Quantum Machine Learning, Homework 2

- Author: Yan-Tong Lin, 0712238, National Chiao Tung University
- Lecturer: Dr. Alexey Melnikov, Principal Investigator, Terra Quantum

- Online Version at https://hackmd.io/MkTt8-07SiWvv-p0LqR6Pw (https://hackmd.io/MkTt8-07SiWvv-p0LqR6Pw)
- Jan 3rd, 2021

- Quantum Machine Learning, Homework 2
  - Note for Day 3 (12/4)
    - Overview
    - PS (cont.)
    - PS agent on Quantum System
  - Note for Day 4 (12/5)
    - Overview
    - PS code study
  - Note for Day 5 (12/11)
    - Overview
    - Quantum Speed up for PS
    - Quantum Communication Tasks with PS
  - Note for Day 6 (12/12)
  - Materials
  - HW2 Problem Set
    - Q1
    - A1
    - Q2
    - A2
    - Q3
    - A3

Note for Day 3 (12/4)

Overview

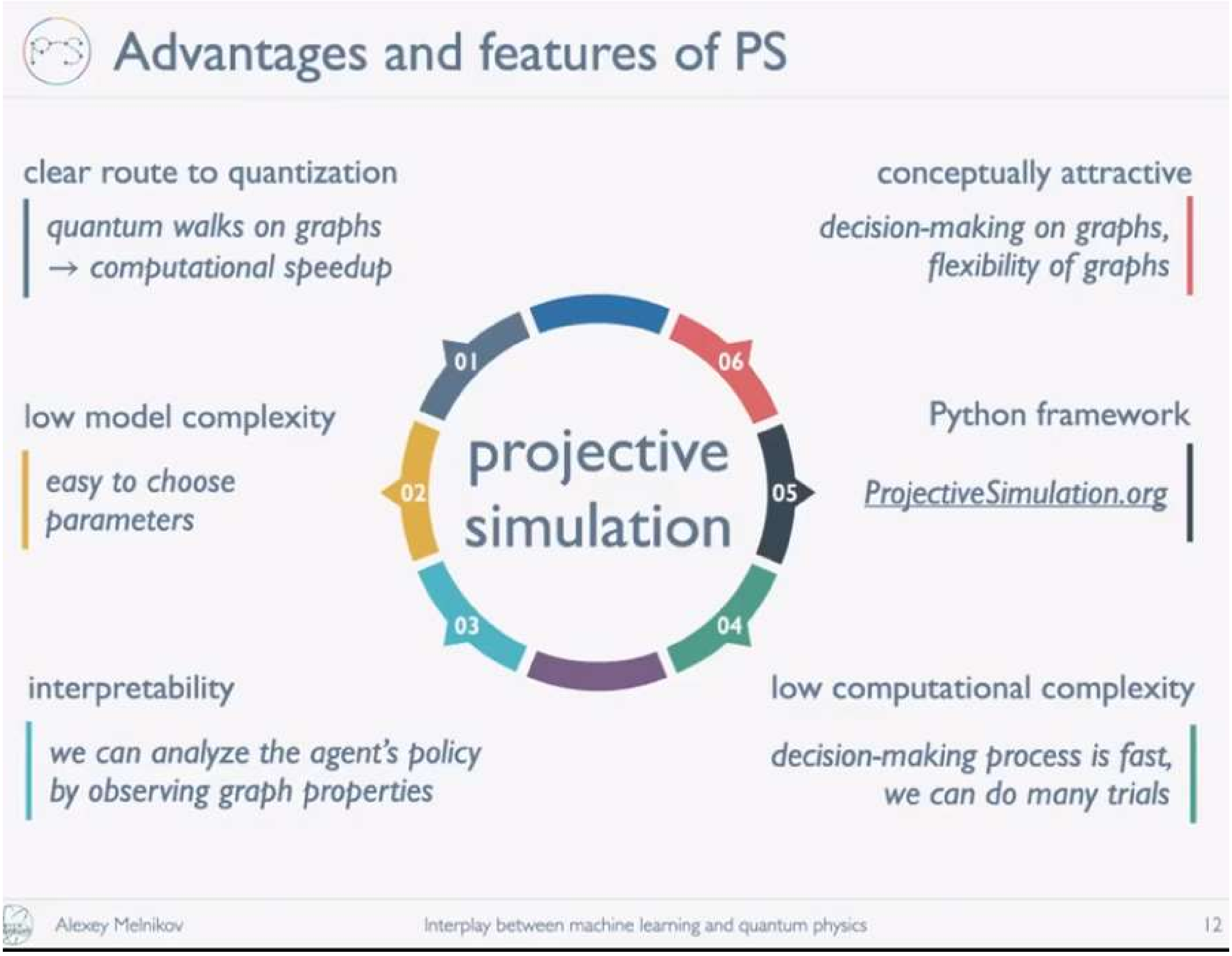
- More on PS

  - analytic solution
  - cost efficiency
  - meta-param net
  - applications
- PS applied in quantum labs
  - definitions
  - the task is hard
  - the model
  - the results

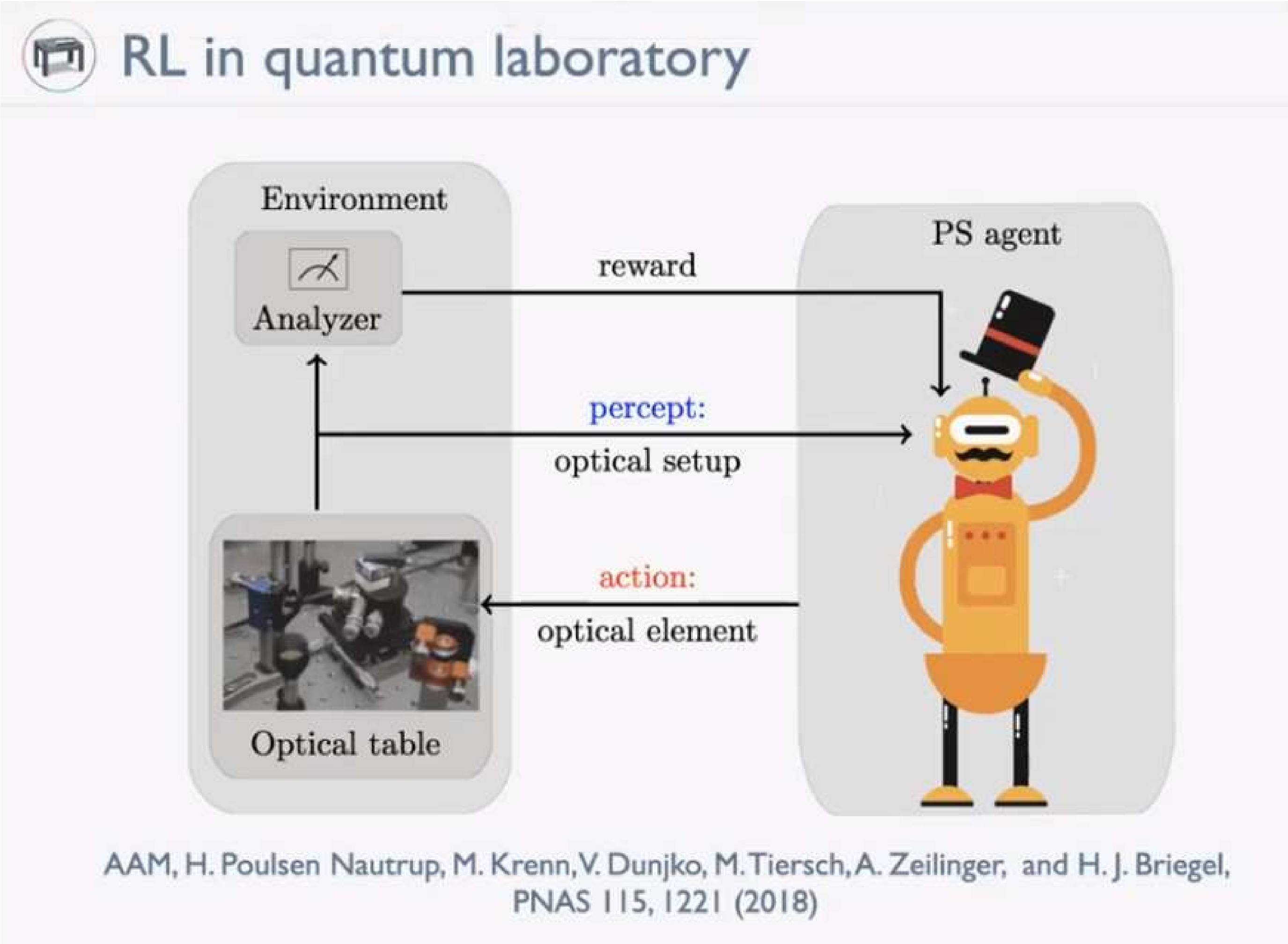
PS (cont.)

- Analytic solution for convergence of PS agents
  - a paper found
  - On the convergence of projective-simulation-based reinforcement learning in Markov decision processes (https://arxiv.org/pdf/1910.11914.pdf)
- PS and its cost efficiency V.S. Q-learning and Sarsa
  - Easier to optimize
  - Fewer meta-param
- PS with meta-param controller
- PS with generalization clips

- PS and their applications



PS agent on Quantum System



- The task: creating photonic quantum experiment
    - actions: state transition
    - reward: given by analyzer (human)
- The model
    - naive + clip composition and deletion
- The experimental results
    - PS is effiecient compared to previous methods
    - analysis of the memory clips (with strong connectivities)
        - rediscover useful subroutines!
    - conclusion (PS help to)
        - find interesting experiments
        - find shorter implementation
        - rediscover subroutines

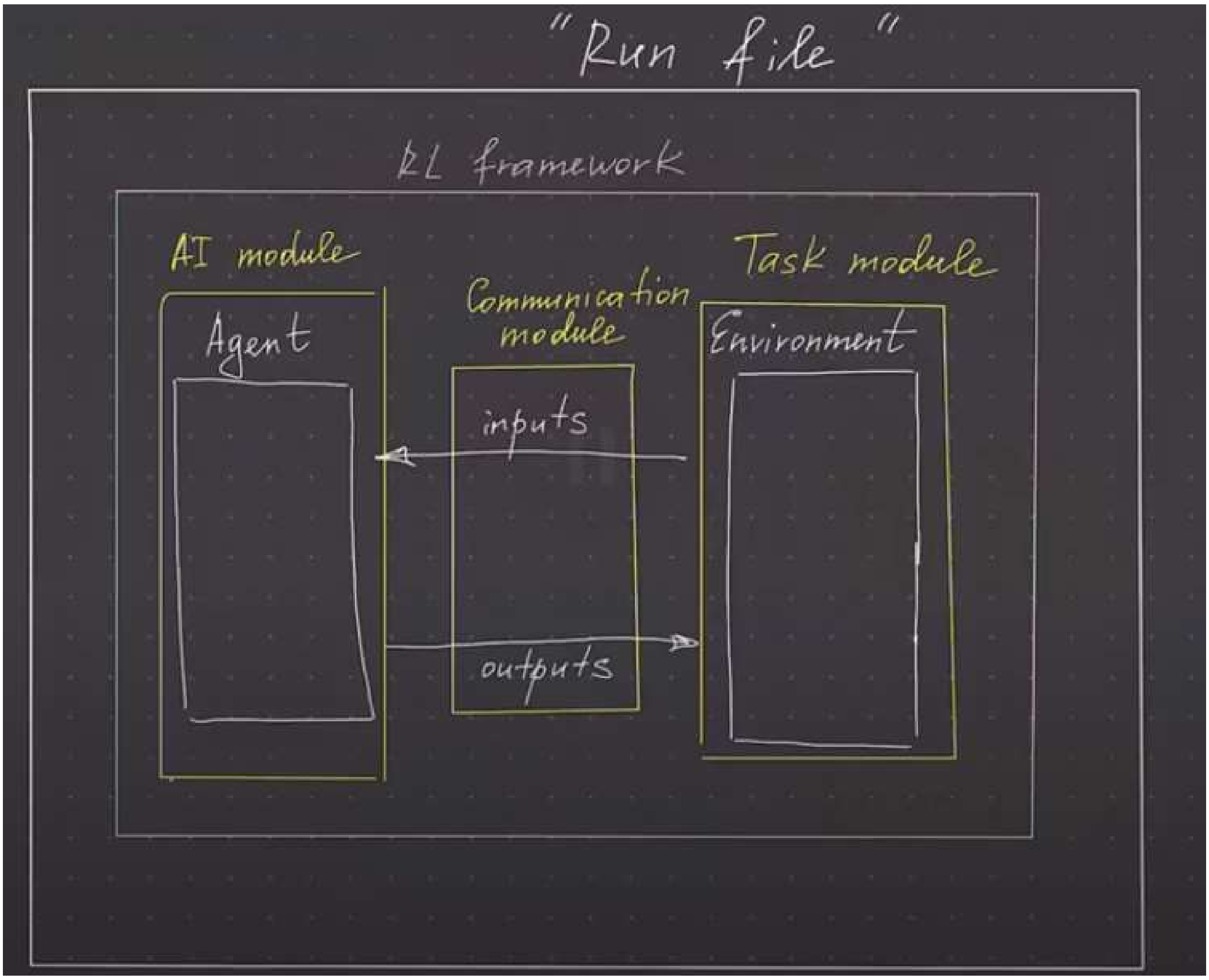
Note for Day 4 (12/5)

Overview

- PS code study

- PS applied to Quantum (practice)
- Quantum Enhanced PS

PS code study

- modularization
  - 

- 4 types of agents (categorized by clips)
  - basic (as in lecture)
  - flexible (states are unknown and can be added, indexed by hash function)
  - sparse (the sparse version of basic agent)
  - generalization (internal clips by state properties, specified by the user)
    - there is some interesting mathematical analysis, by is not covered
    - code is too much to go through line by line in the lecture
- Environments
  - grid world
  - open ai (API)
- Interaction module
  - interaction
- For more information
  - projection simulation with generalization

  - benchmarking PS v.s. Q-learning/SARSA
  - PS for experiment design
  - ML for quantum communication
  - RL for violating Bell's inequalities
- The code for the papers is the one that just covered with some twist and the papers are accepted immediately for the contributions to the community

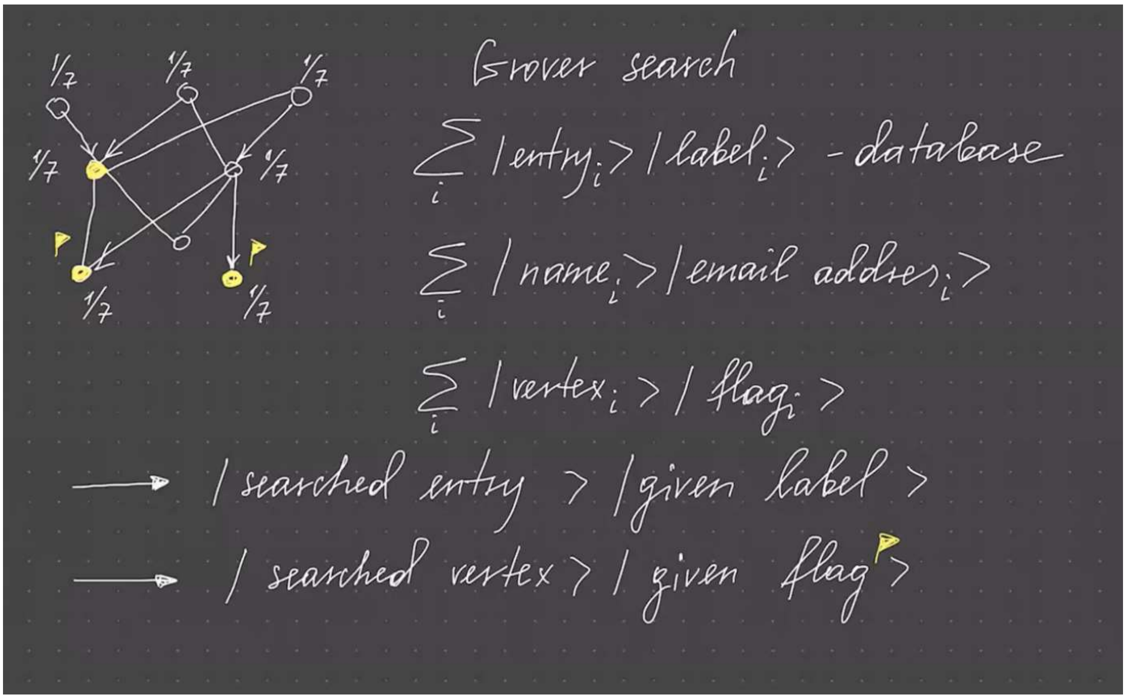
Note for Day 5 (12/11)

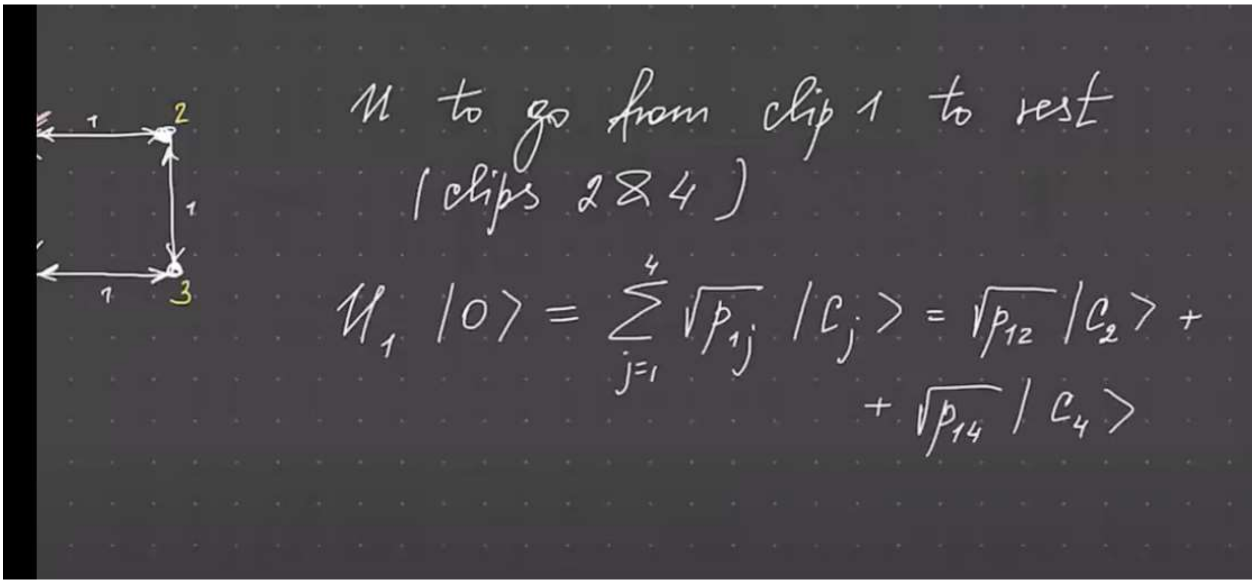
Overview

- Quantum Devices to Speed-up PS (Q for ML)
- Quantum Communication Tasks with PS (ML for Q)
  - Generalized Bell Test

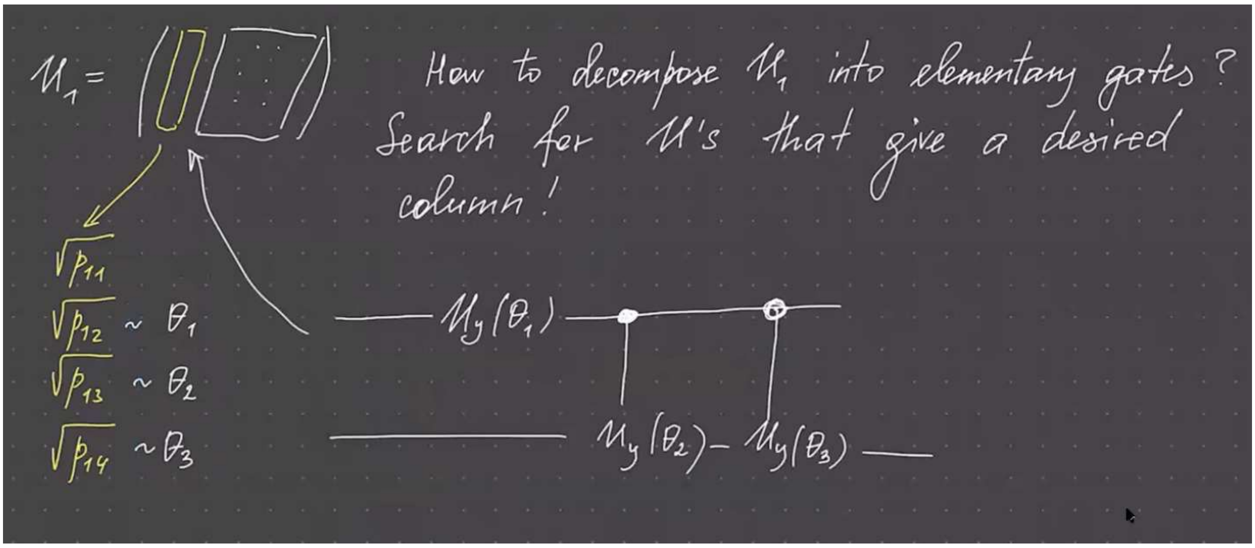
Quantum Speed up for PS

- Refs
  - Quantum Speedup for Active Learning Agents
    - an obvious application of Grover's Search
  - Quantum-enhanced deliberation of learning agents using trapped ions
  - Coherent controlization using superconducting qubits
- quadratic speed up for
  - it takes random walk $t = \frac{1}{\delta}$ to converge to the uniform distribution of all vertex
  - it takes quantum walk $t = \frac{1}{\sqrt{\delta}}$ to converge to uniform distribution of all vertex
  - if the distribution is a uniform distribution, with ϵ prob we will find the desired state
    - this can speed up using amplitude amplification too



- Question:
  - Quantum Walk first or Projective Simulation First? (quantum motivated PS?)
- Amplitude Amplification, in this case, will preserve the h value's effect so that end with a high reward is more likely to be observed
- Quantum Walk Formalism (and an example)
  - $U_i|0\rangle = \sum_j \sqrt{p_{ij}}|j\rangle$
  - 

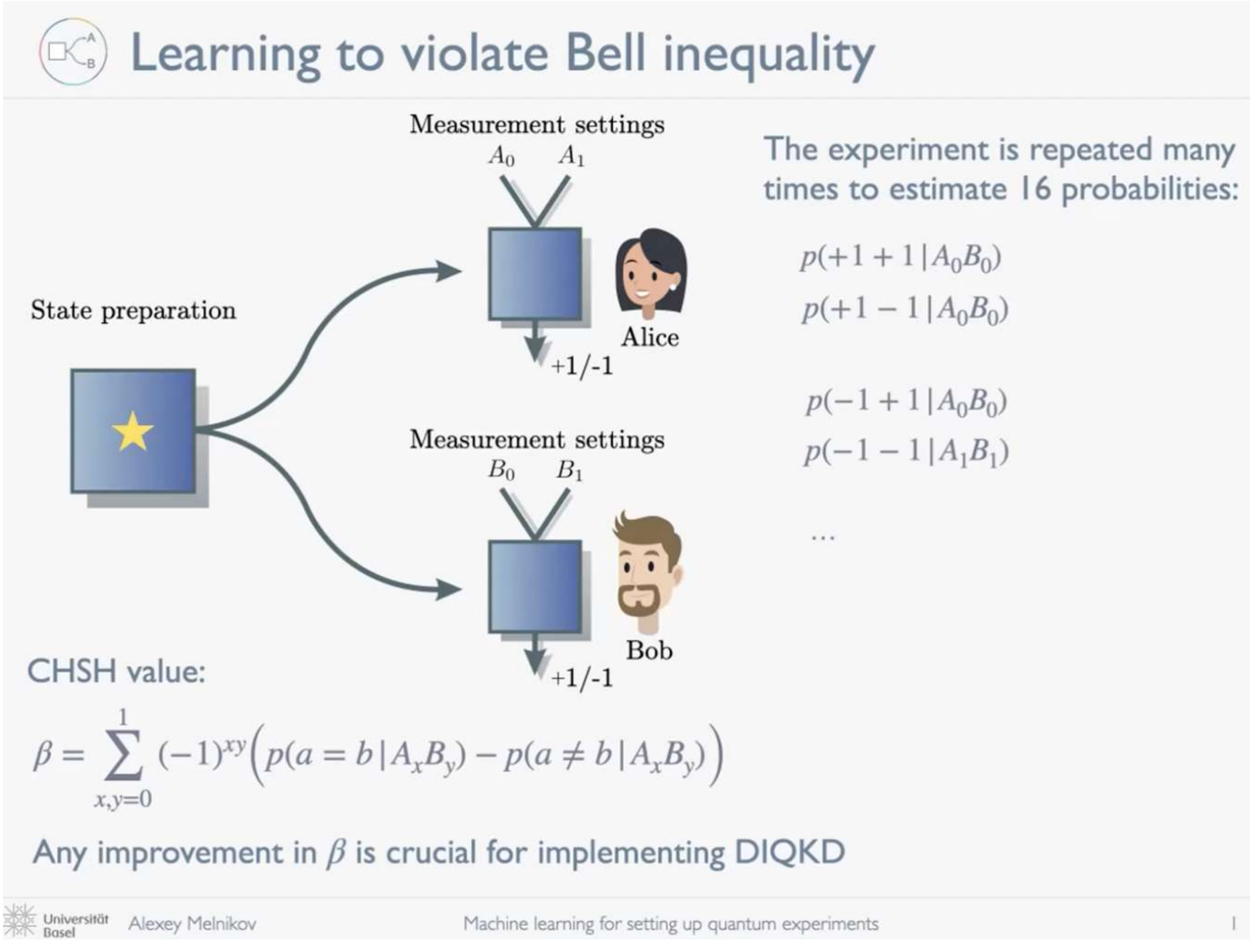
- Circuit Synthesis

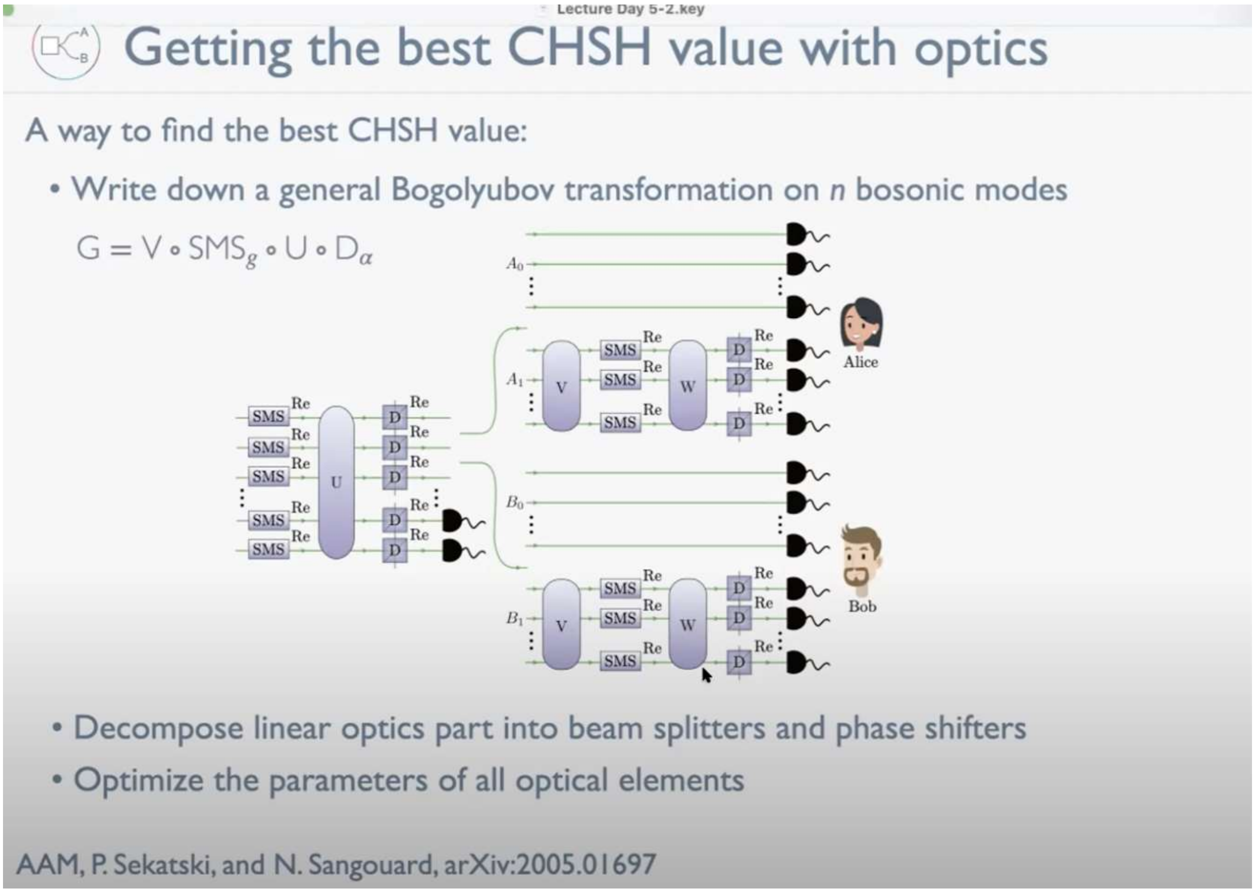
- 

    - Q: how to implement control-U?
    - I am not of physics background, this part is not clear for me. If this will be a crucial part of my future research, I will fill it up.
- Conclusion
    - Any decision making on N clips can be implemented on $\log(N)$ qubits
    - We can prepare stationary distribution ($Tp = p$) faster (w.r.t. spectral gap δ)
        - classical: $t = \frac{1}{\delta}$
        - quantum: $t = \frac{1}{\sqrt{\delta}}$
    - We can find desired actions faster (w.r.t. the fraction of desired state ϵ)
        - classical: $t = \frac{1}{\epsilon}$
        - quantum: $t = \frac{1}{\sqrt{\epsilon}}$

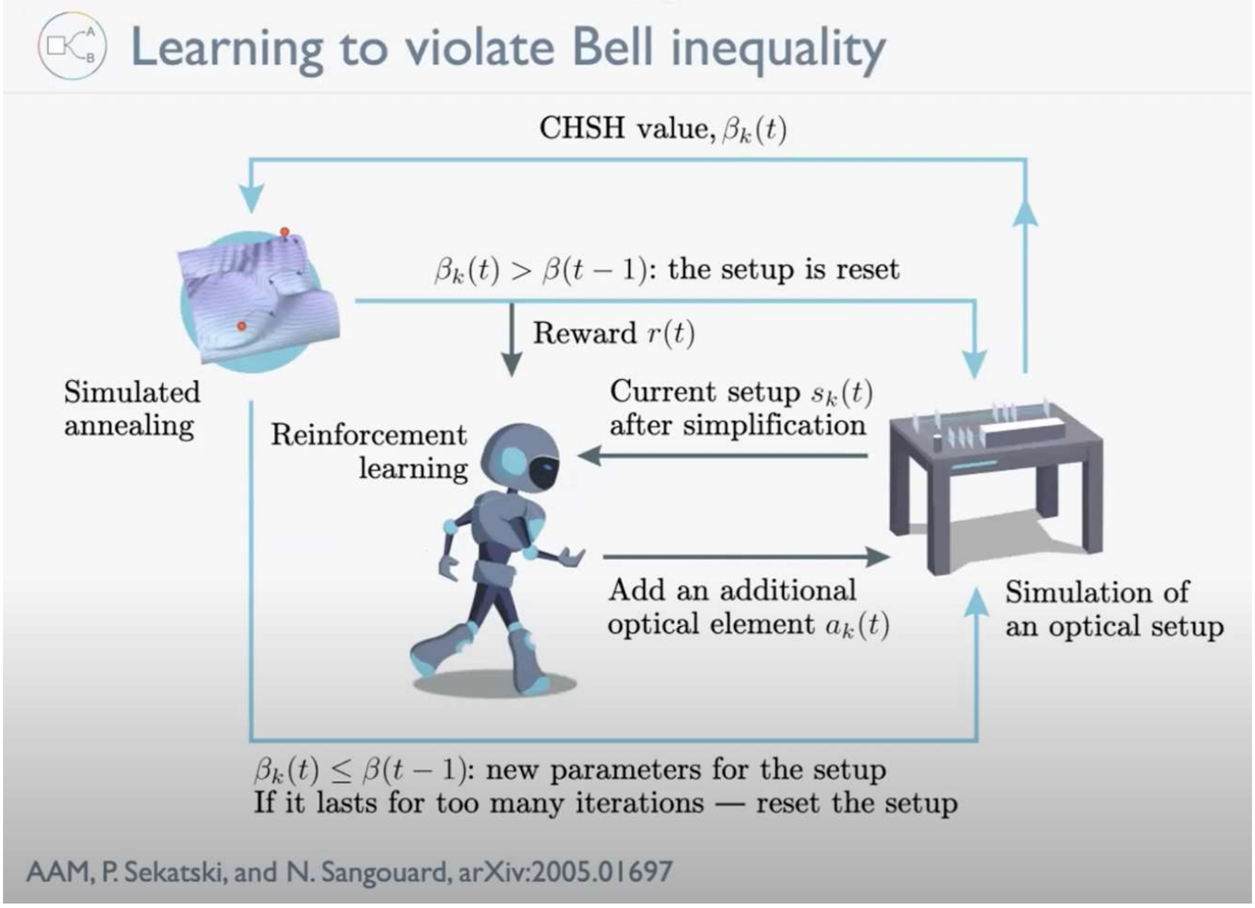
Quantum Communication Tasks with PS

- bell test

- 

- generalized bell test
  - 

  - my observation: this different from previous examples, the sapce is continuous
- the framework

- 

- simulated annealing
  - to find a extreme value in a n dimensional space
  - 1. random initialization
  - 2. choose by prob which to change
  - 3. choose by prob change how much
  - 4. get the new β'
    - $\beta' < \beta$, accept the change
    - $\beta' \leq \beta$, accept the change with prob $e^{\frac{\beta'-\beta}{\tau}}$

Note for Day 6 (12/12)

- For (continuous-time) quantum walk, I actually did a survey for a related paper, the link is shown below
  - https://github.com/EazyReal/QCQI2020fall/blob/main/QCQI_final__Oral_Version_.pdf (https://github.com/EazyReal/QCQI2020fall/blob/main/QCQI_final__Oral_Version_.pdf)
- For CNNs, I have taken DNN courses so that I am quite familiar with them
- The paper tries to predict when will a quantum speedup happen (graph $\rightarrow$ speed-up?)

- Predicting quantum advantage by quantum walk with convolutional neural networks

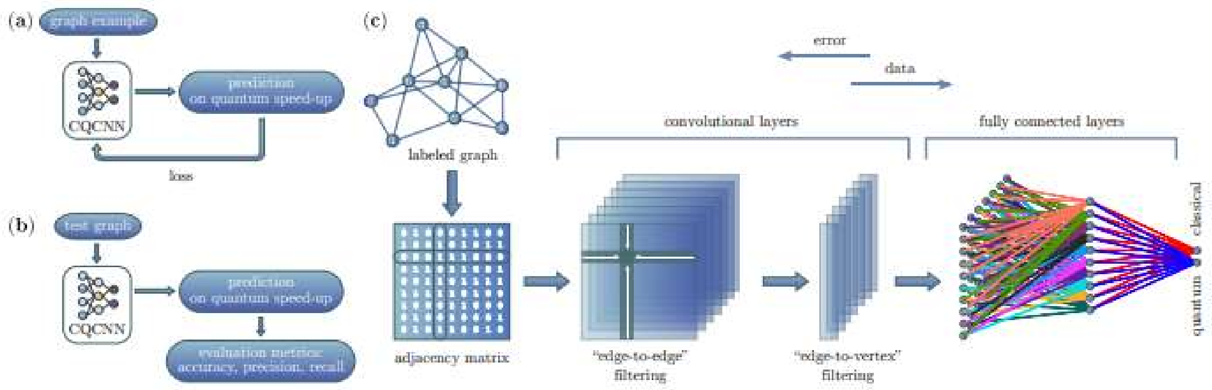


FIG. 2. The machine learning approach that is used for predicting the quantum speedup. (a) A process of training CQCNN. (b) A process of testing CQCNN. (c) A scheme of the CQCNN architecture. The neural network takes a labeled graph in form of an adjacency matrix as an input. This input is then processed by convolutional layers with graph-specific "edge-to-edge" and "edge-to-vertex" filters (see Methods). The convolutional layers are connected with fully-connected layers that finally classify the input graph. The number of layers is the same for all graph sizes. Data and error propagation are shown with arrows.

Materials

- Quantum enhancements for deep reinforcement learning in large spaces
- On the convergence of projective-simulation-based reinforcement learning in Markov decision processes
- Predicting quantum advantage by quantum walk with convolutional neural networks

HW2 Problem Set

- For the code, please visit my GitHub repo (https://github.com/EazyReal/Quantum-Machine-Learning-2020-fall)
  - Grid World
    - For finding the optimal $\eta = 0.1$, visit `run_grid.py`
    - For visualizations, visit `Single Agent in Grid World.ipynb`
  - Grover's Diffusion Operator Synthesis
    - For the environment implementation, visit `environments/env_quantum_circuit_synthesis.py`
    - For finding the optimal $\eta = 0.9$, visit `run_grover.py`
    - For visualizations, visit `Single Agent doing Grover's Diffusion Operator.ipynb`

Q1

Go through the material which we considered during the lectures. Think of what is not clear and ask questions.

A1

- Quantum speedup of projection simulation with quantum walk is interesting, and here is my question. Do the authors use projection simulation to make RL a random walk problem so that we can have a quantum speedup? Or the authors come up with projective simulation first and find that they are random walks?
- Advice for the code (for `rl_framework.py`), you may try `**kwargs` syntax sugar in Python to pass the configuration as parameters!

Q2

- Run the PS code from Github on your computers. Understand how to set up the code to run the Grid World environment and solve it with PS. Try three implementations: flexible, basic, sparse.
- Send me in Direct Message the outputs you obtain in running a single PS agent in the Grid World environment.
  - Plot reward vs. trials curve.
  - Plot number of steps vs. trials curve.

A2

Test How to Run the Code and Solve GridWorld with the `base` agent

- Simply run `run.py` with a proper configuration and the results (performance for each eta average over agents, or something like this) will be stored in a `results` folder
  - `env_name`: which environment to use
  - `env_config`: the configuration of the environment
  - `multiple_agents` determines how the `Env` and `Agent`s interact (1-1 or 1-many)
  - `num_agents`: how many agents to take average on (in the `multiple_agents = False` scene), or how many agents are in the environment (in the `multiple_agents = True` scene)
  - `agent_name`: choose the type of agent you want to use
  - `max_num_trials`: how many trials do you want an agent to have
  - `max_steps_per_trial`: limit the number of steps in each trial (to avoid a trial that is too long or even an infinite loop)
- I improve the code with some additional features
  - `learning curves for each eta`
    - `run.py` and `rl_framework.py`
  - pass `wall`s to `GridWorld` environment constructor with `env_config`
    - `run.py` and `env_grid_world.py`
  - apply `tqdm` to the loop to visualize the progress of the tasks

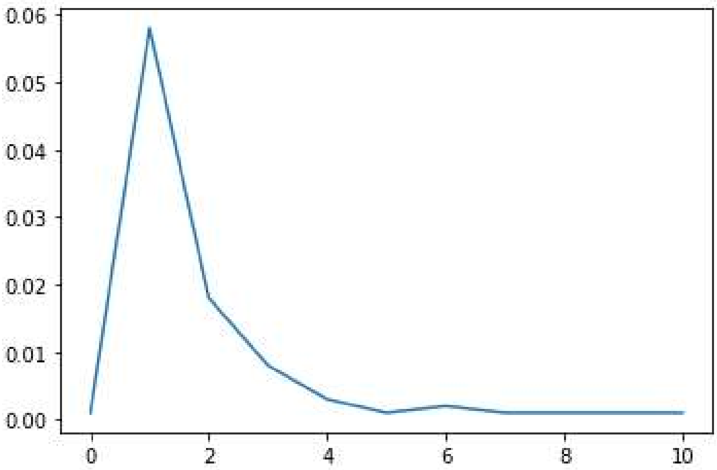
- ```
  PS C:\Users\Maxwell Lin\Documents\projectivesimulation-master> python .\run_grover.py
    0%|                                                    | 0/11 [00:00<?, ?it/s]
   10%|█████                                              | 1/10 [02:10<19:37, 130.83s/it]
  ```

- There's more that can be done if have time (But I am occupied with other things)
  - save the best agent of the best eta as an object, so that we can experiment with it

```
In [17]: performance = np.loadtxt('./results'+'/param_performance', delimiter=',')
```

```
In [18]: plt.plot(performance)
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x1b02b624610>]
```

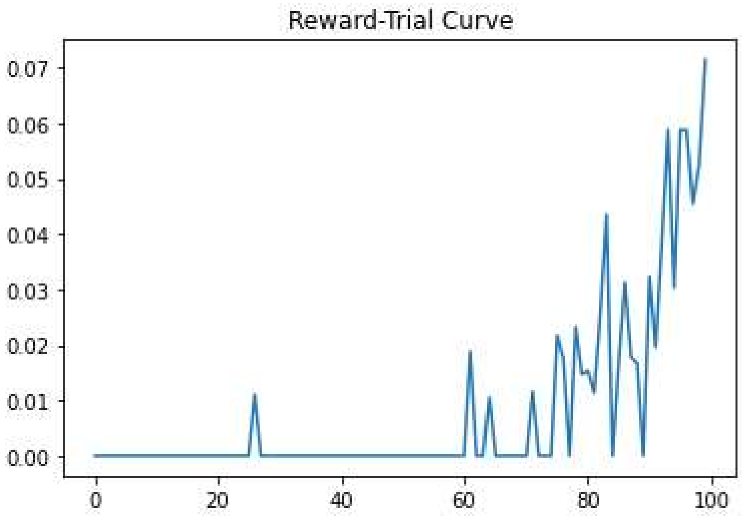


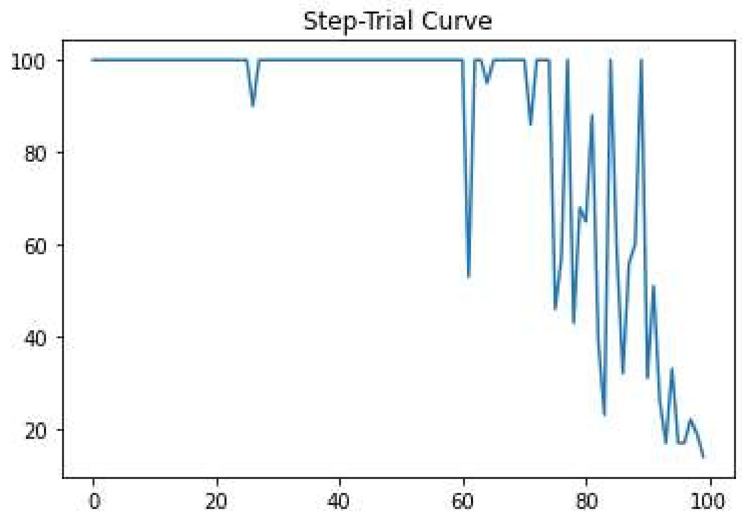
- By running the code for grid world environment (with my walls-passing code) and reading out the `average_param_performance` nd-array, we can see that the optimal η according to this experiment is $\in [0.1, 02]$
- The `sparse` (sparse matrix), `flexible` (dictionary of states) agents differ from `basic` agent only in the way they store states, and the states they stored should be identical, so I think there is no reason to try the same computational consuming process to find ηs from the same distribution again. However, I did test the correctness of the codes with a smaller setting. I will omit this part for the quantum circuit experiment.

Results for the GridWorld problem with a single PS agent

- I modify the `simple_interaction.py` and plot the resulting data
  - the code can be found in the `Single Agent in Grid World.ipynb` in my GitHub repo
- The configuration is as follow
  - agent : `basic` PS agent with softmax and $\eta = 0.1$(delayed reward), w/o reflections or gamma damping (forgetting)
```

- environment: the figure provided in the slide





Q3

Program a toy task environment. And solve it with PS (try basic, flexible, and sparse).

- Actions Implement quantum gates on a register of two qubits.
  - action #1: Hadamard gate on qubit 1
  - action #2: Hadamard gate on qubit 2
  - action #3: X gate on qubit 1
  - action #4: X gate on qubit 2
  - action #5: Z gate on qubit 1
  - action #6: Z gate on qubit 2
  - action #7: CNOT gate on qubits 1-2
- Environmental states
  - Current sequence of actions within current trial.
- Reward

    - Reward of +1 is given for arriving to an environmental state that corresponds to a quantum circuit implementing Grover diffusion operator.
- In this task provide:
    - reward vs. trials curve
    - quantum circuit length vs. trials curve
    - draw circuits to which agent converges

A3

Implementation of the environment

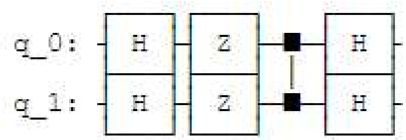
There are two ways I come up with that can implement the environment

- direct calculation with numpy
    - `np.kron(A, B)` = $A \otimes B$
    - `A@B` = AB
- “do not redo” with the existence of qiskit package
    - [Unitary Simulator provided in Qiskit Aer (https://qiskit.org/documentation/stubs/qiskit.providers.aer.UnitarySimulator.html#qiskit.providers.aer.UnitarySimulator)](https://qiskit.org/documentation/stubs/qiskit.providers.aer.UnitarySimulator.html#qiskit.providers.aer.UnitarySimulator)

```
backend_sim = Aer.get_backend('unitary_simulator')
job_sim = execute([qc1, qc2], backend_sim)
result_sim = job_sim.result()
unitary1 = result_sim.get_unitary(qc1)
unitary2 = result_sim.get_unitary(qc2)
np.allclose(unitary1, unitary2)
```

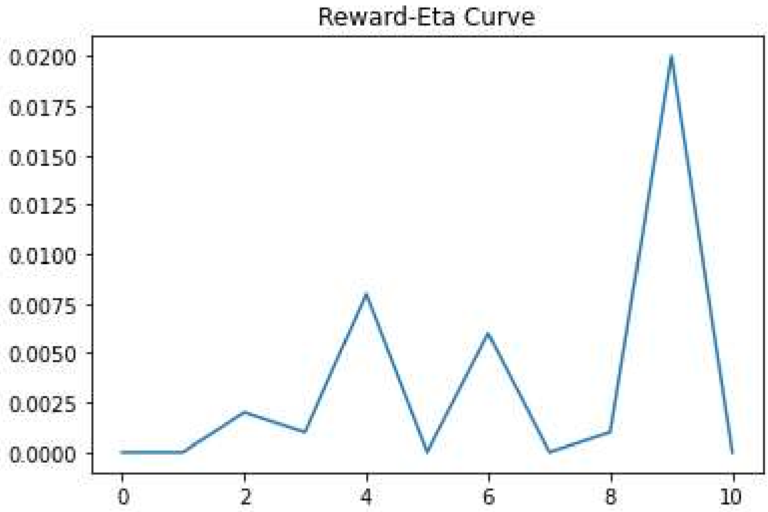
I chose the second one for simplicity, the “do not redo” principle and the convenience to draw a circuit. The code can be found in my Github repo.

The target circuit is represented by a common implementation below.

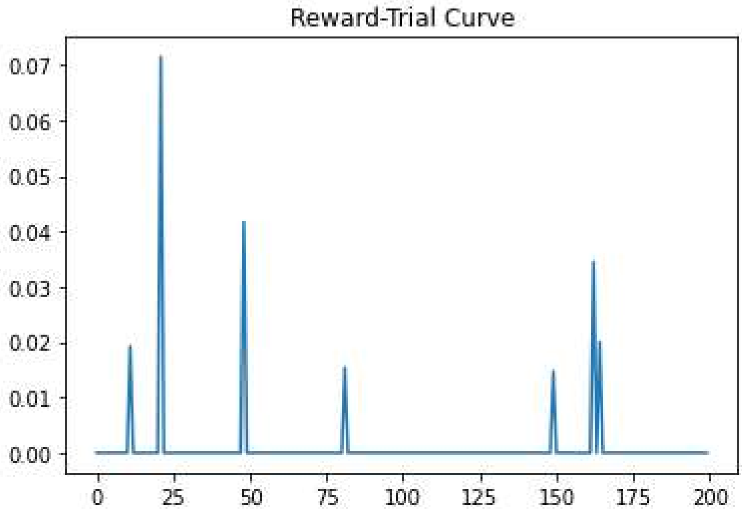


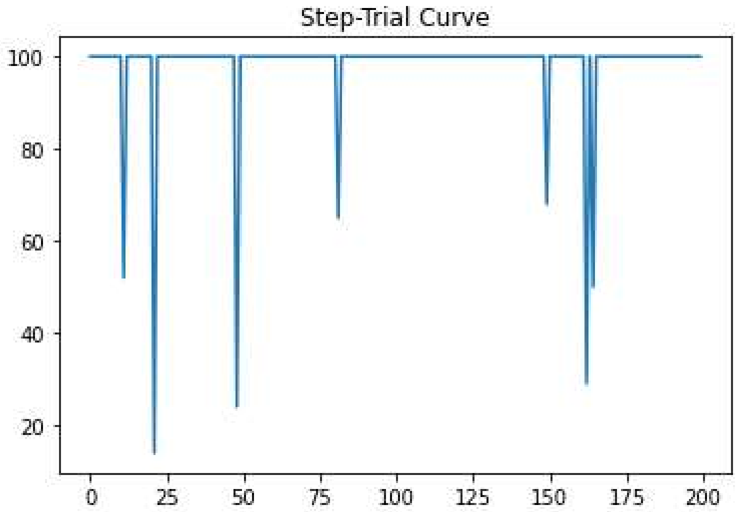
Solving for Optimal η with the `flexible` agent and plot the learning/step-trial curve

To solve for optimal η, I will use the `flexible` agent for the infinite possibilities of quantum circuits without giving a depth limit. Alternatively, one can try using a `sparse` agent with a pre-specified limit of depth.



By running the modified `run.py` and reading out the `average_param_performance` nd-array, we can see that the optimal η according to this experiment is about 0.9.





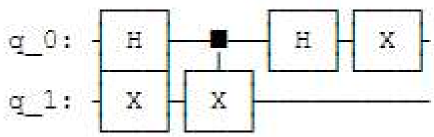
However, I do not observe an obvious “learning”(or say generalization) behavior of the agents since there is no trend of the agents becoming better over trials. I also tried many times and tried with different eta and the results are similar. I think this is because that the state (“current circuit composition”) is too sparse and the agents fail to recognize some “circuit identities” (e.g. $XX = I, ZZ = I, HH = I, HZH = X$, etc.), so unlike in the grid world problem, the agents are unable to tell if a state is equivelant to a state they visited before in terms of unitary.

There are three ways to improve this

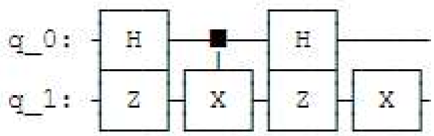
- treat “equivelence class of unitaries” as states directly
  - this violate the spec of the homework
- increase the number of reflections.
  - no direct support for number of reflections in the `flexible` agent
- increase the number of trials.
  - I tried number of trials = 1000, does not help

Draw circuits to which agents converge

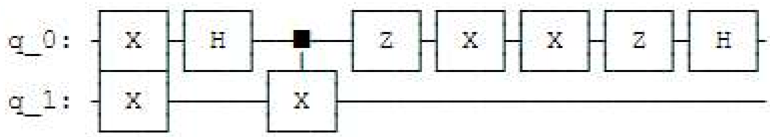
In [38]:
```
saved_circuit.draw()
```
Out[38]:



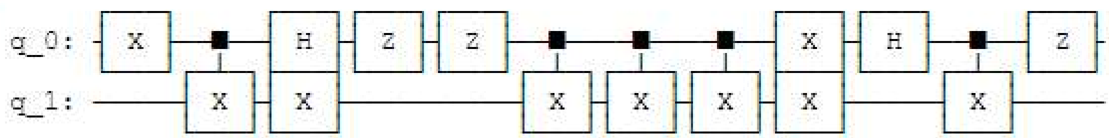
In [16]:
```
saved_circuit.draw()
```
Out[16]:



In [56]:
```
saved_circuit.draw()
```
Out[56]:



In [29]:
```
saved_circuit.draw()
```
Out[29]:



- Despite that learning did not happened, the agent did get some circuits that is equivalent to the Grover’s difussion operator.
- One can verify the circuits are equivalent to the Grover’s Diffusion Operator
  - with the circuit identities: $HH = I, XX = I, ZZ = I, HZH = X$