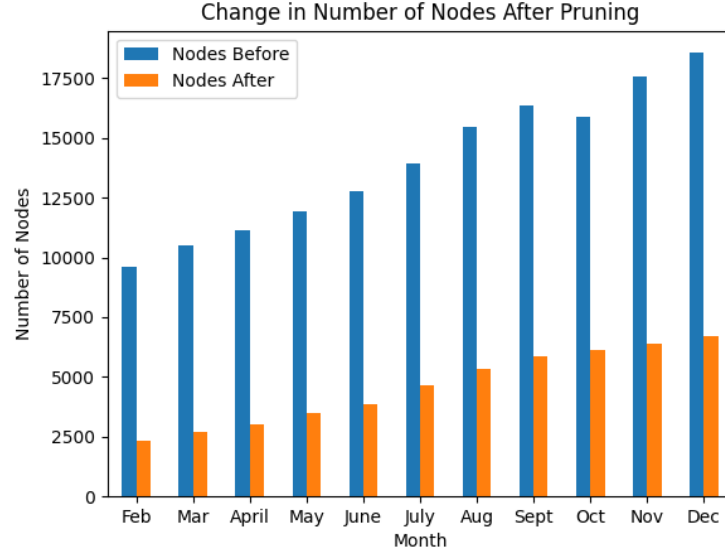


Figure 6.4: Change in number of nodes after removing bridges, smaller connected components, and low degree nodes.

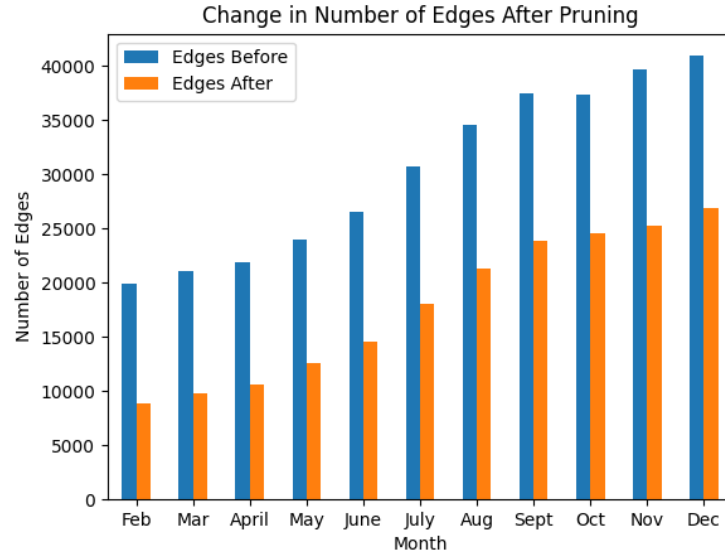


Figure 6.5: Change in number of channels after removing private, inactive, and low capacity edges.

From February 2021 to December 2021 the number of nodes in the network increased 75% from 10k nodes to 17.5k nodes. In the same time, the number of channels

Table 6.2: Change in Number of Nodes and Edges after pruning

Months	Nodes Before	Nodes After	Edges Before	Edges After
Feb	9602	2316	19949	8814
Mar	10516	2676	21063	9818
April	11161	2996	21854	10607
May	11907	3491	24039	12579
June	12770	3876	26550	14528
July	13944	4643	30772	18081
Aug	15469	5316	34600	21277
Sept	16374	5872	37427	23816
Oct	15891	6144	37345	24594
Nov	17572	6409	39672	25298
Dec	18558	6686	40938	26878

doubled from 20k channels to 40k channels. The size of the network is reduced significantly by the pruning operation. However, this amount is decreasing over time. In February, the pruning operation reduced the number of nodes by 75%. In December, the pruning operation reduced the number of nodes by 63%.

The ratio of remaining “well connected” nodes grew steadily over time relative to the entire network. In the February, 75% of the nodes were removed, yet 45% of the edges remained. In the December snapshot, 66% of the nodes were removed and 66% of the edges remained.

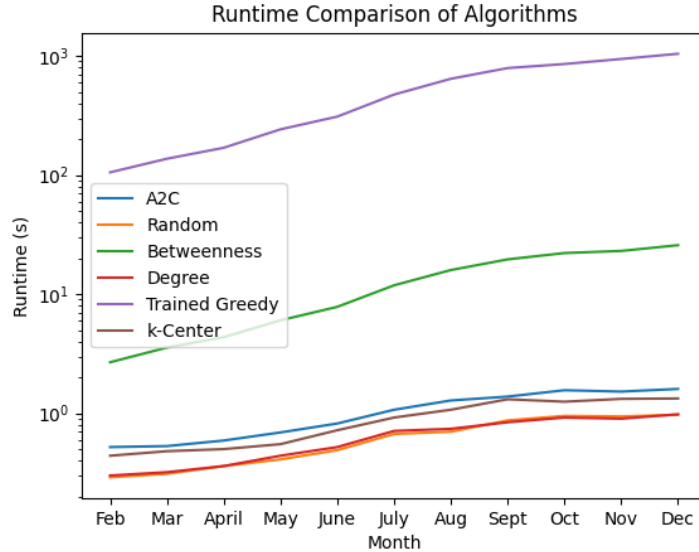


Figure 6.6: Comparison of the runtimes of the five algorithms for each month.

Table 6.3 lists the cumulative runtime of each algorithm over each month. The cumulative runtime is the total time in seconds each algorithm requires to suggest

Table 6.3: Comparison of runtimes for each algorithm.

Months	A2C	Random	Betw.	Degree	Trained	kCenter
Feb.	0.52	0.29	2.68	0.30	105.34	0.44
March	0.53	0.31	3.56	0.32	137.05	0.48
April	0.59	0.36	4.35	0.36	169.39	0.50
May	0.69	0.41	6.03	0.44	241.90	0.55
June	0.82	0.49	7.82	0.52	309.40	0.72
July	1.07	0.67	11.86	0.71	473.69	0.92
Aug.	1.28	0.70	15.93	0.74	642.95	1.07
Sept.	1.38	0.87	19.58	0.84	790.56	1.31
Oct.	1.56	0.95	22.13	0.92	855.17	1.25
Nov.	1.52	0.94	23.05	0.90	942.50	1.32
Dec.	1.60	0.97	25.76	0.98	1042.43	1.33

ten channels on the given graph. Betweenness and Trained Greedy stand out as the two longest running algorithms. The other four algorithms execute faster than the shortest instance of Betweenness. Figure 6.6 shows the runtimes of each algorithm plotted on a log scale.

6.3 Experiment 3 - Results

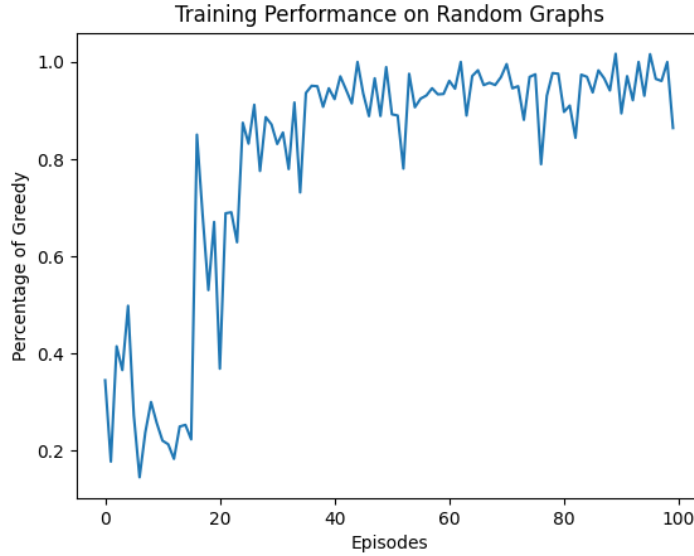


Figure 6.7: Performance of agent trained on different random scale free networks with 128 nodes and a budget $k = 10$.

Figure 6.7 shows the A2C agent’s performance quickly increasing after 40 episodes. The agent scores at least 80% of Greedy from there on. The agent’s performance

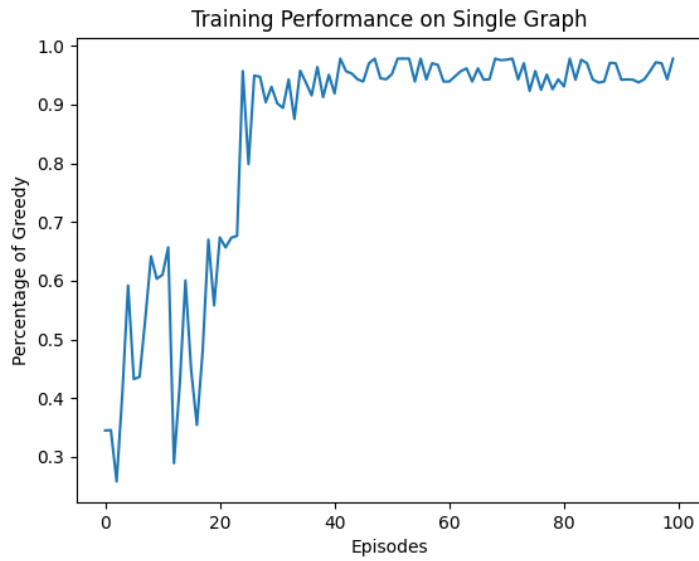


Figure 6.8: Performance of agent repeatedly trained on single instance of a random scale free network with 128 nodes and a budget $k = 10$.

converges even sooner when repeatedly trained on a graph. In Figure 6.8, after 25 episodes, the agent is scoring at or above 90% of Greedy.

Chapter 7 Discussion

7.1 Performance

From the figures 6.1, 6.2 and 6.1, it is clear that the A2C agent is capable of generalizing its learning to different budgets and different sized networks. The A2C agent shows comparable performance in Figure 6.1, breaking out from the centrality based heuristics after a budget of 8. The next best performing algorithm is Degree followed by Betweenness. Surprisingly, k -Center is the *only* strategy whose node has a positive betweenness centrality when given a budget of 2. In Figure 6.2, Betweenness begins to diverge after budget 6, but the A2C and Degree algorithm stay relatively equal. The A2C agent performs competitively, even on unseen networks eight times larger than those it was trained on.

The Trained Greedy algorithm shows that improvement can be gained by trading time to explore several recommended actions. In Figure 6.1, the Trained Greedy algorithm performs identically to the Greedy algorithm. The Greedy algorithm was not evaluated in the larger synthetic network so their performance cannot be compared. However, the Trained Greedy algorithm outperforms every other baseline in the larger network as well.

Through experimentation, it was observed that after training for 100 episodes on random networks the A2C agent performs competitively on unseen instances of the Lightning Network. Figure 6.1 shows the performances of each algorithm on monthly snapshots of the network. Performance is determined by the fee-weighted betweenness improvement of an isolated node in the Lightning Network after adding 10 channels suggested by each of the respective algorithms. The Greedy algorithm was not included in this comparison because of time constraints. However, the Trained Greedy algorithm performs best in ten of the eleven months. May is the only month where the Betweenness algorithm outperforms both A2C and the Trained Greedy algorithm.

From February to June, the Betweenness and A2C algorithms perform similarly. Of those five months, Betweenness outperformed A2C in March, May, and June. From June until December, the A2C agent outperforms Betweenness. Additional improvement is gained through the Trained Greedy algorithm at the expense of time. Degree is the next best algorithm after Betweenness, but it is clearly underperforming. Meanwhile, k -Center has a similar betweenness improvement as picking channels at random. Considering the nature of k -Center and the size of the Lightning Network, it may need a higher budget to gain a significant betweenness centrality in the network.

A possible limitation of this work is indicated by the disparity in performance between the Degree algorithm on the random networks in Figures 6.1 and 6.2 versus the real-world network in Figure 6.1. The future works section discusses the limitations of using the BA model as a substrate network in more detail.

7.2 Scalability

In Figures 6.4 and 6.5, it can be seen that the size of the network has been increasing consistently. Even though the pruning operation removes over half of the nodes, the number of channels is not as strongly affected. This indicates that removal of these nodes had low impact on the connectedness of the remaining network. However, even with the pruning operation, the “well-connected” portion of the network on which these algorithms were tested is clearly increasing.

The Betweenness baseline requires that the betweenness centrality of the network be calculated at least once. As a result, the minimum runtime of the Betweenness algorithm increased 10 fold this year from 2.6 seconds in February to 25.1 seconds in December. The Trained Greedy algorithm also calculates betweenness several times before making a recommendation. Unsurprisingly, its runtime was the greatest among the seven algorithms. However, the Trained Greedy algorithm’s runtime increased from 105 seconds to 1,042 seconds, a similar factor as Betweenness. Unlike the Greedy algorithm, which must try all actions before choosing the best one, Trained Greedy tries a constant number of actions each iteration and therefore scales at the same rate as the Betweenness algorithm.

In contrast, the total time required for the A2C agent to suggest 10 channels only increased by a factor of 3 from 0.5 seconds in January to 1.6 seconds in December. Indeed, all algorithms which did not need to calculate betweenness only increased by a factor of 3. This factor is the same rate of growth as the number of nodes in the Lightning Network snapshots. This relation suggests that the reinforcement learning algorithm, like Degree and Random, scales linearly with the size of the network.

7.3 Training

The GCN/A2C Agent can be trained in two different modes: repeatedly on a single instance, or on a new random instance each time. For better comparison between these modes, the performance during training is plotted relative to the performance of the Greedy algorithm. At the end of each episode, the betweenness centrality of the joining node controlled by the Agent is compared to the betweenness of a joining node controlled by the Greedy Algorithm. This metric indicates how well the Agent performed relative to the Greedy Algorithm by dividing the former’s result by the latter.

By measuring performance as a percentage of greedy, it is clear that the A2C agent learned to generalize between different graphs. Figure 6.8 shows the performance of the A2C agent when trained repeatedly on a single instance. This use case is more analogous to repeatedly solving for MBI on a slowly growing Lightning Network, or any other large real-world network. The A2C agent quickly approaches greedy performance when trained on a single graph. These results show that a reinforcement learning approach would be able to exploit previous work done for solving the MBI problem.

7.4 Future Work

The work of [12] addresses both the MBI problem, and the MRI problem. The former entails deciding which nodes to open channels with to maximize expected routing opportunities, and the latter entails, after having opened these channels, deciding what fees to charge to maximize expected fee revenue. A future work might explore a combined approach using the method described by [12] to decide fees for channels suggested by the A2C agent.

Pruning instances of the Lightning Network is a limitation of this work. The intention is to reduce the memory requirements of the network, decrease noise among candidates for the GCN/A2C agent, and counteract betweenness inflation caused by ill-connected pendant nodes. However, in doing so, important information may be lost from the network. In a future iteration, importance sampling methods [20] may be able to avoid this bias while still saving on memory resources.

Figures 6.1 and 6.1 show a disparity of performance in the Degree algorithm as an attachment strategy. In the random networks used to train the A2C agent, Degree performs similarly to Betweenness and the agent. However, the Degree algorithm under performs on the real-world networks used to test the agent. Recent work [28] has called into question the efficacy of using the BA model to evaluate channel management programs for the Lightning Network. For example, in their analysis, the authors found that the Lightning Network is disassortative while BA graphs are practically neutral. Additionally, the Lightning Network has a larger network diameter, nearly double that of a similarly sized BA graph, and exhibits a closeness-preferential attachment while BA graphs are generated with degree-preferential attachment. Since BA graphs are used to generate training networks in *lightning-gym*, there may be a performance gain in using a different model. A future work might explore training the A2C agent on samples obtained from the Lightning Network, or networks generated with a preference to closeness.

The bottleneck of the current architecture is the reward function that calculates the joining node’s betweenness centrality. As mentioned in the Related Works, dynamic algorithms for updating betweenness centrality are memory intensive and/or have not been designed for directed weighted graphs. An architecture with a guaranteed approximation lower bound for predicting betweenness could significantly speed up training of the agent. The reward function would become probabilistic, but it would still measure the approximate reward of each action. With a guaranteed lower bound, the agent would still be able to learn which actions lead to better states. This architecture change could also save time selecting the “best” action in the Trained Greedy algorithm.

The Trained Greedy algorithm samples multiple actions from the same probability mapping. The inspiration from this design comes from the sampling methods described in [18]. A future iteration of this work might explore training multiple A2C agents to produce multiple probability mappings. Sampling actions from each of the probability mappings rather than just one could produce a diverse set of actions and lead to better performance.

The primary objective of the A2C agent is to maximize betweenness improvement.

However, one may be able to use different, possibly multiple objectives. Minimizing objectives such as Gini coefficient, average fees for the network, or diameter, improve the network. Maximizing metrics such as closeness centrality and number of triangles make the network personally cheaper to use and rebalance. Betweenness improvement becomes sparse as a node adds more channels. It may be useful to consider optimizing other metrics as budget increases. Market context such as a nodes reputation or its function in the network as consumer, merchant, or router would also be of interest. One could even imagine automatically reallocating capital by opening *and* closing channels. Other works[5] have shown that proper allocation of channels leads to a faster return on investment.

Furthermore, if this approach were deployed to nodes running LND, they could combine their models in a federated learning style. One could imagine users identifying what objectives they want optimize for and then entering pools of processing power to collectively train a model towards these ends.

Chapter 8 Conclusion

This work provides a background on the Bitcoin Lightning Network including the scalability problem it solves, and the new challenge that arises, the MBI problem. Others previous work shows that solving the MBI problem directly leads to greater expected routing opportunities in the Lightning Network[17]. Increasing one’s routing opportunities leads to greater expected revenue. Thus, the motivation for solving the MBI problem is apparent.

Current attachment strategies have been shown to have drawbacks which essentially amount to trading time for performance. Just the time required to calculate the betweenness centrality of a node in the Lightning Network has increased significantly. There exists a need for an algorithm for improving betweenness that can scale with the Lightning Network.

Reinforcement learning and graph embedding techniques are a powerful method for solving graph optimization problems. These approaches have been shown to have low approximation ratios when solving classic problems such as the Maximal Independent Set and Minimum Vertex Cover problems. Agents can generalize their learning, allowing the agents to solve problems on graphs magnitudes of size larger than those on which they were trained.

It has been shown that the MBI problem can be formulated as a MDP, thus allowing reinforcement learning techniques to be applied to solving it. An Advantage Actor-Critic model is chosen as the agent, as they are compatible with problems that have a large action space. However, in order to generalize learning, a graph embedding technique is also used. This work chose to use GCN’s as they are able to capture structural information about a node and it n -hop neighborhood.

A custom developed Lightning Network environment, *lightning-gym*, is used to train the A2C agent on random graphs and evaluate it against six other algorithm for suggesting channels in the Lightning Network. Through experimentation, it was shown that an Advantage Actor Critic model can learn a greedy heuristic to suggest channels that maximize a joining node’s betweenness centrality. After training on small randomly generated graphs for 100 episodes, the agent performs competitively with the current selection algorithm implemented in the LND *autopilot* as well as four other baselines. Furthermore, this work showed that the runtime of this reinforcement learning algorithm scales linearly with the size of the Lightning Network. Using a trained A2C agent in combination with a GCN for node embedding to select nodes is a fast, low-resource algorithm for improving one’s betweenness centrality in the Lightning Network.

Bibliography

- [1] A. Antonopoulos, O. Osuntokun, and R. Pickhardt. *Mastering the Lightning Network: A Second Layer Blockchain Protocol for Instant Bitcoin Payments*. O'Reilly Media, Incorporated, 2022.
- [2] A. M. Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. "O'Reilly Media, Inc.", 2014.
- [3] M. Baglioni, F. Geraci, M. Pellegrini, and E. Lastres. Fast exact computation of betweenness centrality in social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 450–456. IEEE, 2012.
- [4] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- [5] F. Béres, I. A. Seres, and A. A. Benczúr. A cryptoeconomic traffic analysis of bitcoin's lightning network. *arXiv preprint arXiv:1911.09432*, 2019.
- [6] E. Bergamini, P. Crescenzi, G. D'angelo, H. Meyerhenke, L. Severini, and Y. Velaj. Improving the betweenness centrality of a node by adding links. *Journal of Experimental Algorithmics (JEA)*, 23:1–32, 2018.
- [7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.
- [8] M. Conoscenti, A. Vetrò, and J. C. De Martin. Cloth: A lightning network simulator. *SoftwareX*, 15:100717, 2021.
- [9] M. Conoscenti, A. Vetrò, and J. C. De Martin. Hubs, rebalancing and service providers in the lightning network. *IEEE Access*, 7:132828–132840, 2019.
- [10] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 2017-Decem:6349–6359, 2017.
- [11] G. D'Angelo, L. Severini, and Y. Velaj. On the maximum betweenness improvement problem. *Electronic Notes in Theoretical Computer Science*, 322:153–168, 2016.
- [12] O. Ersoy, S. Roos, and Z. Erkin. How to profit from payments channels. In *International Conference on Financial Cryptography and Data Security*, pages 284–303. Springer, 2020.

- [13] L. C. Freeman. A set of measures of centrality based on betweenness, 1977.
- [14] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.
- [15] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.
- [16] A. Kazerani and S. Winter. Can betweenness centrality explain traffic flow. In *12th AGILE international conference on geographic information science*, pages 1–9, 2009.
- [17] K. Lange, E. Rohrer, and F. Tschorsch. On the impact of attachment strategies for payment channel networks. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2021.
- [18] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.
- [19] M. Littman. Markov decision processes. In N. J. Smelser and P. B. Baltes, editors, *International Encyclopedia of the Social Behavioral Sciences*, pages 9240–9242. Pergamon, Oxford, 2001.
- [20] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh. Learning heuristics over large graphs via deep reinforcement learning. 3 2019.
- [21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.
- [22] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
- [23] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.
- [24] R. Puzis, Y. Elovici, and S. Dolev. Finding the most prominent group in complex networks, 2007.
- [25] E. Rohrer, J. Malliaris, and F. Tschorsch. Discharged payment channels: Quantifying the lightning network’s resilience to topology-based attacks. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 347–356. IEEE, 2019.
- [26] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.
- [27] W. To. Centrality of an urban rail system. *Urban Rail Transit*, 1(4):249–256, 2015.

- [28] Z. Wang, R. Zhang, Y. Sun, H. Ding, and Q. Lv. Can lightning network's autopilot function use ba model as the underlying network? *Frontiers in Physics*, 9, 2022.
- [29] M. Welling and T. N. Kipf. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016.
- [30] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.
- [31] P. Zabka, K.-T. Foerster, S. Schmid, and C. Decker. A centrality analysis of the lightning network. *arXiv preprint arXiv:2201.07746*, 2022.

Vita

Vincent Michael Davis

Place of Birth:

- Columbus, NC

Education:

- Berea College, Berea, KY
B.A in Computer Science, Aug. 2019
B.A in Mathematics, Aug. 2019
cum laude

Professional Positions:

- Graduate Teaching Assistant, University of Kentucky Fall 2019–Spring 2022
- Google AMLI Bootcamp Teaching Assistant, University of Kentucky Summer 2021