Reinforcement Learning for Robotic Liquid Handler Planning

Mohsen Ferdosi
School of Computer Science, Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

Yuejun Ge
School of Computer Science, Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

Carl Kingsford
School of Computer Science, Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Robotic liquid handlers play a crucial role in automating laboratory tasks such as sample preparation, high-throughput screening, and assay development. Manually designing protocols takes significant effort, and can result in inefficient protocols and involve human error. We investigates the application of reinforcement learning to automate the protocol design process resulting in reduced human labor and further automation in liquid handling. We develop a reinforcement learning agent that can automatically output the step-by-step protocol based on the initial state of the deck, reagent types and volumes, and the desired state of the reagents after the protocol is finished. We show that finding the optimal protocol for solving a liquid handler instance is NP-complete, and we present a reinforcement learning algorithm that can solve the planning problem practically for cases with a deck of up to 20×20 wells and four different types of reagents. We design and implement an actor-critic approach, and we train our agent using the Impala algorithm. Our findings demonstrate that reinforcement learning can be used to automatically program liquid handler robotic arms, enabling more precise and efficient planning for the liquid handler and laboratory automation.

2012 ACM Subject Classification Computing methodologies → Sequential decision making

Keywords and phrases Liquid Handler, Reinforcement Learning, Planning

Digital Object Identifier 10.4230/LIPIcs.WABI.2023.23

Supplementary Material *Software (Source Code)*: `https://github.com/Kingsford-Group/rlforlqh`
archived at `swh:1:dir:d8aa207d3ff7e53d35ada7c8ffcaab4918d851ef`

Funding This work was supported in part by the US National Science Foundation [DBI-1937540, III-2232121], the US National Institutes of Health [R01HG012470], the Center for Machine Learning and Health at Carnegie Mellon University through a fellowship to M.F. and by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program.

Acknowledgements We thank Guillaume Marçais for helpful comments on the manuscript and Haotian Teng and Sam Powers for valuable discussions.

Conflicts of Interest C.K. is a co-founder of Ocean Genomics, Inc.

1 Introduction

A robotic liquid handler is a laboratory instrument that is used to dispense precise amounts of liquids into a variety of containers, such as micro-plates, test tubes, and vials. It automates liquid handling tasks, enabling scientists to process large numbers of samples quickly and accurately. Robotic liquid handlers consist of several components, including a robotic arm that moves the liquid handling tool (e.g. pipette) to the appropriate location, a liquid

dispensing system (e.g. syringe or positive displacement pump) that accurately dispenses the liquid, and software that controls the entire process. These instruments are commonly used in a range of applications, including drug discovery, genomics, proteomics, and clinical diagnostics, where accuracy, precision, and throughput are essential for obtaining reliable and reproducible results [7]. Even though there is a wide range of laboratory equipment and chemicals, diverse preferences in laboratory configurations, as well as variations in the selection of target organisms and research inquiries, there is a fundamental problem common to all protocols in a way that can be generalized to the planning task for liquid handling [2].

In laboratory practices that involve large-scale liquid handling, establishing reliable and consistent protocols is crucial for generating credible data. Robotic liquid handling techniques have played a significant role in accomplishing this objective in medical laboratories. The success of high-throughput PCR screening for SARS-CoV-2, capable of processing millions of samples weekly, is a testament to this [14]. Robotic liquid handling is used for many applications such as protein folding analyses [1], high-throughput processing for selected reaction monitoring assays [17], and in many clinical diagnostics [9], including microbiome screening [10].

There has been to date no analysis of the computational complexity of the planning problem for the liquid handlers. We therefore propose a formal view of the problem with theoretical complexity analysis that shows the problem is NP-complete, as one would expect. This motivates and justifies the development of heuristics for the problem. In order to obtain reliable and reproducible results from experiments while minimizing the cost of human labor in process, there is a clear need for a robust and scalable planning algorithm for liquid handling robots. Roboliq [15] proposed a software pipeline to transform high-level protocols designed by researchers to low-level robot instructions. SynBiopython [16] proposed an open-source Python package aiming to standardize some of the tools used in automated laboratories. ESCALATE [11] proposed a framework that enables writing machine-readable protocols with hybrid human-robot operations that enable the recording of data and optimization of experiments. Aquarium [13] offers visual programming in a way that protocols are represented graphically as blocks that can be assembled to build executable protocols.

All these methods still rely on researchers to design and conduct protocols. Approaches such as ESCALATE and Roboliq require human-intervention for protocol optimization. Automatically designing and finding optimized protocols for liquid handlers will play a significant role in lab automation. We provide the first practical way of automatically finding low-cost liquid handler protocols, significantly reducing the need for human intervention.

Reinforcement learning is a machine learning paradigm that involves learning a policy to make decisions through interaction with an environment. Reinforcement learning has been used for solving combinatorial optimization problems in order to replace using hand-crafted heuristics [8]. By integrating reinforcement learning algorithms into the planning of liquid handler robots, we aim to automate the protocol design of the liquid handler to address common challenges and limitations associated with traditional robotic systems in liquid handling, such as inefficient protocols, human error in protocol design or the high computational cost of planning of the robot. Our results show that it is possible to fully automate the protocol design for the liquid handler in many practical cases. We train reinforcement learning models based on the actor-critic paradigm and using the Impala (Importance Weighted Actor-Learner Architecture) framework [4]. Our trained model solves the vast majority of inputs for cases with a deck of up to 20×20 wells and four different types of reagents. Traditional planning algorithms are computationally intractable for many of these instances due to the super-exponential nature of the problem in terms of the number of possible actions in each step. We also show that the trained model is robust to different problem settings.

2 Formal Problem Statement

A liquid handler consists of a set of small containers (wells) and a robotic arm, called the head, that can transfer liquids between the wells using a set of specified actions. The liquid handler starts with an *initial state* which gives the configuration of the containers at the beginning of the experiment. The aim is to convert the initial state to the goal state using the set of specified actions. Here, we add some idealizing assumptions to make the formulation simpler without losing the generality of the problem:

- The state is defined by a 2D field of wells containing mixtures of reagents. Our idealized handler is designed with a single large "deck", whereas actual handlers may have multiple decks. Any complexities arising from multiple plates in real-world situations can be incorporated into the customized cost function that calculates the protocol cost.
- The head is a 2D array of tips. It can be positioned anywhere within the 2D well field, and its location is represented as an Δ offset from the top-left corner of the field. For the most part of this paper, we assume we are working with head size of 1×1.

▶ **Definition 1** (Reagents). *There is a finite set of* reagents *that can be mixed in each well. We denote them by* $R = \{r_1, r_2, ..., r_K\}$. *The mixture in each well can be represented by a* $1 \times K$ *vector that shows how much of each reagent is in that well.*

▶ **Definition 2** (Deck). *The deck is* 2D *field of wells containing mixtures of reagents. The current state of the reagents in each of the wells, along with the position and contents of the head, gives the current* state *of the liquid handler.*

▶ **Definition 3** (Head). *The head is a smaller* 2D *field of wells positioned anywhere in the deck. The position of the head allows for the two actions of aspiration or dispensation on the wells that are covered by the head. The head can change location within the deck and this movement has a cost.*

▶ **Definition 4** (Aspiration). *Aspiration is the act of transferring some amount of the mixture from the wells covered by the the head to the tips on the head. This action comes with two costs. The cost of aspiration is a constant* c_a *plus a movement cost that is the cost to move the head to the location for the aspiration from its previous location.*

▶ **Definition 5** (Dispensation). *Dispensation is the act of transferring some amount of the mixture from the tips in the head to the wells covered by the the head. This action comes with two costs. The cost of dispensation is a constant* c_d *plus a movement cost that is the cost to move the head to the location for the dispensation from its previous location.*

▶ **Definition 6** (Protocol). *A protocol is the sequence of actions of aspiration, dispensation, and moving the head's location in order to achieve a goal state starting from an initial state. The cost of a protocol is the sum of the cost for each action in the protocol.*

▶ **Problem 7** (Optimal Liquid Handling). *Given an initial state* $\mathcal{I}$ *of the deck (with an empty head), and the goal state* $\mathcal{G}$ *and the set of parameters* c_a *and* c_d *that determines the cost of each action as above, and a choice of polynomial-time computable distance metric that determines the cost to move the head between locations, give a protocol with the minimum cost that will convert* $\mathcal{I}$ *into the goal state* $\mathcal{G}$.

We show that this is an NP-complete problem even in the most restricted case of the unit-sized head and only one type of reagent. We then develop a practical approach to find low-cost protocols using reinforcement learning.

2.1 The Optimal Liquid Handling problem is NP-complete

We show that the Optimal Liquid Handling problem is NP-complete.

▶ **Theorem 8.** *The Optimal Liquid Handling decision problem that asks whether there is a protocol of cost less than or equal to C is NP-complete.*

Proof. Verification of a protocol's cost can be computed easily in polynomial time. We can follow the sequence of actions and calculate the total cost to check if it is less than or equal to C. Hence, the problem is in NP.

We reduce the Traveling Salesman Problem (TSP) to the special case of Optimal Liquid Handling with the head size of 1×1. For this, we need to use the metric TSP where all the cities are points in $\mathbb{R}^2$. The NP-completeness proof in [5] for the Euclidean TSP immediately implies the NP-completeness of a TSP where all the cities are points in $\mathbb{R}^2$ and the distances are defined in a way that the triangle inequality holds [3]. Figure 1 shows the sequence of reductions to show that the Optimal Liquid Handling is NP-Complete.

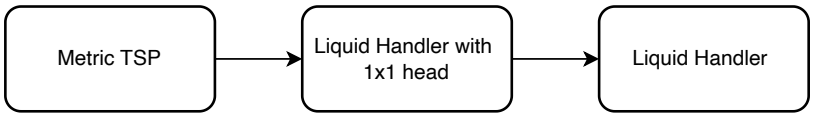

Figure 1 The order of the reductions from the Euclidean TSP to the Optimal Liquid Handling.

TSP takes a list of n cities $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ in $\mathbb{R}^2$ and a budget C as input, and asks whether there is a tour denoted G_{tsp} that visits every city exactly once with total cost at most C. The reduction from TSP to Optimal Liquid Handling is as follows:

Construct an Optimal Liquid Handling instance with a budget C, and the initial state of n units of reagent on the location of c_1 on the deck and the goal configuration of one unit of reagent in each of the city locations on the deck. The cost of aspiration c_a and dispensing c_d are defined to be zero, and the cost of the head movement between the two wells is defined with the same metric as the distance in the TSP. This way the head needs to visit every well that represents the city location on the deck at least once to have the reagent in that location.

▶ **Lemma 9.** *For any sequence of moves of the head of the robotic arm that results in the tour G_0 with cost at most C, there is a tour of G_{tsp} with cost $C^* \leq C$ in TSP where it visits the same cities in the same order.*

Proof. Not every move in the robotic arm's tour G_0 must end up in the cities. The arm can move freely in the grid before visiting the next city. Let's say the moves are segmented by the cities and $C = \{c_1, c_2, ..., c_n\}$ are the cities that have been visited in G_0 in the order of their visit. We can replace any sequence of moves that happened between cities c_i and c_{i+1} with just a direct move from c_i to c_{i+1} and the cost would not be increased due to triangle inequality (which holds for any norm). Hence the new G_{tsp} would have a cost of $C^* \leq C$. ◀

Suppose there exists a protocol that visits the wells in an order that results in a tour G_0 with cost at most C in Optimal Liquid Handling, then there exists a tour of G_{tsp} with a cost of C or less in TSP using Lemma 9. If there is a tour of G_{tsp} with at most cost of C in TSP the robotic arm can follow the same path therefore there is a tour of G_0 with cost at most C in the Optimal Liquid Handling problem. This means the Optimal Liquid Handling can be used to solve the metric TSP problem which proves that Optimal Liquid Handling is NP-complete. ◀

The NP completeness of the Optimal Liquid Handling problem motivates us to use machine learning for a practical solution instead of looking for a polynomial-time algorithm.

3 Methods

3.1 Using Reinforcement Learning to Solve Optimal Liquid Handling

Reinforcement learning (RL) has emerged as a powerful technique for solving complex tasks in various domains. Actor-critic [6] is a type of reinforcement learning algorithm that uses two estimators: an actor and a critic. The actor is responsible for learning a policy, which is a function that maps states to actions. The critic is responsible for learning a value function, which estimates the expected reward from a given state and action. The interaction between the actor and the critic enables the agent to continually refine its policy based on the feedback provided by the critic, leading to improved decision-making over time.

Impala is a distributed reinforcement learning framework that integrates the actor-critic framework with a distributed architecture [4]. Impala is designed to improve the scalability and efficiency of traditional actor-critic algorithms by training multiple agents. It separates the learning process into multiple actors and a central learner [4]. Actors interact with their own instances of the environment, collecting trajectories (sequences of state, action, and reward). Multiple agents simultaneously interact with the environment and collect experience, which is used to update a shared value function and policy. Actors asynchronously send their collected trajectories to the central learner, which processes the data and updates the policy and value function. The central learner periodically sends the updated policy and value function back to the actors, ensuring that they stay relatively up-to-date with the latest learning progress. This enables efficient use of computational resources and allows the algorithm to scale to a large number of actors. This approach allows for faster and more efficient learning compared to traditional reinforcement learning algorithms that rely on a single agent [4].

We tackle the Optimal Liquid Handler problem with the head size of 1×1 using reinforcement learning. We use Impala [4] for the training and CORA [12] for the benchmarking and evaluation. CORA (short for Continual Reinforcement Learning Agents) is a package that provides benchmarking, baselines, and metrics for continual reinforcement learning tasks.

Environment. The environment is the setting in which the RL agent operates. It can be modeled as a state space, where each state corresponds to a particular configuration of the environment, and the agent can take actions to transition from one state to another. The deck, the head, and the configuration of reagents in the deck and the head form the environment in the liquid handler application. Let $N \times M$ be the size of the deck, and K be the number of different types of possible reagents.

Agent. The agent is the entity that interacts with the environment. It learns a policy which maps states to actions. The agent receives information about the environment and takes actions according to its policy and based on the input. Here, the agent outputs steps in a protocol for the liquid handling problem.

State space. The set of all possible states of the environment is described by $\{\mathcal{D}, \mathcal{G}, \mathcal{H}\}$. We define the state space with a pair of $N \times M \times K$ tensors $\mathcal{D}, \mathcal{G}$ where both $\mathcal{D}, \mathcal{G} \in \mathbb{N}^{N \times M \times K}$ and vector $\mathcal{H} \in \mathbb{N}^{K}$. $\mathcal{D}$ describes the current state of the deck where $\mathcal{D}[i][j]$ is a vector of size K that specifies how many units of each reagent are currently present in the well located

at (i, j). Values in $\mathcal{D}$ are initialized by the initial state $\mathcal{I}$ (as the input to the agent) and they can change according to each action defined below. Similarly, $\mathcal{G}$ describes the goal state which shows how many units of each reagent is supposed to be in every well after the arm performs all the actions in the protocol. Finally, $\mathcal{H}$ describes the amount of each reagents currently in the head. The protocol is considered successfully completed if $\mathcal{D} = \mathcal{G}$ at the end. We define the state space by tensors with integer values. This choice limits us in terms of the mixtures we can produce but it is necessary to keep the problem computationally tractable.

Action space. Actions that the agent can take in a given state. We design our agent with an internal state machine that defines the possible actions. There are two states in our setting. The agent is either in aspiration mode or dispensation mode depending on whether the head is empty. These two states alternate as described in Figure 2. For simplicity, a movement of the head is combined with the aspiration or dispensation actions as multiple actions in the same location can be combined into one single action.

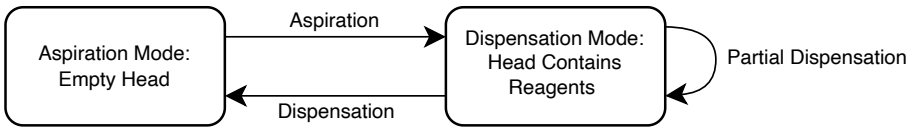

Figure 2 Two sets of actions and their corresponding internal states. Depending on whether the agent's head is empty or not, there are two sets of possible actions: aspiration and dispensation. To avoid potential cross-contamination, aspiration is only allowed when the head is empty because there might be residual liquid in the tip caused by previous aspiration that can mix with the reagent in well.

There are two actions that the agent can take based on the internal states it is currently in. When the agent is in aspiration mode, the head is empty and the agent can perform an "aspiration" action.

Aspiration is defined with the tuple (i, j, P) where P is a vector of size K, $P \in \mathbb{N}^K$, and

$$\exists b \quad 0 < b \leq 1 \quad \text{where} \quad \frac{P[k]}{\mathcal{D}[i][j][k]} = b \quad \text{for every } k \in [1, K]. \tag{1}$$

This action transfers the amount of reagents defined by P from $\mathcal{D}[i][j]$ to the head $\mathcal{H}$. The condition (1) guarantees that the ratios are kept the same for the amount that is being aspirated and the mix in the well. Aspiration is formally defined as:

$$\begin{aligned} \mathcal{D}[i][j] &= \mathcal{D}[i][j] - P \\ \mathcal{H} &= P. \end{aligned} \tag{2}$$

The cost of aspiration is c_a. This cost is independent of P, the volume aspirated, and the location. While this is not completely physically accurate, it is a reasonable model when we want to minimize the number of actions. After each aspiration the internal state of the agent changes to dispensation mode since the head is not empty anymore.

Similarly, when the agent is in dispensation mode, the agent can perform a "dispense" action since there is some liquid in the head. Dispense is defined with the tuple (i, j, P) where P is a vector, $P \in \mathbb{N}^K$, and

$$\exists b \quad 0 < b \leq 1 \quad \text{where} \quad \frac{P[k]}{\mathcal{H}[k]} = b \quad \text{for every } k \in [1, K]. \tag{3}$$

This action transfers the amount of the reagents defined by P from the head $\mathcal{H}$ to $\mathcal{D}[i][j]$. The condition (3) guarantees that the ratios are kept the same for the amount that is being dispensed into the well and the mix in the head. Dispense is formally defined as:

$$\begin{aligned}\mathcal{D}[i][j] &= \mathcal{D}[i][j] + P \\ \mathcal{H} &= \mathcal{H} - P.\end{aligned} \tag{4}$$

We call this partial dispense in the case where $\mathcal{D}[i][j] \neq P$. Partial dispense causes the internal state to stay on dispensation mode since the head is not empty yet. The cost of a dispense is c_d. This is again independent of the volume dispersed.

Each action also comes with a distance cost that models the movement of the head from the well of the previous action to the well of the next action. The cost of the n-th action $(i, j, P)_n$ depends on the previous $(i, j, P)_{n-1}$ action and is defined as

$$\begin{aligned}c_n &= \text{Manhattan Distance}((i, j)_n, (i, j)_{n-1}) \\ &= |i_n - i_{n-1}| + |j_n - j_{n-1}|,\end{aligned} \tag{5}$$

and cost of a protocol is defined as the sum of the cost for all actions and movements. We use Manhattan distance here as our choice of distance, although other metrics are also reasonable.

Reward. The reward is a scalar feedback signal that the agent receives from the environment after taking an action. The goal of the agent is to learn a policy that maximizes the cumulative reward over time. The agent's aim is to reach the goal state while minimizing the distance cost of the protocol. In this task, the agent receives $+1$ reward for dispersing one unit of reagent in a well that is missing at least one unit of that reagent to reach its goal defined by the goal state $\mathcal{G}$. In order to facilitate the exploration and motivate the agent to perform moves, we also add $+1$ reward for aspirating one unit of a reagent that is currently in a well that does not have that reagent in its goal (or the amount currently in the well exceeds the amount described in the goal), meaning that this unit has to be moved in order to achieve the goal state. This way moving one unit of reagent from a wrong well to a correct well results in total of $+2$ reward. There is also -1 reward for aspirating one unit of reagent that is currently in the correct well and another -1 reward for dispersing one unit of reagent in the wrong well. This means moving one unit from a correct well to another correct well or from a wrong well to another wrong well results in 0 reward, and moving a unit from a correct well into a wrong well results in -2 reward. We also weight the distance cost by a parameter α that is the relative weighting of the distance cost to the reward. More formally, we calculate the reward for each action as described below.

In aspiration mode, the reward for the move (i, j, P) is calculated by

$$\begin{aligned}C &= \min(\mathcal{G}[i][j], \mathcal{D}[i][j]) \\ C' &= \min(\mathcal{G}[i][j], \mathcal{D}[i][j] - P) \\ W &= C - C' \\ R &= P - W \\ r &= |R| - |W| - \alpha c_n\end{aligned} \tag{6}$$

where the function "min" between two vectors is defined as a vector of the element-wise minimum. $C \in \mathbb{N}^K$ shows the amount of reagents that are currently in position (i, j) and are part of the goal state (completed units) before the move, and $C' \in \mathbb{N}^K$ shows the amount of reagents that will be in position (i, j) and are part of the goal state after the move. $W \in \mathbb{N}^K$

is the vector of reagents that are being aspirated from a goal well (wrong moves), and $R \in \mathbb{N}^K$ is the vector of reagents that are being aspirated from a well that does not need that type in its goal (right moves). Hence, the reward r is calculated by $+1 \times |R| - 1 \times |W|$ and subtracted by a factor of the distance cost of that move $-\alpha c_n$. We can always choose a small enough α based on the size of the deck so that finding a protocol that completes the task by reaching the goal state has a higher priority than optimizing the distance cost.

Similarly in the dispensation mode, the reward for the move (i, j, P) is calculated by

$$\begin{aligned} L &= \max(\mathcal{G}[i][j] - \mathcal{D}[i][j], 0) \\ R &= \min(P, L) \\ W &= P - R \\ r &= |R| - |W| - \alpha c_n \end{aligned} \tag{7}$$

where the function "max" between a vector and a number is defined as the element-wise maximum of the values in the vector and the number. $L \in \mathbb{N}^K$ shows the amount of reagents that are needed in position (i, j) to complete the goal for well in (i, j) (units left) before the move. Similarly to the aspiration reward, R is the vector of reagents that are being dispersed to a goal well (right moves), and W is the vector reagents that are being dispersed to a non-goal well (wrong moves). The reward r is again calculated by $+1 \times |R| - 1 \times |W| - \alpha c_n$.

Policy. The policy is the mapping between states and actions that the agent uses to make decisions. The goal of the agent is to learn an optimal policy that maximizes the cumulative reward over time.

3.2 The Network Architecture

We use Impala for our training framework. Impala is a distributed reinforcement learning algorithm designed to tackle large-scale RL problems efficiently. It achieves this by employing a distributed architecture, which enables parallel data collection and learning. The actors are responsible for exploring the environment and generating trajectories of state, action, reward, and next-state tuples by following their respective behavior policies. The central learner processes the collected data to update both the policy (actor) and the value function (critic). Each actor in the Impala architecture has an actor network, which is responsible for selecting actions based on the current state of the environment. We design the actor network as a three-layered convolutional network described below.

The designed network needs to take an encoding of the current state $\{\mathcal{D}, \mathcal{G}, \mathcal{H}\}$ as the input and must output a probability distribution over all the possible actions to take at this step. We define observation $\mathcal{O} = \{\mathcal{D}, \mathcal{G}, \mathcal{H}^*\}$ as the encoding, where $\mathcal{D}, \mathcal{G}$ are previously defined as the current state and the goal state. $\mathcal{H}^*$ is a tiling of the vector $\mathcal{H}$ where $\mathcal{H}$ is repeated $N \times M$ times (to count for each well). This way, all three of $\{\mathcal{D}, \mathcal{G}, \mathcal{H}^*\}$ are $N \times M \times K$ tensors. We do this tiling because we want the associations between the current reagents in the head and the goal definition for each well to be highlighted in the convolutional layer (e.g. if the amount in the head matches or mismatches with the goal for each well). We use convolutional layers with kernel size of 1×1 to perform a channel-wise feature mixing. The 1×1 convolution is used to combine features across channels with information on $\mathcal{H}^*$ and the channels with information on $\mathcal{G}$. This is aimed to help the network learn which wells still need some reagents to reach the goal state.

The output of each state is the policy distribution for every possible action. We define the capacity of the tip on the 1×1 head as D. A possible action a is defined as $a = (x, y, p)$ which states "p amount of the the mix and the position (x, y) on the grid". The type of the

action (i.e. aspiration or dispensation) is determined by the internal state machine. If the head is not empty, the action $a = (x, y, p)$ is a dispense, and if it is empty, the action is an aspiration. Since we are using $\mathcal{O} = \{\mathcal{D}, \mathcal{G}, \mathcal{H}^*\}$ as the encoding, we have $3 \times K \times N \times M$ input values, which is equivalent of $(3 \times K)$ channels for images of size $N \times M$, where each channel describes the amount of one reagent in either the current state, the goal state, or the tiled head. Similarly, we have $D \times N \times M$ output values that describe the policy distribution; these are equivalent to D channels for images of size $N \times M$. We use the three layered convolutional network described in Figure 3 for the actor network, and we use a flattening layer at the end to get a one-dimensional vector that describes the the policy distribution.

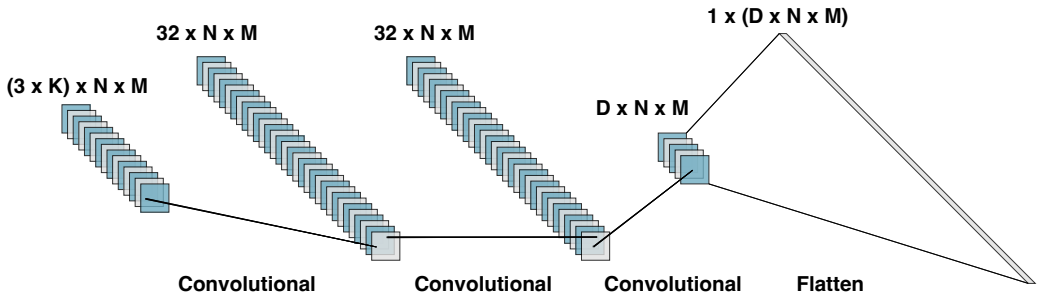

Figure 3 The three-layered convolutional network used for the actor network. The actor learns the policy, which is described by the probability distribution over the possible actions, based on the current state $\{\mathcal{D}, \mathcal{G}, \mathcal{H}\}$ as its input.

The central learner in Impala also has a critic network that estimates the value function, which represents the expected cumulative reward from a given state, following the current policy. For our task, we design the critic network as a fully connected layer that gets the observation $\{\mathcal{D}, \mathcal{G}, \mathcal{H}\}$ as the input and produces a number as its output that describes the expected reward for the given state.

4 Results

We first show some examples to showcase some of the challenges that the agent needs to overcome to find an optimal protocol. We then present experiments benchmarking the proposed reinforcement learning approach compared with several baseline approaches.

4.1 Challenges

The first priority for the RL agent is to reach the goal state. The agent also needs to minimize the distance cost of the protocol. Figure 4 shows two possible protocols for a given state. The agent needs to choose the one with lower distance cost.

One thing that the agent needs to learn is the volume of each action. Always using the whole volume available for aspiration and dispensation would not reach the goal in some cases. Figure 5 shows why it is important for the agent learn to use moves with $b < 1$.

Another challenge for the agent is to learn to mix or not mix reagents when necessary. This is a unique aspect of the planning for liquid handler robotic arm since putting liquid on top of each other results in mixing them and producing a new reagent. This move is irreversible by the liquid handler. Figure 6 shows how mixing when not necessary can result in a state where reaching the goal is not accessible anymore. The agent needs to learn these cases in order to be able to find the optimal protocol.

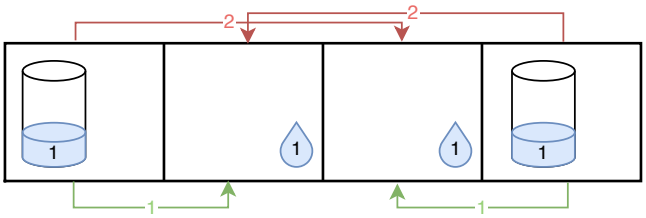

Figure 4 Here we have 1×4 deck. The two side wells each have one unit of the blue reagent in the initial state $\mathcal{D}$ as shown by the two cylinders. The two middle wells each have one unit of the blue reagent in the goal state $\mathcal{G}$ as shown by the two drops on the bottom right of each well. If we do the moves based on the red arrows on the top the distance cost will be equal to 4, but if we do the moves based on the green arrows on the bottom, the distance cost will be equal to 2. We expect the agent to choose the green arrows for the moves.

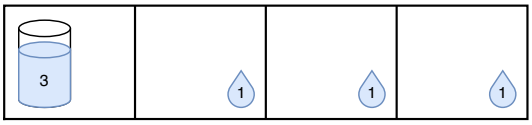

Figure 5 In this example the left wells has three units of the blue reagent in the current state $\mathcal{D}$ and need to distribute it into three well with one unit in each. It would be impossible for the agent to reach the goal state $\mathcal{G}$ without using moves with $b < 1$.

The agent cannot simply collect rewards greedily by moving each reagent to a goal well. Figure 7 shows how a greedy algorithm would fail by falling into a state that makes the goal unreachable because of the irreversibility of mixing.

4.2 Experiments

We show experimental results from the reinforcement learning agent to show the potential of this approach on solving the automated protocol design for robotic liquid handlers. In the first experiment, we show that by training on various settings we are able to receive almost perfect results in terms of receiving the highest possible reward for each instance. In the second experiment, we show that the trained agent is robust to other settings, and in the third experiment we show the comparison of reinforcement learning method and traditional planning and a greedy algorithm in terms of finding the optimal protocols. We used the following hyperparameters for the reinforcement learning: number of actors = 16, batch size = 16, discount factor = 0.9, and learning rate = 10^{-4}. All the experiment are performed on a machine with Apple M2 Pro chip with a 10-core CPU and a 16-core GPU and 32GB of unified memory.

4.2.1 Model Performance in Terms of Completing the Task

In this experiment, we train the model on randomly generated initial and goal states. Every unit of reagent is placed on the locations of the grid uniformly and independently. We make sure that the input is synthesized in a way that the maximum possible reward is fully reachable, meaning if there are S units of reagents in the goal, there is a protocol that reaches $2S$ reward. To create these instances, we separate the wells that initially have the reagents from the wells that have the reagents in the goal state. This guarantees that no reagent is already where it should be in the goal state making the reward for it unreachable. The other condition is that the reagents are placed in a way that a series of aspiration and dispense steps will reach the goal state from the initial state. To guarantee this, we start by creating

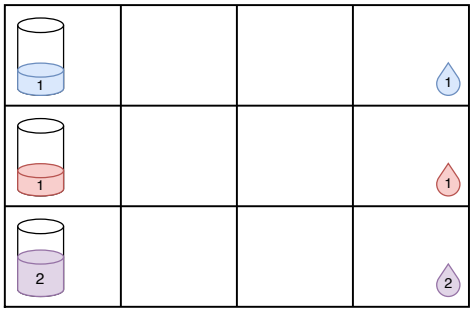

Figure 6 In this example the left column shows the reagents in the initial state. We have one unit of the blue reagent in the top, one unit of the red reagent in middle, and a mix of one unit of blue and one unit of red reagents at the bottom (purple reagent). We have the same formation for the goal state in the column at the right. The agent should never mix the blue reagent and the red reagent to make the purple reagent for the bottom right well since that would make it impossible to reach the goal state. Instead, the agent should move every reagent to its corresponding well on the last column.

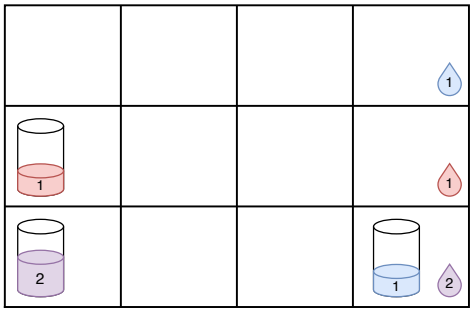

Figure 7 In this example, we have a similar initial state and the same exact goal state as the example in Figure 6. The only difference is that the blue reagent has moved to the bottom right well. A greedy agent could move the red reagent to mix with the blue reagent to receive $+2$ rewards but as we have seen that would make the goal unreachable. Instead the agent should first aspirate the blue reagent from the bottom right corner resulting in -1 reward since one unit of blue reagent is part of the goal state in that well. Only then should it move all three reagents to their corresponding wells.

a random goal state and then randomly move portions of each mixture in the goal state to new wells to create the initial state. This guarantees that there is at least one path to reach the goal from the initial state by doing the reverse of each action that initially produced the test case. We also choose $\alpha = 0$ for this experiment so the distance cost does not affect the rewards.

We train the model on seven different settings with various deck sizes and reagent numbers. Table 1 shows the average performance of each setting for 1000 inputs confirming that the trained model can complete the task for the vast majority of the inputs. For the first four settings, the agent reaches the goal in every single instance and for the three larger settings the model receives around 95% of the rewards on average. That means the model outputs a protocol that gets very close to the goal state, but it is not able to reach the goal in some cases. There is a trade-off between the deck size and the training time in a way that the agent needs more training to work with bigger decks as the state space grows exponentially. In addition, having more reagents increases the complexity of the task, and requires longer protocols to reach the goal. That is the reason we see a drop on the average reward for the bigger instances. Figure 8 shows the average cumulative reward of each run for all the settings over the steps of the training.

Table 1 This table shows the results of training the model on six different settings. The reward is the average reward over 1000 sample and the training time is the amount of time before the model converges. Steps shows the number of training iterations the model was trained on. The training stops if the model converges. For a trained model, it takes less than a second to output the protocol for every setting.

$M \times N$	K	Reagents	Units	Max Reward	Reward	Steps	Training time
2×2	2	$1, 1$	2	4	4 ± 0	1M	3m 59s
5×5	2	$2, 2$	4	8	8 ± 0	10M	21m 46s
10×10	2	$3, 3$	6	12	12 ± 0	16M	1h 31m
12×12	2	$5, 5$	10	20	20 ± 0	16M	1h 53m
15×15	2	$10, 10$	20	40	38.2 ± 0.45	18M	2h 37m
10×10	4	$5, 5, 5, 5$	20	40	39 ± 0.05	18M	1h 41m
20×20	3	$15, 15, 15$	45	90	83.16 ± 5.58	50M	15h

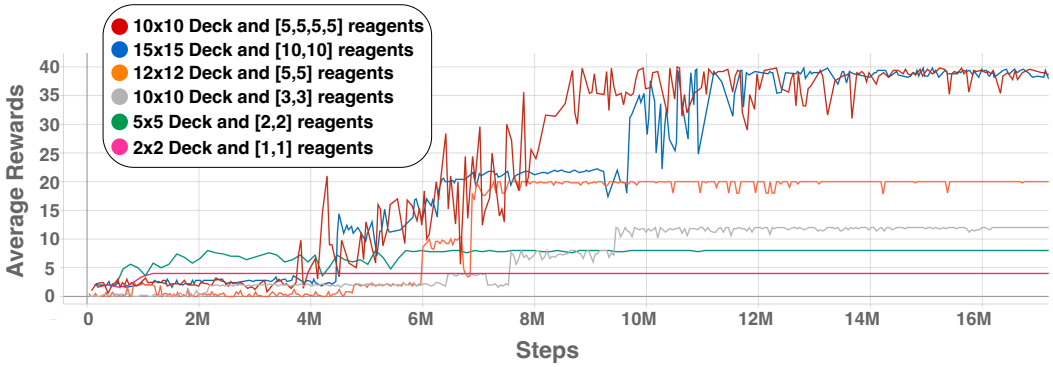

Figure 8 This figure shows the reward over time during the training for different settings. For smaller cases, the model converges to completing all the input tests and receiving full rewards. Every point on the figure is the average of 10 instances and points are generated every 10,000 steps.

4.2.2 Robustness of the Model in Unseen Settings

We show that the trained model can solve settings that are different from what the model has been trained on. To show this, we train a model on the deck size of 10×10, with two types of reagents with total units of $[4, 4]$ (meaning 4 units of reagent r_1 and 4 units of reagent r_2). For evaluation, however, we test the model on the same deck size but with four different settings with total units of $\{[5, 5], [3, 3], [3, 5], [2, 6]\}$ with respective total available rewards of 20, 12, 16, and 16 (taking $\alpha = 0$ to remove the distance costs). Since N, M (deck size) and K (number of different reagents) are the same as the training, the state and action spaces have the same dimensions as before and the network is able to adapt to these new instances. We use the same approach as the previous section to create 1000 random instances. As we can see in Figure 9, the model is able to fully generalize to all of these cases receiving the average reward equal to the max reward possible. Our model is able to do this is because the actor network is designed to be sensitive to every unit of reagent that is not placed correctly and continue working on them until it reaches the goal state making the model generalizable to unseen settings.

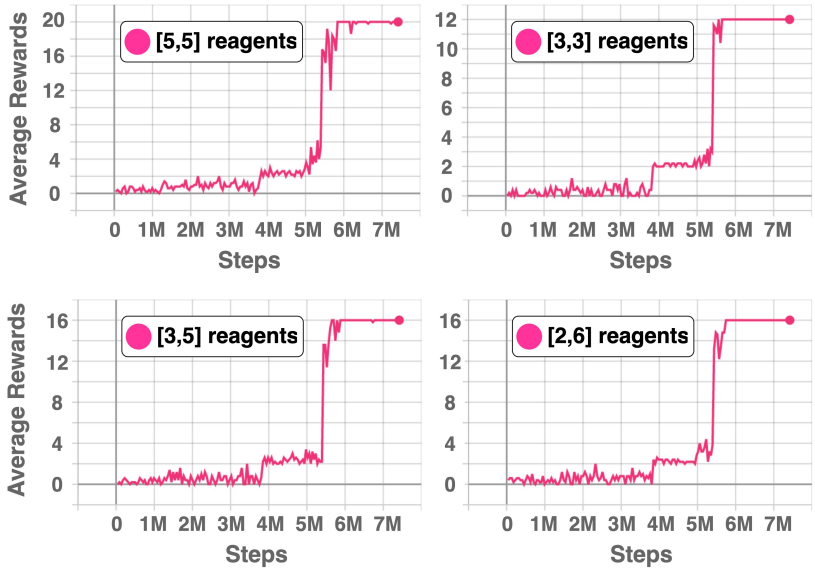

Figure 9 The reward over time for unseen settings during the training. It shows that for all the cases, the model trained on $[4, 4]$ is able to complete all the input tests and receiving full rewards after around 7 millions steps. After each step of the training on $[4, 4]$, we evaluate the model on four new settings that are $[5, 5], [3, 3], [3, 5],$ and $[2, 6]$ with respective total available rewards of 20, 12, 16, and 16 (taking $\alpha = 0$) from left to right.

4.2.3 Optimality of the Protocol in Different Models

Due to the super-exponential nature of the problem, brute force and classic planning algorithm are not tractable for solving this problem. In this section, we compare the proposed reinforcement learning model with a greedy heuristic algorithm and a heuristic-based beam search algorithm as baselines. We compare all the models in terms of success rate which is defined as how often a model outputs a protocol that reaches the goal state, the average distance cost, and the runtime. Table 2 shows the details of each setting.

Greedy Heuristic Algorithm. For this algorithm, we iteratively perform a greedy actions to reach the goal state. We use the same state machine described in Figure 2 but instead of letting the RL agent decide on the tuple $a = (x, y, p)$, we choose them in a greedy fashion. For aspiration, we limit our choices only to the wells that currently have extra reagents compared to what they should have according to the goal state. Similarly, for dispense we only consider the wells that need additional reagents to reach the goal state. We choose the closest x and y to the current position of the head among those choices, and pick p randomly. With this definition of greedy actions, we get closer to the goal state after every step. We stop when we reach the goal state or when there is no more greedy action possible to take.

Heuristic-Based Beam Search. We also use the beam search algorithm, a heuristic search method, as a baseline for this problem. The beam search algorithm uses a breadth-first search (BFS) approach to explore the solution tree. In contrast to the standard BFS that expands all nodes at every level, the beam search uses a heuristic to order the nodes and only keeps a specific set of promising nodes at each stage. The size of this set is called the beam width and is a parameter of the algorithm. At each step, beam search generates all possible successors to the current state by applying all applicable actions. These successors

are then evaluated based on a heuristic function that assigns a score to each state based on its desirability. The states with the highest scores are selected to be the new set of candidate states for the next iteration. This technique significantly cuts down the search space, making it tractable. However, it can also prune the path to the optimal solution, resulting in potentially sub-optimal protocols. We designed a heuristic functions tailored to the optimal liquid handling problem for the beam search. The heuristic function for each action is the sum of the cost and the absolute differences between the volumes of reagents in each well at the current state and the corresponding target volume in the goal state.

As we can see from the Table 2, the proposed reinforcement learning approach outperforms both baselines in terms of the success rate, resulting in a protocol that reaches the goal state in the vast majority of cases. The average distance cost for the proposed reinforcement learning method is also lower than the greedy heuristics method in all the cases. The greedy heuristics method is able to find a valid protocol more often for the cases where the amount of reagents is small compared to the grid size. For those cases, the random positioning of reagents results in a sparse grid which is a simpler task and makes it easier for the greedy heuristic to perform greedy actions one at a time. The beam search algorithm is able to find protocols with smaller distance cost but it has a much smaller success rate. In addition, the beam search has an exponential runtime which makes it not scalable for the larger instances.

Table 2 This table shows the results of comparison between the reinforcement learning (RL), greedy heuristic (GH), and beam search (BS). Each experiment is repeated 1000 times for RL and GH and 100 times for BS. Success rate is the the number each model reached the goal state divided by the total number of experiments. Distance cost is the average of distance cost for those runs that reached the goal. Training time is the amount of time it took for the RL model to train, and query time is the average time it takes each model to output the protocol. We used $\alpha = 0.2$ for the reinforcement learning model and used the beam size of 25 for the beam search on the two larger decks and beam size of 40 for the other instances.

$M \times N$	Reagents	Model	Success Rate(%)	Distance Cost	Training time	Query(s)
4×4	$4, 4$	RL	100	38.496	22m	0.484
		GH	78.7	39.40	–	0.004
		BS	95.0	23.87	–	3.80
6×6	$3, 3$	RL	100	44.471	50m	0.484
		GH	93.2	46.254		0.002
		BS	92	30.74	–	5.37
6×6	$8, 8$	RL	98.9	102.84	1h 21m	0.490
		GH	59.8	113.148	–	0.009
		BS	72	68.44	–	30.10
8×8	$4, 4$	RL	100	76.53	1h 51m	0.486
		GH	93.0	81.455	–	0.003
		BS	91	59.59	–	24.67
8×8	$15, 15$	RL	95.3	264.97	4h 31m	0.494
		GH	40.0	266.652	–	0.014
		BS	43	190.08	–	143.47
10×10	$5, 5, 5, 5$	RL	87.9	232.83	6h 36m	0.474
		GH	70.5	249.99	–	0.008
		BS	34	150.41	–	306.45

5 Conclusion

We demonstrated the potential of reinforcement learning for automating protocol design for robotic liquid handlers. We developed a reinforcement learning agent that automatically generates step-by-step protocols based on the initial state of the deck, reagent types, and volumes. This will result in reduced human labor in the process and further improve the automation of liquid handling tasks. Our proposed reinforcement learning algorithm can effectively solve the planning problem for practical cases involving decks of up to 20×20 wells and four different types of reagents.

Future research could focus on expanding the capabilities of the agent to handle more complex laboratory tasks, larger decks, other distance metrics, and a greater variety of reagent types. Additionally, integrating our approach with existing liquid handling software could streamline the protocol design process and further advance the adoption of this technology in laboratories. Overall, this work represents a promising step towards harnessing the power of reinforcement learning to to further automate the liquid handling process. The implementation of all the models is publicly released at `https://github.com/Kingsford-Group/rlforlqh`.

References

1 Philip An, Dwight Winters, and Kenneth W Walker. Automated high-throughput dense matrix protein folding screen using a liquid handling robot combined with microfluidic capillary electrophoresis. *Protein Expression and Purification*, 120:138–147, 2016.

2 Dominik Buchner, Till-Hendrik Macher, Arne J Beermann, Marie-Thérése Werner, and Florian Leese. Standardized high-throughput biomonitoring using DNA metabarcoding: Strategies for the adoption of automated liquid handlers. *Environmental Science and Ecotechnology*, 8:100122, 2021.

3 Rainer E Burkard, Vladimir G Deineko, René van Dal, Jack AA van der Veen, and Gerhard J Woeginger. Well-solvable special cases of the traveling salesman problem: a survey. *SIAM Review*, 40(3):496–546, 1998.

4 Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

5 Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

6 Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12, 1999.

7 Fanwei Kong, Liang Yuan, Yuan F Zheng, and Weidong Chen. Automatic liquid handling for life science: a critical review of the current state of the art. *Journal of Laboratory Automation*, 17(3):169–185, 2012.

8 Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

9 Christoph B Messner, Vadim Demichev, Daniel Wendisch, Laura Michalick, Matthew White, Anja Freiwald, Kathrin Textoris-Taube, Spyros I Vernardis, Anna-Sophia Egger, Marco Kreidl, et al. Ultra-high-throughput clinical proteomics reveals classifiers of COVID-19 infection. *Cell Systems*, 11(1):11–24, 2020.

10 Jeremiah J Minich, Greg Humphrey, Rodolfo AS Benitez, Jon Sanders, Austin Swafford, Eric E Allen, and Rob Knight. High-throughput miniaturized 16s rRNA amplicon library preparation reduces costs while preserving microbiome integrity. *mSystems*, 3(6):e00166–18, 2018.

11 Ian M Pendleton, Gary Cattabriga, Zhi Li, Mansoor Ani Najeeb, Sorelle A Friedler, Alexander J Norquist, Emory M Chan, and Joshua Schrier. Experiment specification, capture and laboratory automation technology (ESCALATE): a software pipeline for automated chemical experimentation and data management. *MRS Communications*, 9(3):846–859, 2019.

12 Sam Powers, Eliot Xing, Eric Kolve, Roozbeh Mottaghi, and Abhinav Gupta. CORA: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In *Conference on Lifelong Learning Agents*, pages 705–743. PMLR, 2022.

13 Justin Vrana, Orlando de Lange, Yaoyu Yang, Garrett Newman, Ayesha Saleem, Abraham Miller, Cameron Cordray, Samer Halabiya, Michelle Parks, Eriberto Lopez, et al. Aquarium: open-source laboratory software for design, execution and data management. *Synthetic Biology*, 6(1):ysab006, 2021.

14 Yishan Wang, Hanyujie Kang, Xuefeng Liu, and Zhaohui Tong. Combination of RT-qPCR testing and clinical features for diagnosis of COVID-19 facilitates management of SARS-CoV-2 outbreak. *Journal of Medical Virology*, 92(6):538, 2020.

15 Ellis Whitehead, Fabian Rudolf, Hans-Michael Kaltenbach, and Jörg Stelling. Automated planning enables complex protocols on liquid-handling robots. *ACS Synthetic Biology*, 7(3):922–932, 2018.

16 Jing Wui Yeoh, Neil Swainston, Peter Vegh, Valentin Zulkower, Pablo Carbonell, Maciej B Holowko, Gopal Peddinti, and Chueh Loo Poh. SynBiopython: an open-source software library for Synthetic Biology. *Synthetic Biology*, 6(1), 2021.

17 Min Zhu, Pingbo Zhang, Minghui Geng-Spyropoulos, Ruin Moaddel, Richard D Semba, and Luigi Ferrucci. A robotic protocol for high-throughput processing of samples for selected reaction monitoring assays. *Proteomics*, 17(6):1600339, 2017.