Adaptable Resource Generation Protocols For Quantum Networks

Reinforcement Learning For Fast Quantum Resource Generation Policies

Efe Aksel Tacettin[1]

Supervisors: Gayane Vardoyan[1]**, Bethany Davies**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Efe Aksel Tacettin
Final project course: CSE3000 Research Project
Thesis committee: Gayane Vardoyan, Bethany Davies, Rihan Hai

Abstract

Quantum networks allow quantum processors to communicate over large distances. These networks often require simultaneously existing multiple entangled pairs of quantum bits (entangled links) as a fundamental resource for communication. Link generation is a sequential and probabilistic process, and successfully generated links are stored in a quantum memory. Links in memory are subject to noise that causes their quality to decay and become unusable. This paper uses reinforcement learning (RL) to investigate dynamic tuning of the entanglement generation protocol to minimise the time to generate multiple links. By comparing a fixed number of actions to a continuous action space, we analyse the importance of finer-grained tunings of the protocol. This is tested in simulated near-term and medium-term network abstractions. The results show that protocol tuning significantly reduces the mean time to generate entangled links, with finer tuning providing greater benefits up to a point. Furthermore, a heuristic is derived from the RL policies which matches and exceeds their performance. Future work can explore more advanced reinforcement learning algorithms to find better policies, as well as using different noise models to make more generally applicable policies.

1 Introduction

Quantum networks, analogous to classical networks, allow for the communication of quantum processors between physically large distances. They promise a wide variety of applications, many of which require entangled pairs of quantum bits between the parties in the network. These entangled pairs of quantum bits, or entangled links, can be seen as a type of resource necessary for communication [1]. Entangled link generation can be carried out by protocols which have a success probability p, and generate links with some fidelity F [2]. The fidelity is a measure of quality which indicates how similar two quantum states are [3]. In this case, the fidelity represents how similar the generated entangled pair of qubits is to a perfect quality entangled qubit pair.

Entangled link generation attempts occur sequentially. When multiple links are required, successfully generated links are stored in a quantum memory so that further link generation attempts can be made. Links that are stored in memory are subject to noise, causing their fidelity to decay over time. Links which have decayed below a certain fidelity are no longer usable, and are therefore discarded. The fidelity below which links are no longer useful is called the threshold fidelity F_{thresh}, and it depends on the application which requires the links. If links expire rapidly, simultaneously generating a desired number of links can take a very long time, limiting the network throughput. This motivates the investigation of fast generation procedures for multiple links. One possible approach is to tune the generation protocol throughout the generation process. By tuning a protocol, the success probability of the protocol can be increased or decreased, at the cost of decreasing or increasing the fidelity, respectively.

Previous work has been done by the authors of [2] to develop the tools to analyse entanglement generation protocols in a similar multiple link setup. The work focuses on static protocols with a fixed link generation probability, that is, without the tuning of the protocol parameters during generation. They derive analytical methods to understand the distribution of waiting times to achieve a desired amount of links for a protocol.

Due to the dynamic nature of our problem, particularly when scaling to many entangled links, reinforcement learning (RL) is well suited to find good policies. The authors of [4] have previously demonstrated the potential of model-based RL techniques in optimizing end-to-end entanglement generation between multiple nodes in a quantum network In this paper, we apply RL to investigate the use of dynamic protocols in minimising the time taken to generate multiple entangled links. To this aim, we analyse how beneficial fine-grained tunings of the protocol throughout generation are. We compare fixed numbers of generation probabilities versus a continuous tunable parameter. Additionally, we derive a heuristic from the RL policies.

In Section 2, a mathematical modelling of the problem is discussed. Section 3 introduces the reinforcement learning algorithm and heuristic used. Section 4 describes the experimental setup. Section 5 displays and interprets the results. Section 6 briefly concludes the paper and discusses possible future works. Section 7 describes ethical considerations in this paper.

2 Problem Formulation

The following section provides a formal model of the system, which is modelled as a Markov Decision Process (MDP). An MDP consists of a state space, an action space, a transition probability function, and, for RL purposes, a reward function [5]. There are two main reasons for choosing this model. Firstly, we assume entanglement generation attempts are done in constant time steps. This is important as MDPs are discrete time processes. Secondly, the system is Markovian, which means that the transition from one state s to another state s' depends only on s and not on any other previous state [5]. The Markov property holds in our system because the transition depends only on the success probability of generating a link, and the currently existing links in the state. We detail how each component of the MDP is modelled below.

2.1 State Space

A state s uniquely characterises the relevant system information at a point in time. The state space $\mathcal{S}$ is the set of all possible states of the system. For entanglement generation, the relevant information is the number of currently existing links and their respective fidelities. As fidelity is a continuous value, modelling the system in this way leads to an infinite state space, which can make it computationally expensive to use RL.

In order to deal with the infinite state space, it is useful to consider a specific noise model by which the fidelity decays.

In this paper, we make the assumption that the noise model of the memory is a depolarizing channel [2], and the fidelity F of an entangled link undergoing depolarizing noise for t time steps is:

$$F(t) = (F_0 - \frac{1}{4})e^{-t\Gamma} + \frac{1}{4}, \tag{1}$$

where F_0 is the initial fidelity and Γ is the decay rate. The number of time steps for which an entangled link with initial fidelity F_0 will exist is given by:

$$t = \left\lceil \frac{1}{\Gamma} \ln \left(\frac{F_0 - \frac{1}{4}}{F_{thresh} - \frac{1}{4}} \right) \right\rceil, \tag{2}$$

where $\lceil . \rceil$ is the ceiling function and F_{thresh} is the threshold fidelity for discarding a link. The ceiling function arises from the fact that time step sizes are discrete. Entangled links with fidelities that result in the same number of time steps to exist can be grouped into bins. A state can then be represented as the number of links in each fidelity bin. If F_{thresh} is larger than $\frac{1}{4}$, only a finite number of bins are required to represent a state, equal to the highest number of time steps for which any generated link can survive. Thus, we can model the state space as

$$\mathcal{S} = \{(n_{t_{max}}, \ldots, n_2, n_1) | \sum_{i=1}^{t_{max}} n_i \leq N_{links}\}. \tag{3}$$

Here, n_i represents the number of links in bin i, t_{max} is the time to live of the highest fidelity bin, calculated by (2) for the highest fidelity link that can be generated in the system, and N_{links} is the desired number of links in the environment. The starting state is a state where all fidelity bins contain zero links. A terminating state is any state where the sum of all bins equals the desired number of links.

As a note to the reader, in this paper we order a state such that the leftmost entry represents the highest fidelity bin, and the rightmost entry represents the lowest fidelity bin. As an example, for an environment which can be represented by four bins, the state $(1, 0, 0, 0)$ represents a single link at the highest fidelity bin, which will survive four time steps including the time it was generated. We index the bins in descending order: bin four is the left-most bin, and bin one is the rightmost bin.

2.2 Action Space

The action space $\mathcal{A}$ is the set of actions that can be taken in the environment. In our system, the actions are represented as the particular tuning of the protocol, characterised by its success probability p and resulting fidelity F. Importantly, we model the relationship between the fidelity and probability as a linear trade-off where $F = 1 - \lambda p$, for some parameter $\lambda > 0$ [2]. This trade-off is motivated through a heralded single-photon entanglement generation protocol [6].

We consider two cases for the action space: continuous and discrete. A continuous action space is represented as a parameter $p \in (0, \min(\frac{1-F_{thresh}}{\lambda}, 1))$ signifying the success probability of link generation. p cannot exceed $\frac{1-F_{thresh}}{\lambda}$ as the fidelity would drop below the threshold fidelity. A link generated with probability p would be placed in a bin as calculated by (2). The second case is a discrete action space, which can be represented as a finite list of fidelity-probability pairs $[(F_1, p_1), (F_2, p_2), ..., (F_n, p_n)]$. The relationship between every F_i and p_i still obeys the linear trade-off $F_i = 1 - \lambda p_i$. The discrete action space can be thought of as points on the continuous action space.

With the use of fidelity bins to model the environment, an insight can be made about the action space. Bins consist of a range of fidelities, meaning multiple probabilities can correspond to the same bin. Two protocols with different probabilities which place a link in the same bin have the same outcome, but one has a higher probability of success. Physically, the entangled link with the lower probability (and thus higher fidelity) will indeed survive longer than the other, but will nevertheless decay below the threshold by the time the next generation attempt is completed. If there are two protocols with different probabilities but the resulting entangled links are placed in the same bin, then a policy can be improved by always choosing the higher of the two probabilities. Subsequently, policies in continuous and discrete action spaces may be able to perform equally if the discrete actions are chosen to be the highest probability that places a link in each bin.

2.3 Transition Function

A transition function $P_a(s, s')$ gives the probability that a given action a while in state s will lead to some other state s'. In our MDP, taking an action (F, p) has one of two outcomes: successful or unsuccessful generation of a link. With probability $1 - p$, generation is unsuccessful, and all existing links in memory will decay by one time step. Any link that decays below the threshold fidelity is discarded. With fidelity bins, the state $(n_{t_{max}}, \ldots, n_2, n_1)$ will become $(0, n_{t_{max}}, \ldots, n_2)$. As an example, in the state $(1, 0, 0, 1)$, the leftmost link with the highest fidelity would move to the next bin, and the last link would be discarded, leaving the state $(0, 1, 0, 0)$. In the case of successful generation with probability p, the same process occurs, and a new link is added to the bin t as calculated by (2).

2.4 Reward Function

The reward function $R(s, a)$ assigns a value to a given state and action pair. The RL agent uses the reward to calculate the benefit of taking an action in a state. Therefore, the reward function should be chosen such that it represents the objective of the problem. In this case, the goal is to minimise the time to deliver the desired number of links. Furthermore, all terminal states are equally acceptable, hence we are not interested in the specific configuration of the entangled links. Thus, the reward function is modelled such that any non-terminal state is given a reward of -1, and a terminal state is given a reward of 0.

2.5 Policy

A policy is a function $\pi(s)$ which maps a given state s to an action a. The goal of this problem is to find a policy that minimizes the time to achieve the desired number of entangled links. Methods such as value iteration can be used to find the exact optimal policy [5], but with larger state and action

spaces, this method quickly becomes computationally infeasible. Thus, RL is a useful approach for finding approximate solutions.

3 Methodology

This section contains a brief explanation of RL and the specific agents that were used in this paper, as well as the explanation of our heuristic. Two RL algorithms were used, one for a discrete action space, and one for a continuous action space. The agents were implemented using TensorFlow's TF-Agents library, which provides customisable standard implementations of the agents [7].

3.1 Reinforcement Learning

Many RL algorithms follow a common feedback loop. Understanding this process can help understand the differences of the specific algorithms. In RL, there is an agent and an environment. The agent chooses how to interact with the environment by picking actions, and the environment consists of what the agent cannot control, such as the available states, actions, transition probabilities, and rewards. The agent observes some initial starting state and picks an action based on its policy. The action causes some change in the environment, which gives feedback by outputting a new state and a reward. The agent uses this feedback to improve the approximation of some function, which depends on the type of algorithm. For example, an agent can approximate the action-value function, which assigns a value to taking an action in a state. The agent quantifies how close this approximation is to a target (using the environment feedback) with a distance metric. The agent minimises this distance with an optimisation algorithm, updating the parameters for its function approximation. With the new function approximation, the agent derives a new policy, picks a new action, and repeats the loop [5].

3.2 Categorical Deep Q-Networks

For the discrete action space, a Categorical Deep Q-Network (CDQN) was used [8]. CDQN works by estimating the action-value distribution, which is the distribution of returns from taking an action in a state. The return is the reward from taking the action, as well as the sum of the future rewards from the states that can be reached after taking the action. Regular Deep Q-Networks only model the expectation of the return, which may not account for actions whose value distributions are multimodal. We believe this is relevant in the entanglement generation environment, where a successfully generated link can immediately complete the environment, or lead to all the links being discarded (if, for example, there is only one link left to generate and the existing links will expire in two time steps). The optimisation algorithm used to update the value distribution is Adam (Adaptive Moment Estimation) [9], which is a widely used algorithm in RL due to its computational efficiency and robust performance.

3.3 REINFORCE

For the continuous action space RL algorithm, we use REINFORCE [10]. This algorithm was selected due to its ease of implementation within the project's time constraints. While we attempted to implement more advanced algorithms, such as Soft Actor Critic [11] and Proximal Policy Optimisation [12], we faced challenges preventing them from being used.

REINFORCE is a policy gradient algorithm, meaning the policy is not calculated using an action-value function, as is the case in CDQN. Instead, REINFORCE uses a neural network to directly approximate the policy function. This allows REINFORCE to operate on continuous action spaces, since it does not have to estimate the value of actions in every state. The policy function is updated by minimising the distance from a target. Like CDQN, the optimising algorithm that was chosen for the agent is Adam. TF-Agents implements episodic REINFORCE, which means that the agent requires every time step from the starting state to the terminating state to update, rather than every few steps like CDQN. The pseudocode can be seen in Algorithm 1 [5].

Algorithm 1 REINFORCE Algorithm

input: $\gamma \in [0, 1], \alpha > 0$
Initialize policy parameters θ randomly
for $n = 0$ **to** $N_{iterations}$ **do**
  Collect episode $\tau = s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T$ using policy π_θ
  for $t = 0$ **to** $T - 1$ **do**
    $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_{k+1}$
    $\theta \leftarrow AdamUpdate(\theta)$
  end for
end for

3.4 Heuristic Policy

One of the aims of this paper is to use the RL policies to create a general heuristic that can be applied to a wider set of parameters without the need to run intensive training. Since our heuristic is tested alongside the RL policies and other baselines, an explanation is provided in this subsection, while we relate it to the RL policies in Section 5. We first give an intuition for the heuristic and then explain the details of how it works.

The heuristic can be motivated by thinking about the outcomes for links. Either a link has a possibility of surviving until the end of the generation process, or it will be discarded beforehand. The second case is simpler than the first; if we do not expect a link to survive, we can treat the state as if that link does not exist. A trivial example would be the state $(0, 0, 0, 3)$ when four links are desired. No matter which action is picked, the three links in the last bin will decay below the threshold fidelity and be discarded. There is no possibility of the links existing until four links are generated. Thus, the policy can ignore these links and treat this state the same way it would treat the $(0, 0, 0, 0)$ state.

If a link has a possibility to survive, then we have three cases for generating the next link. We can generate at a higher, lower, or equal fidelity bin as the first link. The heuristic policy is to generate the new link at the equal fidelity bin as the previous link, except for the last link, which it generates with the highest possible probability. The reason for not

generating at a higher fidelity bin is that since we already expect the previous link to survive, the next link does not need to survive longer. The reason for not generating at a lower fidelity bin is that this does not make use of the extra time steps that the previous link will survive for. When there is only one link left to generate, the environment ends as soon as the link is generated, so the heuristic generates it at the highest probability possible.

In order to determine whether an entangled link has a possibility of surviving, it must survive at least as many time steps t as the number of links left to generate N_{left}, calculated as $N_{left} = N_{total} - N_{current}$. Here, N_{total} is the total number of desired links, and $N_{current}$ is the current number of links that survive at least t time steps. $N_{current}$ can be calculated as $N_{current} = \sum_{i=t}^{t_{max}} n_i$, where n_i is the number of links at bin i, and t_{max} is the highest fidelity bin's time steps. For a heuristic that can work on any given state, we need to find the lowest fidelity bin which contains links that meet the condition $t \geq N_{left} + 1$, which we do by iterating from the lowest fidelity bin to the highest fidelity bin. Once we find the bin t that meets the condition, we pick the action with the highest probability p that will generate an entangled link at bin $t-1$. The reason for picking one bin lower is that the link currently at t will decay to $t-1$ once our generation attempt is completed. For a continuous action space, we can use (1) to find the probability p as

$$F_{thresh} = (F - \frac{1}{4})e^{-(t-1)\Gamma} + \frac{1}{4},$$

$$F = (F_{thresh} - \frac{1}{4})e^{(t-1)\Gamma} + \frac{1}{4}.$$

Using the probability-fidelity trade-off $F = 1 - \lambda p$ yields

$$p = \frac{1}{\lambda}(\frac{3}{4} - (F_{thresh} - \frac{1}{4})e^{(t-1)\Gamma}).$$

Our heuristic is described here for a continuous action space, although it can be adapted to any set of actions where there is an action corresponding to each fidelity bin. Instead of calculating the highest probability to place a link in a bin, the heuristic can choose the action corresponding to bin $t-1$.

For the starting state with no links, or any state without a link satisfying the remaining link condition, the heuristic picks the generation probability p_{in} , the initial probability, which is a parameter of the heuristic and is defined by the user. The pseudocode can be seen in Algorithm 2.

As an example, the state $(1, 0, 0, 0)$ has a link in bin four, and $N_{left} = 3$. Then, $\pi((1, 0, 0, 0))$ will output a probability for a link that will go in the third bin.

Algorithm 2 Heuristic Policy

input: $s \in \mathcal{S}, p_{in}, p_{max} \in \mathbb{R}$
$a \leftarrow p_{in}$
for $i = 1$ **to** t_{max} **do**
  $N_{left} \leftarrow N_{total} - \sum_{j=i}^{t_{max}} n_j$
  if $n_i > 0$ **and** $i \geq N_{left} + 1$ **then**
    if $N_{left} == 1$ **then**
      $a \leftarrow p_{max}$
      return
    end if
    $a \leftarrow \min\left(\frac{0.75-(F_{thresh}-0.25)e^{\Gamma(i-1)}}{\lambda}, p_{max}\right)$
    return
  end if
end for

4 Experimental Setup

In this section, we discuss the experimental setup of the simulation as well as presenting and discussing the results. The simulation has many parameters, and it is difficult to individually motivate each parameter, especially for future networks with low decay rates. Thus, experiments have been carried out in two parameter regimes of the simulation. The first is a high decay environment with a small amount of required entangled links. This simulates a closer to near term network, where large applications cannot be carried out due to noise. The second is a lower decay environment with a medium number of links required. The idea behind this regime is to allow more actions which can benefit from longer decay times to explore the trade-off between the fidelity and probability of success.

Two policies are used as baselines, a random policy and a single action policy. The random policy chooses a random probability at every state as the action. The single action policy chooses the same probability for every state. The probability is chosen as the action which leads to the smallest average time to achieve the desired links out of all possible actions. This is determined by evaluating the single action protocol with probabilities at each bin.

An aim of this paper is to find the effect of having more available actions. An important concern is how those actions are chosen. One possible method is sampling equally spaced points from the continuous action space. However, this runs into the limitation of placing multiple actions in the same bin, or picking actions that have low probabilities in their respective bins. In Subsection 2.1, we identified that the action space can be reduced to the largest probability in each bin. Even with this, there are still many ways to pick some actions in some number of bins.

In this paper, we have opted to pick actions in decreasing order of bins. If the network has four bins, decreasing order would mean picking the highest probability action in bin four, then bin three, then bin two, etc. This allows for picking between one and four actions. This method of picking actions ensures that all policies have the highest fidelity action available. If this were not the case, there might not be an action that can generate links which can last long enough to generate the desired number of links. For example, for an environment with four bins and four desired links, a policy requires at least one action which can generate a link that can survive for four time steps. Additionally, with this order of picking actions, we guarantee that a policy can be at least as good as a policy with fewer actions. For example, a four action policy has one additional action as compared to a three action policy. If this additional action turns out to not be useful, the four action policy can perform equally to the three action policy by picking the same policy. This allows us to have a better idea of

how useful each additional action is.

Additionally, since it is possible for the policy to repeatedly pick infeasible actions that cannot lead to the desired number of entangled links (i.e. a policy generating links that expire immediately after generation cannot possibly generate two or more links simultaneously), a time step limit is set in the environment. If the agent is not able to achieve the desired number of links by the time step limit, the environment is terminated. This time step limit was set to be much larger than the single action baseline so that it would not interfere with finding the optimal policy.

5 Results

5.1 Near-Term Network with Four Links

The first environment is a network with high decay. In this environment, the highest fidelity link is able to exist for four time steps. Since we need four links, a link needs to be generated every time step. The parameters of this environment are $N_{total} = 4, \Gamma = 0.2, F_{thresh} = 0.5, \lambda = 0.7, p_{min} = 0.4, p_{max} = 0.7$. p_{min} and p_{max} are the minimum and maximum probabilities of any action possible in the environment. The maximum number of time steps for this environment was 500.

Model Hyperparameters

For the sake of reproducibility, the hyperparameters of both reinforcement learning algorithms are disclosed. Additionally, the code is made publicly available and can be found in Section 7.

For the near-term network, the CDQN algorithm was run for 20000 iterations, with 2000 initial randomly collected samples, a replay buffer size of 2000, two fully connected hidden layers of size 256, a batch size of 128, an optimiser learning rate of $3 \cdot 10^{-4}$, a discount rate of 1, with 81 atoms for the support of the value distribution, with atom minimum and maximum values being -500 and 0 respectively.

The REINFORCE algorithm was run for 2000 iterations, with a replay buffer capacity of 10000, two fully connected hidden layers of size 256, an optimiser learning rate of $3 \cdot 10^{-4}$, and a discount rate of 1.

Results and Discussion

Figure 1 plots the number of available actions against the expected time to generate the desired number of links. We observe that increasing the number of actions reduces the mean time to achieve the desired state.

Figure 2 compares the different policies in generating up to four links. We see that the difference between policies becomes especially significant at higher numbers of desired links. Unexpectedly, the continuous action RL policy performs worse than the discrete action space RL policy. In theory, we expect it to perform at least as well as the discrete policy (as it has the same actions available). However, in practice, there are a number of reasons which may cause it to perform worse. A fundamental reason why the continuous agent may perform worse is that the infinite action space introduces challenges for exploration. There are far more actions in every state, and the agent might fail to accurately evaluate which actions perform best. Other reasons could be limitations of the research method, such as insufficient hyperparameter tuning or unsuitable choice of algorithm. These reasons are addressed in Section 6. Notably, the heuristic seems to perform equally well as the discrete RL policy.

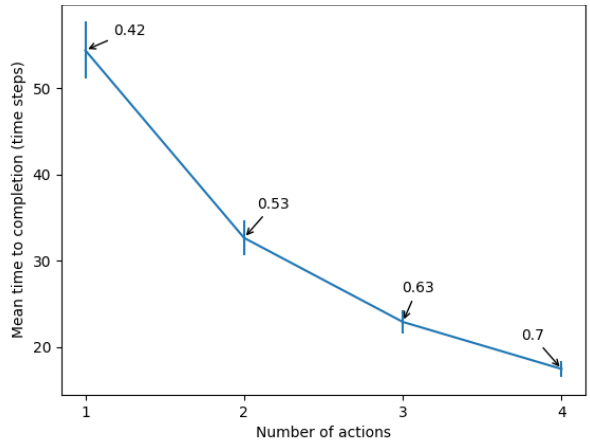

Figure 1: Action probabilities are chosen in decreasing order of bin number. The numbers in the figure are the success probability of the additional action compared to the previous number of actions. The one action policy is the single action heuristic, while the remaining policies come from the discrete action space RL agent. The values are averages over 2000 runs of the environment with a confidence interval of 99.7%.

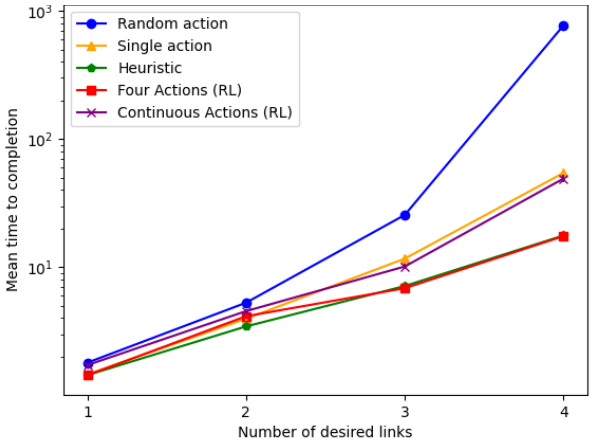

Figure 2: The y-axis uses a log scale. The single action probabilities for $1, 2, 3, 4$ desired links are $0.70, 0.63, 0.53, 0.42$, respectively. The heuristic initial probabilities are $0.70, 0.53, 0.42, 0.42$. For both single action and heuristic, the probabilities were chosen as the lowest mean time for the respective number of links. The values are averages over 2000 runs of the environment with a confidence interval of 99.7%. The confidence intervals are present, but too small to be seen.

Figure 3 visualises the RL policy for four available actions. We notice that the policy assigns higher probabilities to certain states. For example, all states which are assigned the highest probability action already have three out of four links. However, not all states with three links are assigned the highest probability. The states $(0, 1, 1, 1)$ and $(0, 0, 0, 3)$

have three links, but the policy chooses the same action for these states as for the starting state, $(0, 0, 0, 0)$. This indicates that the policy considers certain links redundant. This is also applicable for states assigned higher probabilities, for example, states $(1, 0, 0, 2)$ and $(1, 0, 0, 0)$ are assigned the same action. Two general rules are highlighted by this policy: ignoring redundant links, and choosing higher probability actions when there are valuable links. These two ideas have been used as motivations for the heuristic. It is observed that the policy does not always follow these rules, such as the state $(1, 0, 1, 1)$ which seems to share the same valuable first link as the previous two states, but is assigned the lowest probability action instead.

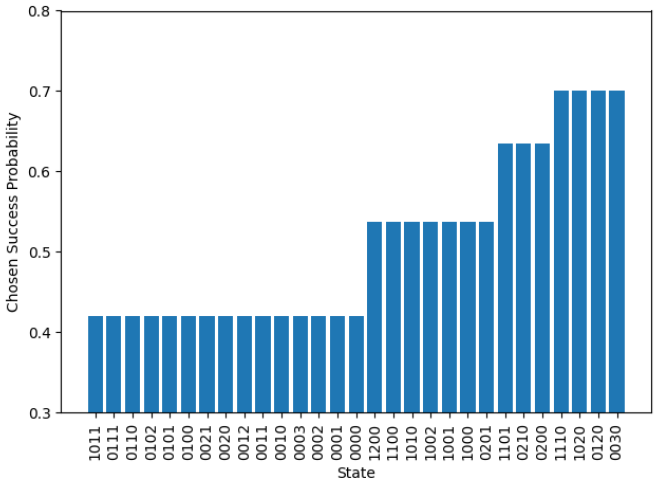

Figure 3: The x-axis is the state represented as a string. The state $(1, 0, 2, 0)$ is written as 1020. The y-axis is the probability that the policy chooses given the state. The states are ordered by chosen probability for easy comparison of states that have the same probability.

5.2 Medium-Term Network with Six Links

The second environment is a network which requires six links. In this environment, the highest fidelity link is able to exist for ten time steps. Compared to the previous environment, a link can exist longer than the number of minimum time steps required to achieve six links. This allows for policies which lower probability but longer lasting links. The parameters of this environment are $N_{total} = 6, \Gamma = 0.08, F_{thresh} = 0.5, \lambda = 1, p_{min} = 0.2, p_{max} = 0.5$. The maximum number of time steps for this environment was 1200.

Model Hyperparameters

For the near-term network, the CDQN algorithm was run for 100000 iterations, with 10000 initial randomly collected samples, a replay buffer size of 12000, two fully connected hidden layers of size 256, a batch size of 256, an optimiser learning rate of $3 \cdot 10^{-4}$, a discount rate of 1, with 81 atoms for the support of the value distribution, with atom minimum and maximum values being -1200 and 0 respectively.

The REINFORCE algorithm was run for 5000 iterations, with a replay buffer capacity of 10000, two fully connected hidden layers of size 256, an optimiser learning rate of $3 \cdot 10^{-4}$, and a discount rate of 1.

Results and Discussion

Figure 4 plots the number of available actions versus the mean time to completion. Unlike Figure 1, we see that the mean time to achieve the desired links does not decrease with every additional action. While it still trends down, the additional action can sometimes increase the mean time to completion, such as between actions eight to nine. As actions were picked in decreasing order of bins, the nine action policy is able to pick the same actions as the eight action policy, and so should be able to perform equally well. The discrete agent may be suffering from the same issues as the continuous agent in the near-term network setup.

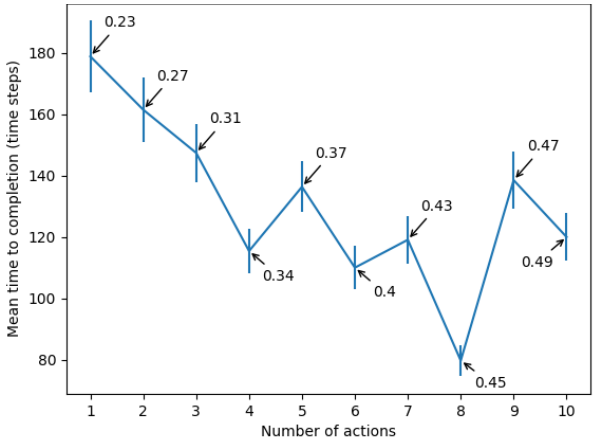

Figure 4: Action probabilities are chosen in decreasing order of bin number. The numbers in the figure are the success probability of the additional action compared to the previous number of actions. The one action policy is the single action heuristic, while the remaining policies come from the discrete action space RL agent. The values are averages over 2000 runs of the environment with a confidence interval is 99.7%.

Figure 5 plots the different policies against the baselines in generating between one and six links. Here we notice a similar order of performance as in the near-term setup. The heuristic outperforms the other policies.

Figures 6 and 7 visualise the policies from the eight action RL policy and the heuristic. The eight action RL policy is chosen as it is the fastest of all the RL policies, which we believe will allow for more meaningful comparisons with the heuristic. We observe that some trends are common to both policies. For example, both policies tend to prioritise high fidelity links rather than high probability links when the average fidelity is low. At higher average fidelities, the policies typically choose higher probability actions as the number of links increases. The RL policy is typically less structured. For example, for states with two links and an average fidelity of 0.54, the policy chooses the action with a success probability of 0.43. However, the time to live of this link is less than the number of links remaining, indicating this link will not survive. Thus, we see that the RL policy can make suboptimal decisions in certain states.

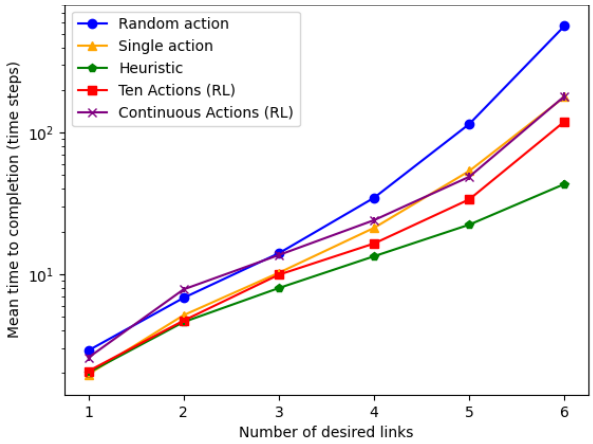

Figure 5: The y-axis uses a log scale. The single action probabilities for the six desired links in increasing order are $0.49, 0.40, 0.37, 0.34, 0.31$, and 0.27. The heuristic initial probabilities are $0.49, 0.40, 0.34, 0.31, 0.23$, and 0.23. For both single action and heuristic, the probabilities were chosen as the lowest mean time for the respective number of links. The values are averages over 2000 runs of the environment with a confidence interval is 99.7%. The confidence intervals are present, but too small to be seen.

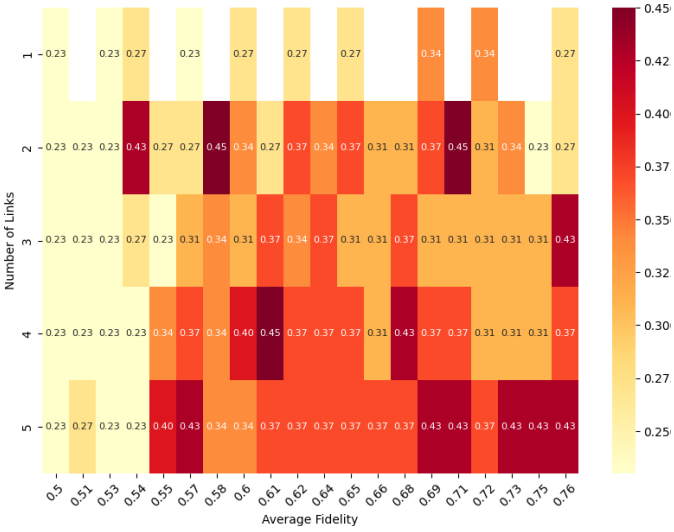

Figure 6: Heatmap of eight action policy with respect to number of links and the average fidelity of the state. The probability is the most commonly chosen probability for the average fidelity and number of links. States with average fidelities in between two labels were grouped into the closest average fidelity.

6 Conclusion and Future Work

In this paper, we have demonstrated how tuning entanglement generation protocols affects the time taken for resource generation in quantum networks. In particular, we show how much impact finer tuning of the protocols have. We evaluate the policies on two parameter regimes of the simulation, one representing a near term network with four desired entangled links, and one representing a medium term network with six desired entangled links. Thus, we conclude that tuning

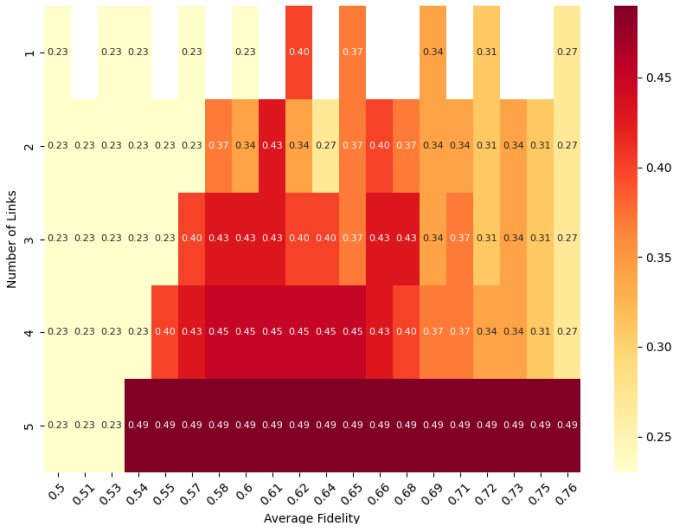

Figure 7: Heatmap of the heuristic policy with respect to number of links and the average fidelity of the state. The probability is the most commonly chosen probability for the average fidelity and number of links. States with average fidelities in between two labels were grouped into the closest average fidelity.

protocols is beneficial to significantly reduce the mean time to generate desired numbers of entangled links. Furthermore, finer tuning of the protocols can lead to faster times, up to a limit determined by the model. We have also presented a heuristic that matches the performance of the reinforcement learning policies.

For future research, improvements can be made to alleviate the limitations of the experimental setup in this paper. Some limitations are discussed below. Firstly, the continuous action space reinforcement learning algorithm used was REINFORCE. This is a simple algorithm for continuous action spaces, and was chosen due to practical limitations of the project regarding time and ease of implementation. The algorithm performs worse than the discrete action space, which in theory should serve as a lower bound for its performance. Future work can use state-of-the-art algorithms such as PPO, SAC, or TD3 which work in continuous action spaces to find better results. This would help verify whether the discrete and continuous action spaces are equivalent in this model, as well as work on other noise models where binning ranges of fidelities cannot be done.

Another limitation of this project lies in the method by which actions were sampled for the discrete action space. The selection of actions could have been worse than another particular selection. Then, it could be possible that fewer actions could perform almost as well as a higher number of actions. If a more appropriate method of selecting actions were explored, this could improve the computational efficiency of using reinforcement learning for different configurations of networks.

7 Responsible Research

In order to do research responsibly, it is crucial that the results in this paper are reproducible. Furthermore, it is important that all data is available and support the conclusions made.

To this end, all environment parameters and model hyperparameters have been stated in the paper. Furthermore, all the code used, and all trained models can be found at https://github.com/atacettin/entangled-link-generation.

Due to the stochastic nature of the simulation, results may vary. To amend this, all simulation statistics have been averaged over 2000 runs, and confidence intervals of 3 standard deviations are included. Thus, we believe we have representative results to make our conclusions with.

References

[1] Stephanie Wehner, David Elkouss, and Ronald Hanson. Quantum internet: A vision for the road ahead. *Science*, 362(6412):eaam9288, October 2018.

[2] Bethany Davies, Thomas Beauchamp, Gayane Vardoyan, and Stephanie Wehner. Tools for the analysis of quantum protocols requiring state generation within a time window, April 2023. arXiv:2304.12673 [quant-ph].

[3] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 1 edition, June 2012.

[4] Álvaro G. Iñesta, Gayane Vardoyan, Lara Scavuzzo, and Stephanie Wehner. Optimal entanglement distribution policies in homogeneous repeater chains with cutoffs. *npj Quantum Information*, 9(1):46, May 2023.

[5] Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts London, England, second edition edition, 2020.

[6] Sophie L. N. Hermans, Matteo Pompili, Laura D. S. Martins, Alejandro R.-P. Montblanch, Hans K. C. Beukers, Simon Baier, Johannes Borregaard, and Ronald Hanson. Entangling remote qubits using the single-photon protocol: an in-depth theoretical and experimental study. *New Journal of Physics*, 25(1):013011, January 2023. arXiv:2208.07449 [quant-ph].

[7] Danijar Hafner, James Davidson, and Vincent Vanhoucke. TensorFlow Agents: Efficient Batched Reinforcement Learning in TensorFlow, 2017. Version Number: 2.

[8] Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning, July 2017. arXiv:1707.06887 [cs, stat].

[9] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014. Version Number: 9.

[10] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, August 2018. arXiv:1801.01290 [cs, stat].

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. arXiv:1707.06347 [cs].