Estimating Policy Functions in Payments Systems Using Reinforcement Learning*

Pablo S. Castro[1], Ajit Desai[2], Han Du[2], Rodney Garratt[3] and Francisco Rivadeneyra[2]

[1]Google Research, Brain Team
[2]Bank of Canada
[3]University of California Santa Barbara

September 29, 2021

Abstract

This paper uses reinforcement learning (RL) to approximate the policy rules of banks participating in a high-value payments system. The objective of the agents is to learn a policy function for the choice of amount of liquidity provided to the system at the beginning of the day. Individual choices have complex strategic effects precluding a closed form solution of the optimal policy, except in simple cases. We show that in a simplified two-agent setting, agents using reinforcement learning do learn the optimal policy that minimizes the cost of processing their individual payments. We also show that in more complex settings, both agents learn to reduce their liquidity costs. Our results show the applicability of RL to estimate best-response functions in real-world strategic games.

JEL Classification: A12, C7, D83, E42, E58

Keywords: Artificial intelligence, Reinforcement learning, Payments systems

*The opinions here are of the authors and do not necessarily reflect the ones of the Bank of Canada or Google. The order of the authors is alphabetical. Thanks to participants of the Machine Learning for Economic Policy workshop at Neural Information Processing Systems (NeurIPS 2020) conference, International Conference on Economic Modeling and Data Science (EcoMod 2021), Regional Conference on Payments and Market Infrastructures (2021), and seminar participants at the Bank of Canada for opinions and suggestions.

1 Introduction

High-value payments systems are used to settle transactions between large financial institutions and are considered part of the core national financial infrastructure.[1] Most high-value payments systems settle in real time and require financial institutions to fund their payments with costly central bank liquidity. Fortunately for banks, they do not have to fund the entire aggregate value of their daily payments. Rather, they can, in part, count on liquidity from incoming payments (McAndrews and Rajan 2000).

Banks make hundreds of payments to each other throughout the day. While banks make some payments related to their own activity, the vast majority of the payments they make reflect underlying customer requests, which are determined from real economic activity. Banks take the payment requests they receive from customers as exogenous. Payment requests from customers must be processed on the day they are received, but for most payments banks have flexibility in terms of when, during the day, they process payment requests.[2]

Each bank starts the day with an initial liquidity allocation that it arranges with the central bank.[3] As payments come in and out, the bank's initial liquidity balance fluctuates. If a bank's liquidity balance falls to zero, it cannot make any payments until new liquidity comes in.[4] This bank suffers a delay cost that is proportional to the period of delay. The delay costs represents customer dissatisfaction with the lack of timely processing of their requests. In addition, if at the end of the day the bank does not have sufficient liquidity to complete its payments, it must borrow additional liquidity from the central bank. The cost of borrowing at the end of day is greater than the cost of providing liquidity at the beginning of the day.

A key decision that banks make in these systems is the provision of initial liquidity. This is a strategic decision since the timing of the arrival of incoming liquidity from others depends on how much initial liquidity they post. If others post more liquidity, which means incoming payments are likely to arrive sooner, then a bank can make more of its own payments with recycled liquidity and thus require less of its own liquidity (Bech and Garratt 2003).

The initial liquidity decision of banks is complex. In a system with multiple banks and multiple periods in which payment requests arrive, researchers have been unable to solve for the equilibrium initial liquidity profile. This raises two interesting questions. First, if equilibrium strategies are difficult to determine, what do banks actually do? And second, can advanced machine learning techniques find equilibrium solutions and hence potentially be a valuable guide for practitioners and infrastructure providers?

[1] To get a sense of their importance, in Canada the Large Value Transfer System (LVTS) processes payment values equivalent to annual GDP every five days.

[2] An exception are CLS payments, which must be processed immediately upon request.

[3] Most large value payments systems around the world fit this description, although systems differ in terms of how this is done. An exception is Fedwire Funds in the United States, which allows uncollateralized overdrafts.

[4] Ignoring overdraft possibilities or posting new collateral.

There is a burgeoning economics literature on agent-based modelling of actors in payments systems, which we discuss below, that is attempting to answer the first question. This paper focuses largely on the second question. That is, as an initial investigation into the potential application of machine learning to payments systems, we seek to explore the extent to which these techniques can deliver solutions.

Estimating policy functions in payments system is important for two additional reasons. First, looking down the road to a time when system participants may be using these algorithms, knowledge of these functions could help regulators achieve their mandates of ensuring safety and efficiency of these systems. For example, regulators could assess the liquidity and risk implications to the system should banks themselves attempt to use these machine learning algorithms for their liquidity management.[5] Second, understanding equilibrium behaviour of participants through these functions can be helpful to design new payments systems. This is particularly relevant given the worldwide trend to modernize core national payments systems.[6]

We estimate the best-response functions for liquidity provision in a stylized large value transfer system using machine learning techniques. In particular, we apply an approach called reinforcement learning (RL). RL is a computational approach to automate learning of sequential decision-making. RL emphasizes the learning of an agent by trial and error through the association of actions to states that maximize certain objectives in the form of cumulative rewards (Sutton and Barto 2018). RL is particularly suitable for the problem of estimating the policy functions of the participants in a payments system. In the case of the payments system, the experience provided by trial and error should allow agents to learn to process payments at the lowest possible cost. RL allows a very broad specification of the learning environment, which could be stochastic, have a large state space, include more than one agent learning simultaneously, etc.

We start off with a two-period model with known customer payment requests. We then consider a twelve-period model in which payment requests are drawn from payment profiles estimated from the Canadian Large Value Transfer System (LVTS) data. In either case, the RL agent does not know if the payment demand profile is constant or drawn from a distribution, or if the distribution changes over time. Rather, the RL agent learns about the environment as it experiments with liquidity choices in an attempt to minimize its cost of processing payments.

The twelve-period model with randomly drawn payment requests is more realistic. We consider the two-period fixed-payment model because we can solve it analytically. We derive

[5] Some papers are already examining the policy implications of RL agents trained to solve problems like algorithmic pricing. See Calvano et al. (2020).

[6] Australia and Denmark launched new wholesale systems in 2017, Canada is expected to launch its own in 2021, and other countries have announced new systems like FedNow in the United States. Traditionally, the process for designing a new high-value payments system relies on the empirical evaluation of different design options through simulation of the outcomes of a potential system using historical data as inputs (Rivadeneyra and Zhang 2020).

the best-response functions of each bank and compute the Nash equilibrium. This provides a target solution for the RL estimation and allows us to determine unequivocally whether or not the RL procedure works. We cannot solve the twelve-period model analytically, however, we can solve it using brute-force search, which again provides the target outcomes.

We demonstrate that RL techniques can be used successfully to replicate equilibrium behaviour. In addition to demonstrating the usefulness of machine learning in guiding participant behaviour, we observe quantitatively the importance of the relative size of the delay and initial liquidity costs for their initial liquidity choices. Having estimates of the sensitivity of the agent's best responses across different levels of delay cost is important because delay cost is unobservable to researchers and policymakers.

We proceed as follows. Section 2 discusses related literature. In Section 3 we describe the stylized payments system environment we will use in the training of the RL agents. In Section 4 we derive the game-theoretical equilibrium to the initial liquidity problem in a specialized version of the environment. We will use these results to benchmark to the results of the training. Section 5 discusses the RL algorithm and the multi-agent learning setup. In Section 6 we turn to the training of the initial liquidity problem. We examine two versions of this problem: one with two intraday periods and one with twelve intraday periods. Section 7 provides some concluding remarks. Several appendices show the training for the problem of the intraday timing of payments, the pseudocodes and robustness exercises.

2 Related Literature

As an economics paper using machine learning techniques, we are related to the expanding literature exploring the connections between these two fields. Calvano et al. (2020) study the implications of RL agents trained to solve important economic problems. They examine the likelihood that the pricing algorithms of multiple sellers trained with RL could result in collusion without the use of any coordination, a potentially challenging problem for competition policy. Igami (2020) examines the similarities and differences between structural estimation (as understood in econometrics) and dynamic programming and RL in the context of two-player, perfect-information, zero-sum games like chess and Go. The game we examine in this paper is more complex, most importantly because it's a game of imperfect information that can yield cooperative or non-cooperative solutions.

In terms of the application to payments systems, our work is related to the empirical work by Galbiati and Soramäki (2011), who examine the initial liquidity choice problem and simulate an agent-based version of that game. Similarly, Arciero et al. (2008) used an agent-based model of a payments system to study interbank flows of liquidity and propagation of systemic risk under critical conditions. These models are useful to understand the intuition of the behaviour but limited insofar as agent-based models make strong assumptions about

the shape of the objective functions. In our case, the only assumption we need to use RL is that the agent seeks to minimize the costs of processing payments.

Our work is also related to the applied game theory literature. Previous work is the theoretical model of Bech and Garratt (2003), which solved for the equilibrium of the intraday liquidity game. In that paper, banks that receive a customer payment request make a strategic decision whether to provide liquidity in the morning period or wait until the afternoon, hoping that they can then fund this payment using incoming liquidity. Waiting is costly, however, since, as we assume in this paper, there is a customer dissatisfaction cost associated with delaying payments. The initial liquidity provision decision agents make in our paper is closely related to the decision made by banks in Bech and Garratt (2003). A key difference is that our agents make this decision *before* observing any of the customer payment requests. We also tackle more complicated payment request scenarios.

More generally, we are related to the game theory literature exploring how equilibria of games emerge through various methods like reinforcement learning (Roth and Erev 1995) or others (Fudenberg and Levine 2016). Our results, showing the convergence of the trained policy functions to the optimal solutions for the special cases for which we can solve them, confirm the convergence conditions to a unique equilibrium derived by Funai (2019).

3 The Payments System Environment

Our environment is a real-time gross settlement (RTGS) payments system. To settle transactions, participants in the system require liquidity provided by the central bank. Transactions are settled in real time without netting, which means that, for example, two offsetting positions between two banks cannot be cancelled out even if they were submitted to the system in the same instant and for the same amount. Banks participating in the system use it to satisfy the exogenous and random payment demands they receive throughout the day from their clients. Banks can source the liquidity required to process the payments from collateral they post at the central bank, which is costly due to its opportunity cost, or from incoming payments from other banks, which is not as costly. This creates an incentive to delay processing their own payments to wait for incoming payments. On the other hand, delaying payments is costly to banks if they don't service their clients in a timely fashion. Therefore, the bank's problem is to choose a policy that balances the cost of initial liquidity and the cost of delay by choosing the level of initial liquidity and the timing of their own payments, all of which conditional on the other bank's own policies. At the end of the day, banks are required to satisfy all payment demands. If they don't have enough liquidity, they can borrow from the central bank at a cost higher than the cost of collateral at the beginning of the day.

In our environment, each day is an episode e that is divided into intraday periods $t = 0, 1, 2, ..., T$. At the beginning of the day $t = 0$, the agent is faced with making the decision to

Table 1: Timeline, decisions of the agent and constraints of the environment

Beginning of the day, $t = 0$	
Available collateral:	$\mathcal{B}$
Initial liquidity decision:	$x_0 \in [0, 1]$
Liquidity allocation:	$\ell_0 = x_0 \cdot \mathcal{B}$
Cost of initial liquidity:	$r_c \cdot \ell_0$
Intraday periods, $t = 1, ..., T-1$	
Payment demand:	P_t
Decision to send:	$x_t \in [0, 1]$
Liquidity constraint:	$P_t x_t \leq \ell_{t-1}$
Agent receives incoming payments:	R_t
Evolution of liquidity:	$\ell_t = \ell_{t-1} - P_t x_t + R_t$
Cost of delay:	$r_d \cdot P_t(1 - x_t)$
End-of-day borrowing, $t = T$	
Borrowing from central bank (for remaining payments):	ℓ_{cb}
Cost of end-of-day borrowing:	$r_b \cdot \ell_{cb}$

allocate share $x_0 \in [0, 1]$ of collateral $\mathcal{B}$ for its initial liquidity, $\ell_0 = x_0 \cdot \mathcal{B}$. Subsequently, at each intraday period $t = 1, ..., T-1$ the agent receives payment demands P_t and has to choose the share $x_t \in [0, 1]$ of requested payments to send in that intraday period. The agent's choice is constrained by the available liquidity in that intraday period so that $P_t x_t \leq \ell_{t-1}$. At the end of each intraday period the agent receives payments from other agents, which we denote by R_t. Then the available liquidity evolves according to: $\ell_t = \ell_{t-1} - P_t x_t + R_t$.[7]

The associated cost with the initial liquidity choice is $r_c \cdot \ell_0$, and the cost of delay is given by $r_d \cdot P_t(1 - x_t)$. If the agent does not have enough liquidity at $T - 1$ to satisfy all the remaining payment demands, the agent can borrow from the central bank at rate r_b, which is more expensive than the morning liquidity cost r_c. Table 1 summarizes the timeline and decisions of the agent.

Learning in the high-value payments system is appropriately modelled by the RL approach. RL relies on the assumption that an agent learns from observing the result of its actions in the form of explicit rewards given by the environment. By repeatedly interacting with the environment, the agent can learn to associate state-action pairs that bring high rewards.

[7]For ease of notation, P_t includes any payment demand that was not sent in the previous intraday period $P_{t-1} \cdot (1 - x_{t-1})$.

Our model of the environment abstracts away from two important dimensions of payments systems: the indivisibility of payments and the interbank market for liquidity. Typically, due to legal restrictions, a payment request can only be fulfilled by the bank in its entirety or not at all. In other words, the payment amount cannot be fulfilled by splitting the payment into smaller amounts. In our learning setup, however, agents are presented with continuous payment demands at each intraday period and are assumed to pay the largest fraction $x_t \in [0, 1]$ they can afford given available liquidity.[8] Divisible payments may be a reasonable approximation of reality when aggregate payment requests are large relative to the size of individual payments. The second aspect we do not consider in this paper is that banks participating in actual payments systems can typically source liquidity in an interbank market, and not only from the central bank or from incoming payments. Including these two dimensions of the environment are avenues of future research.

4 The Initial Liquidity Game

The initial liquidity game is a one-shot game in which agents $i = \{A, B\}$ simultaneously choose their allocation of initial liquidity, ℓ_0, to minimize the cost of processing all payments. We present the case where payments are known and the day is composed of two periods, $T = 2$, however our empirical analysis also includes the case of unknown, randomly drawn payments with $T = 12$. Let the total payment demand to agent i be the sum of each intraday period payment demand:

$$P^i = P_1^i + P_2^i.$$

We assume that both agents behave optimally with respect to the intraday policy conditional on having available liquidity, i.e., $x_t = 1$, so that the remaining value of payments to be sent is:

$$\max(P_t - \ell_{t-1}, 0) \qquad t = 1, 2.$$

Then, the total cost in terms of the initial liquidity choice ℓ_0 is:

$$\mathcal{R}(\ell_0) = r_c \ell_0 + \max(P_1 - \ell_0, 0) \cdot r_d + \max(P_2 + P_1 - R_1 - \ell_0, 0) \cdot r_b \tag{1}$$

where $r_c < r_d < r_b$ are the per-dollar cost of initial liquidity, delay and end-of-day borrowing, respectively. Therefore, the first term is the total cost of initial liquidity, the second term is the cost of delay after period $t = 1$ and the third term is the cost of borrowing at the end of the day to send the remaining payments. Note that R_1 are the payments received from the

[8] The assumed decision for agents to pay as much as they can in each intraday period is in fact optimal given divisible payments. Because payments are assumed to be divisible, banks do not have an incentive to hold back on making payments during the day as a precautionary move to conserve liquidity for large payments that might arrive later in the day. In Appendix A we show that our RL agents can learn this simple policy.

other agent in the first period but only available in the second period. See the definitions in Table 1.

In general, we have four cases depending on two conditions. The first condition is the relative size of initial liquidity and the first period payment demand, $\ell_0 \lessgtr P_1$. This determines the delay cost in the first period irrespective of the other agent's choice, as the sent payments in that period are not available to the agent until the second period. The second condition is the relative size of the initial liquidity choice and the difference between the total payments demand and the received payment, $\ell_0 \lessgtr P - R_1$. The term on the right is the net intraday payment demand. For each case, the total cost is as follows:

1. $\ell_0 > P_1$ & $\ell_0 > P - R_1$. Initial liquidity is sufficient to cover the first period payment demand as well as the second period, irrespective of the value of the received payment. The cost is only the initial liquidity cost:

$$r_c \ell_0.$$

2. $\ell_0 > P_1$ & $\ell_0 < P - R_1$. Initial liquidity is sufficient to cover the first period payment demand but not the second period given the value R_1 of the received payment. The total cost is the initial liquidity cost and cost of net borrowing at the end of the day:

$$(r_c - r_b)\ell_0 + (P - R_1)r_b.$$

3. $\ell_0 < P_1$ & $\ell_0 > P - R_1$. Initial liquidity is short to cover the first period payment demand but enough for the second period given the R_1 of the received payment, which would be expected to be relatively large in this case. The total cost is the initial liquidity cost and cost of delaying $P_1 - \ell_0$:

$$(r_c - r_d)\ell_0 + P_1 r_d.$$

4. $\ell_0 < P_1$ & $\ell_0 < P - R_1$. Initial liquidity is short to cover the first period payment demand and short for the second period given the R_1. The total cost is the initial liquidity cost, cost of delaying $P_1 - \ell_0$ and of borrowing the rest:

$$(r_c - r_d - r_b)\ell_0 + P_1 r_d + (P - R_1)r_b.$$

We make the following assumptions about the behaviour of agents and the environment. First, agents are risk neutral and will choose their initial liquidity to minimize the total liquidity cost. Second, we assume that the payment demand profile is common knowledge.

And third, $r_c < r_b$ so the cost of end-of-day borrowing is higher than in the morning and there is no compensation for positive end-of-day balances if, for example, $R_1 > P$.

Definition 1 *A pure strategy Nash equilibrium of the initial liquidity game is a pair of initial liquidity choices* (ℓ_0^A, ℓ_0^B) *such that for* $i = \{A, B\}$

$$\ell_0^i = \arg\min \mathcal{R}(\ell_0^i).$$

The best-response function for each agent is:

$$\ell^i = P_1^i + \max\left(P_2^i - \min(\ell^{-i}, P_1^{-i}), 0\right) \tag{2}$$

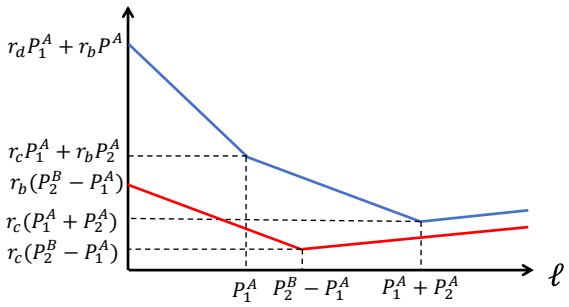
where $-i$ means not i. This is intuitive: since the payment expected to be received is not available until the second period, the agent should at least allocate enough initial liquidity to cover the first period payment. Deviating from this strategy would only add delay cost, which is more costly than the initial liquidity cost. The second term is the net liquidity needed for the second period payment that is net of the payment received. Since we assume both agents follow the optimal intraday payment policy (i.e., send as much payments conditional on already available liquidity), the payment received by i will be the lesser of the first period payment demand and liquidity choice of agent $-i$. It is easy to verify that given $r_c < r_d < r_b$, there is a unique minimum of the cost function 1. In fact, even if the other agent were to deviate, say, by choosing $\ell_0^{-i} = 0$, agent i would still be better off by choosing $\ell_0^i = P_1^i + P_2^i$ according to 2. Therefore, under the assumption of the parameter values, there is a unique Nash equilibrium. Figure 1 illustrates this graphically using a special case where $P_1^A > 0$ for agent A and $P_1^B = 0$ for B.

The equilibrium of this game provides a benchmark to the machine learning problem presented later in Section 6. As discussed by Fudenberg and Levine (2016), in game-theoretic setups of learning, if agents keep track of their frequency of play and the actions of players converge, they will converge to the Nash equilibrium or the Nash distribution (known as the quantal response equilibrium) when actions have a degree of randomness.

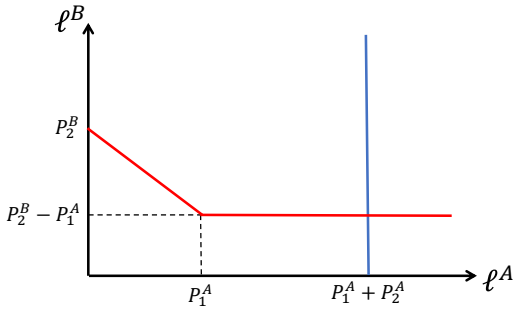
Noticing the differences in setups between the machine learning environment and the game-theoretic solution illustrates the power of reinforcement learning algorithms. As is typical in some setups using the Nash equilibrium concept, players in the game have complete knowledge of the environment and in particular of the payoffs of the opponent. In our case, this requires knowing the size of the state space, the payment profiles and the utility function of the opponent. Consequently, learning has no purpose and does not occur. In contrast, as we will later show, the reinforcement learning agents do not know the payment profiles nor the fact that there is another agent or how it behaves. Therefore, results showing that the reinforcement learning agent achieves the optimal behaviour supports our use of the algorithm.

Figure 1: Cost function (left) and equilibrium of the game (right). Agent A is in blue and B in red with $P_1^A > 0$, $P_1^B = 0$ and $P_2^B > P_1^A$. Without loss of generality, this example shows that there is a unique minimum for the total cost for both agents and a unique pure strategy Nash equilibrium.

(a) Cost as a function of initial liquidity choice



(b) Equilibrium



5 RTGS as a Multi-Agent Reinforcement Learning Problem

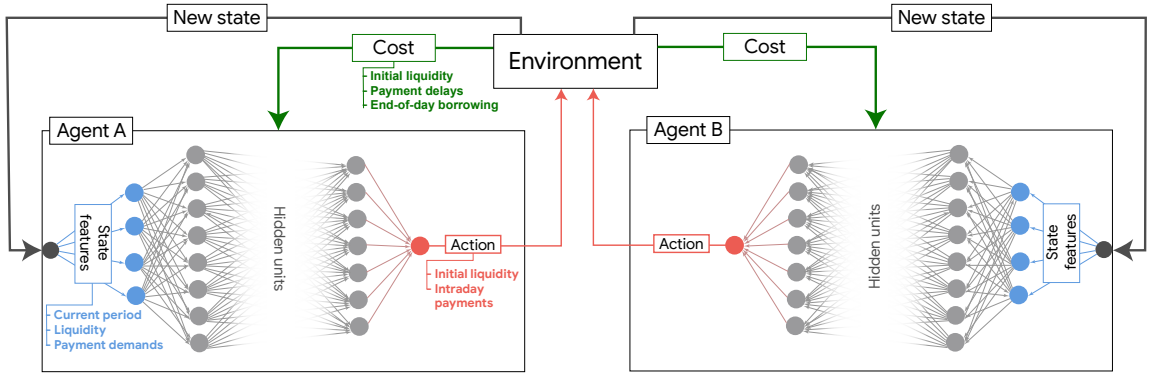
Reinforcement learning (RL) methods are used for optimizing the behaviour of an agent in an uncertain environment. The agent improves its behaviour by interacting with the environment via the selection of *actions* (from a set of possible actions $\mathcal{A}$) in discrete time steps, and transitioning between *states* of the environment (from a set of possible states $\mathcal{S}$). Specifically, while at state $s_t \in \mathcal{S}$ at time step t, the agent selects action $a_t \in \mathcal{A}$; the environment responds with a new state $s_{t+1} \in \mathcal{S}$ and a numerical cost r_t. The goal of the agent is to discover a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ mapping states to actions such that following this policy will minimize the cumulative sum of costs incurred. Traditionally, there is a single agent being trained; in our current work, we train two agents concurrently in the same environment, which results in more complex learning dynamics. See Appendix B for further technical details.

Figure 2 depicts our learning setup. Two agents in the payments system interact with the same environment by each making independent action choices. Based on the actions chosen by each agent, the environment will return a cost to each, as well as a new state. The state for each agent consists of the current period, the agent's current liquidity and the payment demands. Although the same learning algorithm is used for both agents, each agent executes this algorithm independently and maintains its own policy. Appendix B provides further details of the learning setup.

Given the multiple complexities involved with the setting just described, we aim to isolate some of them so as to better explore the learning dynamics and more carefully develop algorithms for handling the complete, independent multi-agent setting. Our strategy for training the agents is to separately tackle the initial liquidity choice and the intraday payments decision. This strategy allows us to verify that our learning algorithm is performing as expected. The training of the agents proceeds as follows.

For the initial liquidity decision, the agent will be trained while the intraday decision is

Figure 2: Reinforcement learning in the context of a payments system, where the agents represent separate banks. Each agent must learn to optimize the choice of initial liquidity as well as the choice of intraday payments. Each agent maintains its own policy, represented as a neural network (in grey), to make the action choices (in red), conditioned on the current state (in blue). The environment provides the agents with the cost (composed of the initial liquidity, delay and end-of-day borrowing costs in green) which they use to improve their policy.



fixed to send all payments.[9] The optimal initial liquidity policy, given the intraday policy, is the one that minimizes the cost of processing all payments. Intuitively, when the agent behaves optimally in its intraday decision, making the initial liquidity choice requires estimating its own payment demand profile and the payments it expects to receive.

This is akin to the well-known bandit problems (see Sutton and Barto (2018), Chapter 2), with the added complexity that the transition dynamics for each agent is affected by the learning process of the other agent. Under this lens, this bandit setting allows us to evaluate the learning process in the independent multi-agent setting without the compounding effects present when dealing with sequential decision problems.

The learning task is episodic. In our setup a day is divided into multiple intraday periods, which we take to be one hour each. Each episode starts with the beginning of the payment cycle and ends at the end of the payment cycle. In the last intraday period, banks are required to satisfy all payment demands. If they do not have enough liquidity, they will automatically borrow the shortfall amount from the central bank. The central bank is not strategic and will always lend the necessary amount at the fixed borrowing cost.

We simplify further by limiting our training to only two agents simultaneously, as shown in Figure 2. Both agents are transferring payments to each other through the payments system. The payments demand is exogenous and received throughout the day from their clients. To a certain extent, training only two agents is not a big limitation in our exercise if, in reality,

[9]For the intraday payment decision, we know the analytical solution: in order to minimize the intraday cost of processing payments, the agent must learn to send as much as it can at every intraday period, $x_t = 1 \quad \forall \quad t$. In Appendix A we verify that our agents indeed learn this simple policy.

banks use only one policy function to manage their payments irrespective of which agent they are sending to and if they do not distinguish from which agent they have received a payment from. Therefore, a similar training setup would be agents learning a policy to process payment demands to many other agents as long as the learning agent does not know or use information specific to the recipient. We plan to investigate more interacting agents in future work.[10]

Note that the information provided to the agents in the state does not reveal if they are playing against one or more players. To see this more clearly, imagine that while we train agent A, agent B follows a policy that does not depend on the state that agent A observes; then, after training, agent A should infer that the payments it receives do not depend on its own choices or the state. In this non-strategic environment, the learning task of agent A would be to estimate the payments demand from its clients and derive the policy that minimizes its costs. On the other hand, when training the agents simultaneously, if the choices of agent A affect the available liquidity of agent B, then agent A should infer that the environment is more complicated than simply estimating its own payment demand.

6 Initial Liquidity Policy Estimation

6.1 Learning setup

The learning setup for the initial liquidity policy is as follows:

- **Number of intraday periods:** We will consider two scenarios, one in which each episode is divided into only two intraday periods, $T = 2$, and one in which the episode is divided into twelve intraday periods, $T = 12$. The reason for testing the first scenario is that for this case we have the analytical solution to the policy and therefore we can verify if the learning algorithm is performing as expected.

- **Agents:** Agents A and B are trained simultaneously over multiple episodes (depending on the exercise).

- **Payments demand:** The payments demand profile is drawn from the LVTS data shown in Figure 3.

- **Intraday payment policy:** In the current environment there is no benefit in delaying payments, therefore the optimal intraday policy is to always send all payments. Therefore, both agents' intraday policy is to send all the payments if possible or the entire value of

[10] In the payments system environment, a multi-agent setup presents particular difficulties for the training exercise because the payment decisions of every agent affect the available liquidity of all agents. Furthermore, for each agent we would need to train an intraday payment policy function for individually deciding which payments to send to each of the other agents.

liquidity in that intraday period, i.e., if $P_t < \ell_{t-1}$ send $x_t = 1$, otherwise they choose x_t such that payments sent are equal to ℓ_{t-1}.

- **State space:** In every episode e, the agent observes the entire vector of intraday payment demands.

- **Action space:** The action space is the liquidity choice as the fraction x_0 of the available collateral $\mathcal{B}$. We discretize the unit interval for x_0 into 21 evenly-spaced fractions: $\{0,\ 0.05,\ 0.1,\ 0.15,\ ...,\ 1\}$. Recall also that $\ell_0 = x_0 \cdot \mathcal{B}$.

- **Cost:** The total processing cost per episode is:

$$\mathcal{R} = r_c \ell_0 + \sum_{t=1}^{T-1} P_t(1 - x_t) \cdot r_d + r_b \ell_{cb},$$

where r_c is the cost of initial liquidity, r_d is the intraday delay cost and r_b is the end-of-day borrowing cost. The last two terms are included because the agent might incur delay costs and a borrowing cost if its chosen initial liquidity was not sufficient to cover all payments. Note that in the last period, T, the agent borrows an amount equal to its shortage to satisfy that period's payments demand. Therefore, no delay cost occurs in the last period.

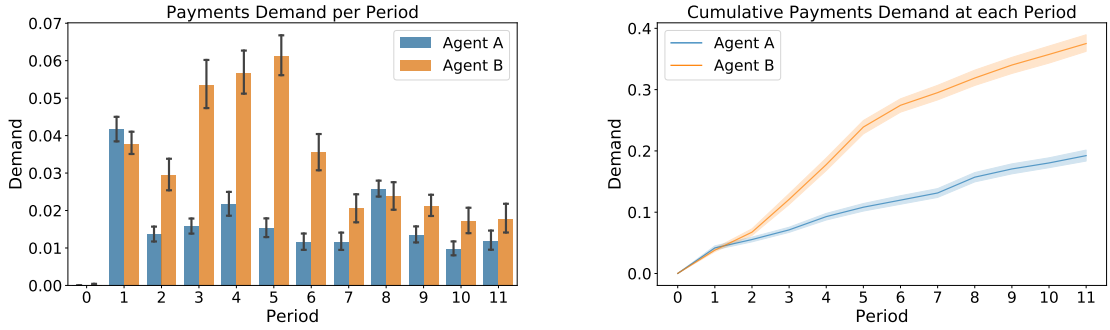
6.2 Data and parameters

As payment demands we use transactions observed between two actual participants in the Canadian Large Value Transfer System (LVTS).[11] Each episode is a daily LVTS payments cycle that runs, typically, from 6 a.m. to 6 p.m. We divide the episodes into twelve hourly periods. We sum the value of transactions between these two participants for each hour; to standardize them, we divide the payment values by the collateral each agent pledged to the system on each particular day. We selected a sample of 380 business days between January 02, 2018, and August 30, 2019. The resulting payment demands are displayed in Figure 3. Agent B's average payment demands and range of payment values are higher than that of agent A's. Agent B's demand peaks in the earlier part of the day before slowing down in the last few periods. To perform the model training and evaluation, we randomly split the sample, reserving 90% for training and 10% for testing.

For the initial liquidity decision problem, we consider two scenarios. The first scenario is a problem with only two intraday periods, for which we have the analytical solution of the optimal policy. The second scenario is a problem with twelve intraday periods. For both

[11] We are not permitted to reveal the identities of the participants. However, these two participants were chosen because they are similar in terms of the value they process in the LVTS. For robustness, in Appendix F.5 we show two other pairs of participants.

Figure 3: Data used for training and testing, showing the hourly sum of payment flows between two participants of the Canadian LVTS. The left plot shows the hourly averages and standard deviation over the sample period. The right plot is the cumulative payments value. Sample is between January 02, 2018, and August 30, 2019. Payments are standardized daily as a percentage of the collateral pledged to the LVTS system in each given day.



scenarios we use the following costs: for the initial liquidity cost $r_c = 0.1$, for the per-period delay cost $r_d = 0.2$ and for the end-of-day borrowing cost $r_b = 0.4$. Compared to the initial liquidity cost, the high end-of-day cost should discourage agents from borrowing at the end of the day instead of allocating sufficient liquidity at the beginning of the day.

The end-of-day borrowing cost (per dollar) is chosen to be the largest of the three because the design of wholesale payments systems encourages participants to provide liquidity early in the day instead of borrowing from the central bank. Consistent with this design, the data show that banks rarely borrow from the central bank. The borrowing rate from the central bank is 25 basis points above the interbank market for overnight loans, which is policy rate of the other borrowing alternative for banks short of liquidity. At the end of the day, these loans are typically priced at the policy rate.[12]

Furthermore, we observe in the LVTS data that banks allocate substantial initial liquidity to the system but not enough to cover all their payment demands alone. This indicates that, while the cost of liquidity in the morning is smaller than the cost of borrowing in the evening, liquidity in the morning is still costly. One potential source of this cost are collateral reallocation costs: once the collateral has been pledged to the Bank of Canada to collateralize their payments in the system, changing this portfolio of securities is cumbersome (Bulusu and Guérin 2019). This cost is smaller than the 25 basis points band between the target rate and the borrowing cost of the bank. Note that this cost is not the opportunity cost of the collateral itself as banks still receive the coupon proceeds from the pledged securities.

[12]The rules governing a payments system are explicitly stated in legal documentation and to a large extent allow banks to estimate the cost of their payments operations, namely the cost of initial liquidity, any compensation to their clients from delaying their payments and any end-of-day borrowing from the central bank to settle all their payment obligations at the end of the day.

Choosing the per-period delay cost is more difficult because it depends on the contractual arrangements that banks have with their clients, which we do not observe.[13] The theoretical literature has analyzed various equilibria in which the delay cost can be higher or lower than both the initial liquidity or the end-of-day liquidity (Bech and Garratt 2003). Therefore, there is no a priori reason to choose one or the other. The empirical literature has not estimated the relative magnitude of the delay cost. Therefore, as our baseline we choose a delay cost in between the initial liquidity cost and the end-of-day cost, $r_c < r_d < r_b$. For robustness, Appendix F.4 presents alternative cases.

For training, we utilize the REINFORCE algorithm (Williams 1992) with a RL setup and implement it in Python using TensorFlow (Abadi et al. 2016). Each agent's policy function π is approximated using a feed-forward neural network. Additional details on the REINFORCE algorithm are discussed in Appendix A, and further details on the learning setup, implementation and model parameters selection are discussed in Appendix D.

For each training episode, we simultaneously generate multiple trajectories using each agent's current policy, where each trajectory is a complete path of state, action and reward. Subsequently, using the experience gathered through multiple trajectories, we update the policy function at the end of each episode. This way, we train the agents over multiple episodes and monitor each agent's learning progress with the average cost over the training episodes. To compute the confidence intervals of the payment decisions at each intraday period, we simultaneously perform multiple independent training exercises.

6.3 Results of the training with two intraday periods, $T = 2$

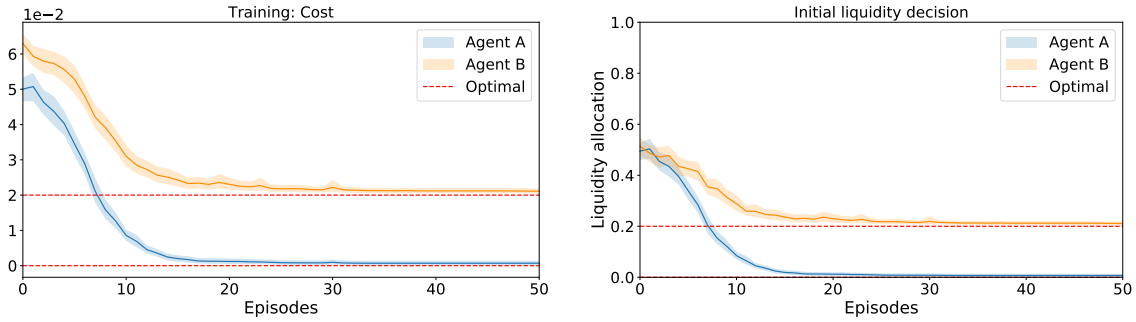
In this scenario, to help in the illustration of the intuition, we use dummy payment demands that have the same support as the discretized action space: $P^A = [0,\ 0.15]$, $P^B = [0.15,\ 0.05]$. Each entry in the vector represents the demand in each intraday period. In this case the cost function simplifies to $\mathcal{R} = r_c\ell_0 + P_1(1 - x_1)r_d + r_b\ell_{cb}$ for both the agents because in the second period all payment demands have to be satisfied, possibly by borrowing at cost r_b.

Notice that agent B has no incoming payment from A in the first period. Therefore, to avoid delay and borrowing costs (recall $r_c < r_d < r_b$), B has to allocate initial liquidity equal to its total payment demands. If agent B performs the optimal action, then agent A will have an incoming payment in the first period, which is equal to its payment in the second period. Consequently, agent A can use the received payment to meet its demand. Therefore, the optimal choice of A is to allocate nothing. In sum, the optimal liquidity choices for agent A and agent B are 0 and 0.2. Accordingly, the optimal costs are $\mathcal{R}^A = 0$ and $\mathcal{R}^B = 0.2 \times 0.1 = 0.02$. See Section 4 for the formal derivation of the optimal choices.

[13] Another source of delay cost we are not modelling here is the failure to meet an obligation to send an urgent payment to a financial market infrastructure. This delay would incur potentially extremely high penalties from regulators.

Figure 4 shows the average training costs (left) and actions (right) for both agents A and B obtained by performing 50 independent training exercises using 50 episodes in each exercise. It can be observed that after initial exploration, both agents quickly learn to minimize the cost by selecting actions closer to their optimal choices. Agent B, which has higher payment demands and has no incoming payments, converged to a higher cost than agent A.

Figure 4: Average cost (left) and average initial liquidity decision (right) over training episodes of the simplified initial liquidity problem with only two intraday periods. The cost converges to the optimal for both agents ($\mathcal{R}^A = r_c \ell_0^A = 0$ and $\mathcal{R}^B = r_c \ell_0^B = 0.02$). Likewise, the initial liquidity decision converges to the optimal choice shown in red for both agents, shown here in the scale of the bins of the action space vector. Solid lines are the averages and shaded areas are the 99% confidence interval bands.



6.4 Results of the training with twelve intraday periods, $T = 12$

In the previous section, we showed that agents learn to select optimal initial liquidity for the simplified two-period setting. In the current scenario, we test the agent's ability to learn the optimal liquidity allocation in the twelve-period setting when faced with the LVTS payments demand showed earlier in Figure 3.

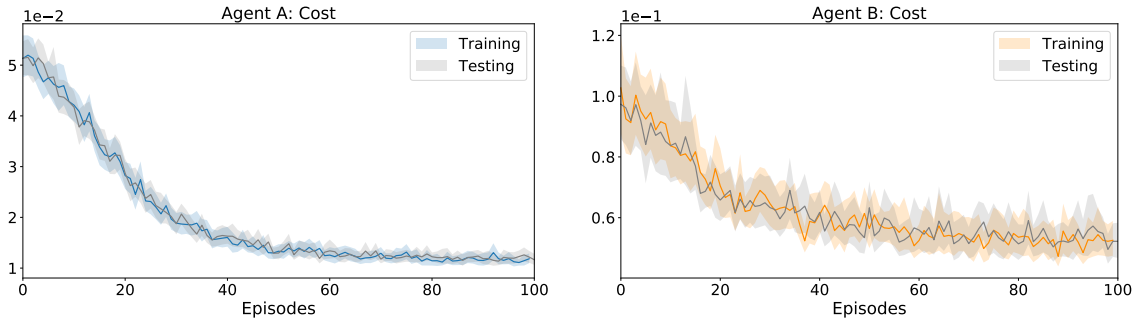
Similar to the two-period case, both agents are trained simultaneously while playing against each other, using the REINFORCE algorithm for 100 episodes. The average cost during training and testing computed using 50 independent exercises is shown in Figure 5. The cost for both agents plateaus to their minimum observed during this training around the 60th training episode. The confidence intervals over the course of training (at each episode calculated using 50 training exercises) show that both agents do not converge to a single liquidity choice. This is seemingly due to the nonstationarity in the environment. For instance, the total cost associated with the selected action from agent A could sometimes be influenced by the choice of agent B and vice versa. This could cause uncertainty of payment arrivals in all twelve periods and hence influence the agent's learning ability.

Figure 6 shows this more clearly. The figure shows the average and range of agents' initial liquidity decisions at various points of the training. At the start of training, both agents chose initial liquidity around 1/2. As training progresses, both agents reduce their liquidity

choice with A showing the largest reduction. Agent B, which has higher payment demands, converged to a higher liquidity choice than agent A. Also, neither agent converged to a single solution by the end of the training, likely due to the uncertainty of payment arrivals from the other participant in each intraday period.

Note that for this more general case with twelve intraday periods, the learning is slower compared to the two-period case in terms of training costs. Also, the confidence interval in the twelve-period case is wider compared to the two-period case. These findings demonstrate the complexity involved in generalization of learning with higher intraday periods (see Figures 4 and 5).[14] Also, remember that for the twelve-periods case, we do not have the analytical solution of the optimal initial liquidity choice and therefore we cannot know what is the minimum equilibrium cost.[15]

Figure 5: Evolution of the cost over the training of the initial liquidity choice problem. These learning curves show the cost incurred during training and testing for agent A (left) and agent B (right). The solid lines are the average cost at each point in the training computed using 50 independent training exercises, and the shaded areas are the 99% confidence interval bands.

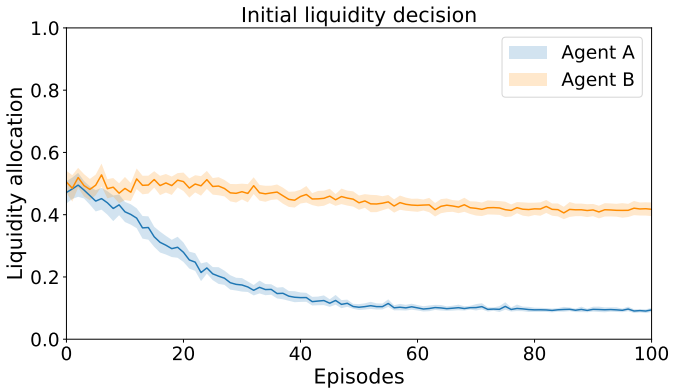


A key parameter in the environment is the delay cost that banks face, which is unobserved by policymakers. Therefore, we examine the sensitivity of our results to the choice of delay cost, r_d. We performed the robustness exercise varying the cost of delay ranging from 0.01 to 0.3. Recall that in previous exercises $r_c = .1$, $r_d = .2$ and $r_b = .4$. Figure 7 show the box plots of the total cost of each agent at the end of the 50 training episodes for the different levels of delay cost. When we vary the delay cost above the initial liquidity cost, i.e., for values above 0.1, agents increase their initial liquidity allocation, incurring a larger total training cost, in order to minimize the delay cost. This sensitivity is, however, relatively small. In contrast, the total cost of agent A is very sensitive when delay cost is below 0.1. Agent A is the one with lower payment demands, therefore when delay cost is low enough, it learns to wait for

[14] Training convergence can be increased by using a neural network with hidden layers; see Figure 12 in Appendix F.1 where we perform a robustness study with different network sizes.

[15] To tackle the problem of benchmarking our results in this case, we conducted a brute force computation of the cost resulting from all possible combinations of initial liquidity choices and payment profiles for both agents. Results are shown in Appendix C.

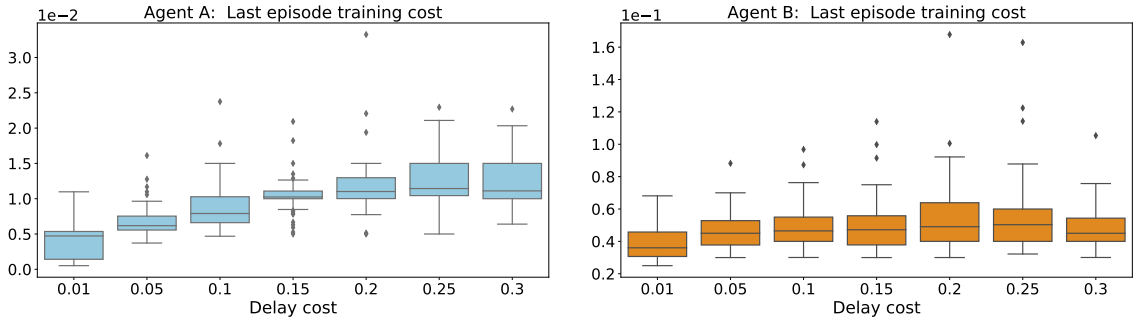
Figure 6: Evolution over the training process of the initial liquidity choice, x_0. Each solid line is the average choice over 50 training exercises. Shaded areas are the 99% confidence interval bands.



the incoming payments from B, reducing its initial liquidity allocation and its total cost.

The case where $r_d \leq r_c$ is interesting because the best response function of the agents is no longer the one presented in Section 4, even in cases with only $T = 2$. With this configuration of parameters, the agents might have the incentive to allocate less initial liquidity because if the value of incoming payments is large enough, the reduction of initial liquidity cost could compensate for the delay costs.

Figure 7: Boxplots showing the sensitivity of final costs after training across different selections of delay cost. Whiskers are defined as 1.5 · IQR.



7 Conclusion

This paper employs RL to estimate the policy rules of two banks participating in a high-value payments system. Our results show that—in a simplified learning problem for which we know the optimal solution—policy rules trained with the RL algorithm converge to the optimal solution. Using the intuition and confidence in the training algorithm, we generalize to a problem where agents have to simultaneously learn a policy to choose their initial liquidity. Both

agents learn to choose the initial liquidity that minimizes their cost of processing all payments. We performed a large number of robustness checks on the environment inputs (the payment demand profiles, the choice of agents and cost parameters) and training hyperparameters. These results show the applicability of RL to estimate best-response functions in real-world strategic games.

The significance of these results are underlined by the fact that the initial liquidity decision of banks is complex even when agents have full knowledge of the environment, the data generating process and their opponent's strategies. Our agents had no knowledge of the environment, the data, or the game itself. From a normative point of view, if agents trained with RL methods can find alternative solutions, regulators could use RL policies to help in their mandates of ensuring the safety and efficiency of these systems. Also, understanding these techniques will be relevant for policymakers as banks themselves are likely to attempt to use these algorithms for their liquidity management.

Before these promises can be realized, we need a more complete understanding of the liquidity management rules of financial institutions. This paper is the first in that agenda. Future work will involve pursuing the following extensions. First, it is important to explore how the agents would perform if they were learning the initial liquidity and intraday payment problems at the same time. Second extension is to tackle the problem with more than two-agents, which will require generalizing over more complex sets of payment demands to potentially derive richer policies that account for who a payment is being sent to and who a payment has been received from. A major extension will be to introduce some realistic features of the payments system, like non-divisible or urgent payments, and an intraday market for liquidity. These could be significant departures to the learning setup that might require new algorithms and neural networks.

References

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. (2016). TensorFlow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283.

Arciero, L., C. Biancotti, L. d'Aurizio, and C. Impenna (2008). Exploring agent-based methods for the analysis of payment systems: A crisis model for StarLogo TNG. *Bank of Italy Temi di Discussione (Working Paper) No 686*.

Bech, M. L. and R. Garratt (2003). The intraday liquidity management game. *Journal of Economic Theory 109*(2), 198–219.

Bowling, M. (2003). *Multiagent learning in the presence of agents with limitations*. Ph. D. thesis, Carnegie Mellon University.

Bulusu, N. and P. Guérin (2019). What drives interbank loans? Evidence from Canada. *Journal of Banking & Finance 106*, 427–444.

Calvano, E., G. Calzolari, V. Denicolò, and S. Pastorello (2019). Artificial intelligence, algorithmic pricing and collusion. *CEPR Discussion Paper* (DP13405).

Calvano, E., G. Calzolari, V. Denicolò, and S. Pastorello (2020, October). Artificial intelligence, algorithmic pricing, and collusion. *American Economic Review 110*(10), 3267–97.

Foerster, J., I. A. Assael, N. de Freitas, and S. Whiteson (2016). Learning to communicate with deep multi-agent reinforcement learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 29*, pp. 2137–2145. Curran Associates, Inc.

Fudenberg, D. and D. K. Levine (2016). Whither game theory? Towards a theory of learning in games. *Journal of Economic Perspectives 30*(4), 151–170.

Funai, N. (2019). Convergence results on stochastic adaptive learning. *Economic Theory 68*(4), 907–934.

Galbiati, M. and K. Soramäki (2011). An agent-based model of payment systems. *Journal of Economic Dynamics and Control 35*(6), 859–875.

Garratt, R. (2019). An application of Shapley value cost allocation to liquidity savings mechanisms. Technical Report 2019-26, Bank of Canada Staff Working Paper.

Igami, M. (2020). Artificial intelligence as structural estimation: Deep blue, bonanza, and alphago. *The Econometrics Journal*.

McAndrews, J. and S. Rajan (2000). The timing and funding of fedwire funds transfers. *Federal Reserve Bank of New York Economic Policy Review 6*, 17–32.

Rivadeneyra, F. and N. Zhang (2020). Liquidity usage and payment delay estimates of the new Canadian high value payments system. Technical report, Bank of Canada Discussion Paper.

Roth, A. E. and I. Erev (1995). Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior 8*(1), 164–212.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT Press.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8*(3-4), 229–256.

A Intraday Payments Policy

In this appendix we describe the learning setup of the intraday payments policy and verify that the RL agents learn the expected policy.

A.1 Learning setup

Here we describe the learning setup for the intraday payments policy:

- **Agents:** Agents A and B are trained simultaneously over 50 episodes.
- **Intraday periods:** Each episode is divided into twelve intraday periods, $T = 12$.
- **Payments demand:** For the payments demand profile over the twelve intraday periods we use data from the Canadian LVTS. See Figure 3. During every episode in the training, the agent confronts a payment profile drawn from these data. The agent does not know the payment demand of a given intraday period until that period has arrived. We denote p_t for the new payment demand in period t, and P_t for the accumulated payment demand in period t, the latter including previously delayed payment demands.
- **Initial liquidity:** At the beginning of every episode, the agent is provided enough initial liquidity (ℓ_0) to settle the total value of the episode's payments demand (i.e., $\ell_0 > \sum_{t=1}^{T} P_t$). This means that regardless of the choice of the other agent, the training agent would have enough liquidity to satisfy its demands.
- **Initial Liquidity Cost:** There is no cost associated with the initial liquidity allocation (i.e., $r_c = 0$).
- **State space:** At every intraday period t, the agent observes the following state space: the current period t, the current period payments demand p_t, accumulated payments demand P_t (including previously delayed payment demands) and the current period liquidity ℓ_t:

  $$s_t = (t, p_t, P_t, \ell_t).$$

  In the state, we provide the agent with new and previously delayed payments separately, which will help in the training. An alternative would have been to provide only the total accumulated payment demand.
- **Action space:** The action space is the fraction x_t of total payments demand P_t to be sent at time t. We discretize x_t, the unit interval, into 21 evenly-spaced fractions: $\{0,\ 0.05,\ 0.1,\ ...,\ 1\}$.

- **Cost:** The total processing cost per episode is:

$$\mathcal{R} = \sum_{t=1}^{T-1} P_t(1 - x_t) \cdot r_d,$$

  where r_d is the intraday delay cost and P_t is per-period payment demand, which includes any delayed payments from the previous intraday periods.

- **End-of-day:** Since each agent starts with sufficient initial liquidity to settle all payments even if it does not receive any payments from the other agent, there will be no end-of-day borrowing from the central bank and costs will be zero.

In this environment, it is straightforward to verify using $\mathcal{R}$ that the optimal choice is $x_t = 1$, $\forall t$. The intuition for this result is that given a level of initial liquidity at no cost, there is no incentive to delay the payments. Therefore, the agent minimizes the cost of processing payments by sending all the payment demands in each and every period, thus avoiding the cost of delay. Correspondingly, this choice minimizes the cost.

Similar to the initial liquidity decision case, as payment demands we use the data described in subsection 6.2. For training we use the REINFORCE algorithm. We approximate the policy function, π, using a linear feed-forward neural network. The learning setup is implemented in Python using TensorFlow. Additional details on the learning setup, implementation and model parameters selection are discussed in Appendix D. Note that in Appendix F.5 we show two other pairs of participants.

A.2 Results for the intraday payments decision x_t

We present the results by showing the intraday payment choices and cost incurred by agents over the training process. Figure 8 shows the evolution of the average cost, $\mathcal{R}$, incurred by each agent over 50 training episodes. Figure 9 shows the evolution, over the training process, of the average payment choice, action x_t, across samples in each episode. We present the choices separately for the first four intraday periods, $t = 1, 2, 3, 4$. We also show the entire range of choices observed across the training exercises, and the optimal choice, $x_t = 1$, in red.

These results show that both agents quickly learn to reduce the average cost and closely converge to the optimal behaviour of sending all payments in each intraday period. Recall that the parametrized policy function has to learn to make 11 choices, one for each of the intraday periods. Note that in the first episodes of the training, agents, on average, choose to send roughly half of the payments in each of the intraday periods we display. This is due to random initialization of the network parameters. Agent B, which has a higher payments demand, starts at a higher average cost compared to that of agent A. Also, across the training exercises, the range of cost for agent B is wider than for agent A. These plots suggest that,

although the rate of learning can vary across the two agents, on average they both converge close to the optimal behaviour, i.e., $x_t = 1$.

Note that from intraday period $t = 5$ onwards, there is no difference in the shape of the learning curves as both agents quickly converge to optimal in about the same number of training episodes; therefore we omit them.

Figure 8: Evolution of the cost over the training process of the intraday payment decision. These learning curves show the cost incurred during training and testing for agent A (left) and agent B (right). Only 25 training episodes are shown. The solid lines are the average cost at each point in the training, computed using 50 independent training exercises, and shaded areas are the 99% confidence interval bands..

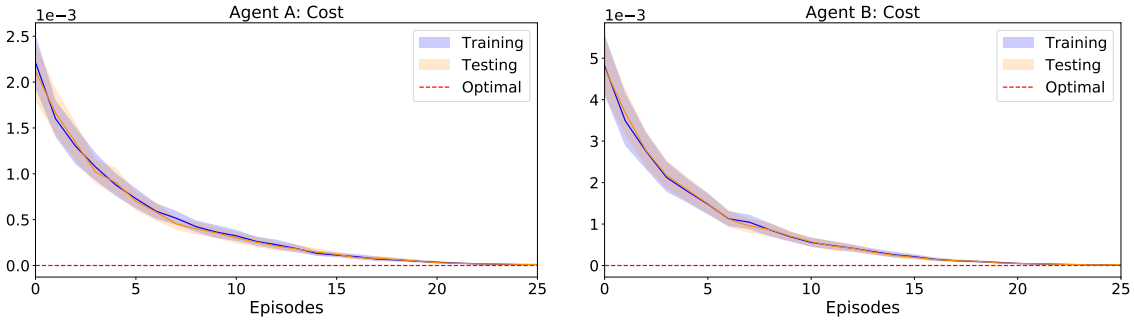
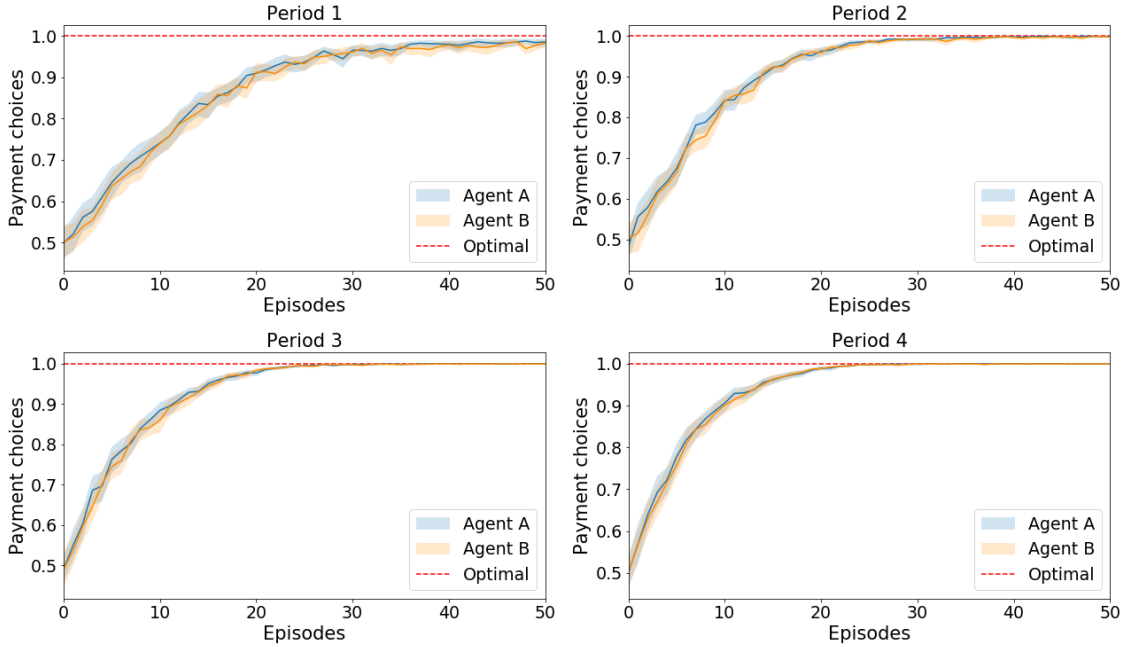


Figure 9: Evolution over the training process of the intraday payment choices, x_t, at $t = 1, 2, 3, 4$. Each solid line is the average choice over 50 training exercises. Shaded areas are the 99% confidence interval bands. From intraday period $t = 5$ onwards, learning curves converge at roughly the same rate.



B Reinforcement Learning

In computer science, reinforcement learning methods are used for learning how to act (near) optimally in sequential decision-making problems in uncertain environments. In episodic tasks, reinforcement learning problems typically involve a single agent (the learner) interacting with an environment, formally defined as a finite Markov decision process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H \rangle$, where $\mathcal{S}$ is a finite set of *states*; $\mathcal{A}$ is a finite set of *actions*; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ encodes the next-state transition dynamics, where $\mathcal{T}(s, a)(s')$ is the probability of ending in state s' after performing action a from state s; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [R_{min}, R_{max}] \subset \mathbb{R}$ is the bounded *reinforcement* function, providing a positive or negative signal that the agent can leverage to improve its behaviour; and H is the horizon length. An agent's *behaviour* is represented as a policy $\pi : \mathcal{S} \to \Delta\mathcal{A}$, mapping states to a distribution over actions. Let Π be the set of all policies.

Each policy induces a Markov chain, and we can quantify its *value* from state $s_0 \in \mathcal{S}$ via the following equation:

$$V^{\pi}(s_0) = \sum_{t=0}^{H} \mathbb{E}_{a_t \sim \pi(s_t), s_{t+1} \sim \mathcal{P}(s_t, a_t)} R(s_t, a_t)$$

This can also be expressed via the following set of recurrent equations, known as the Bellman equations:

$$\begin{aligned} V_0^{\pi}(s) &= 0 \\ V_H^{\pi}(s) &= \mathbb{E}_{a \sim \pi(s)} \left[\mathcal{R}(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(s,a)} V_{H-1}^{\pi}(s') \right] \end{aligned} \tag{3}$$

When the system dynamics (e.g., $\mathcal{P}$ and $\mathcal{R}$) are known, the system of equations above can be solved via dynamic or linear programming.

We are typically interested in finding the *optimal* policy $\pi^* := \arg\max_{\pi \in \Pi} V^{\pi}$, which satisfies the Bellman optimality equations, where for succinctness we write V^* instead of V^{π^*}:

$$\begin{aligned} V_0^{*}(s) &= 0 \\ V_H^{*}(s) &= \max_{a \in \mathcal{A}} \left[\mathcal{R}(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(s,a)} V_{H-1}^{*}(s') \right] \end{aligned} \tag{4}$$

In addition to V^{π}, it is often useful to maintain a state-action function Q^{π}, defined as follows:

$$Q^{\pi}(s, a) = \mathcal{R}(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(s,a)} V^{\pi}(s')$$

Access to $Q^* := Q^{\pi^*}$ immediately yields the optimal policy $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$.

When the system dynamics are known (e.g., $\mathcal{P}$ and $\mathcal{R}$), both equations 3 and 4 can be

solved via dynamic or linear programming. However, in reinforcement learning we typically assume the environment dynamics are unknown, and the value functions are estimated by samples obtained from interacting with the environment. One natural approach is to use Monte Carlo methods, where a set of trajectories are collected, and then for each $s \in \mathcal{S}$, $V^\pi(s)$ is estimated by averaging over all trajectories that passed through s (see Sutton and Barto 2018 for more details).

A more promising approach is one that combines ideas from Monte Carlo estimation and dynamic programming: temporal-difference learning (TD-learning). Like Monte Carlo methods, TD-learning updates its estimates from sampled experiences; but like dynamic programming methods, it does so with *single-step* transitions. In its simplest form, after performing action a from state s and observing reward r and next-state s', TD-learning updates its estimate of $V^\pi(s)$ with the rule:

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$$

where α is known as the *step size* and quantifies how aggressively to update V when a new experience is received.

There are a number of learning methods that build on the TD update, and the interested reader is referred to Sutton and Barto 2018 for a comprehensive overview. We will focus on *policy gradient* methods, which updates a policy π_θ parameterized by a weight vector θ. For example, the policies may be encoded as neural networks and θ represents the network's weights, which is the approach we take in this paper. Under this paradigm, one can refer to the *performance* of θ from an initial state $s_0 \in \mathcal{S}$ as:

$$J(\theta) := V^{\pi_\theta}(s_0)$$

Under this formulation, updating the parameters so as to increase the expected future value can be done by updating θ using an estimate $\widehat{\nabla J(\theta)}$ of the true *gradient* of the return function: $\nabla J(\theta)$:

$$\theta \leftarrow \theta + \alpha \widehat{\nabla J(\theta)}$$

where α is once again the step size. The *policy gradient theorem* (see Sutton and Barto 2018, Section 13.2) states that:

$$\nabla J(\theta) \propto \sum_{s \in \mathcal{S}} \mu_\theta(s) \sum_{a \in \mathcal{A}} Q^{\pi_\theta}(s,a) \nabla \pi_\theta(s)(a) \tag{5}$$

where μ_θ is the on-policy distribution relative to π_θ[16] (see Sutton and Barto 2018, Section 9.2 for details). Note that this requires that the parameterized policy π_θ is differentiable, which is typically the case when using a softmax activation from the output of a neural network.

[16] Intuitively, this is the fraction of time the agent expects to spend in a state s when following policy π_θ.

Specifically, if our network outputs a value estimate $\hat{Q}_\theta(s,a) \approx Q(s,a)$, then the softmax action selection would be:

$$\pi_\theta(s)(a) = \frac{e^{\hat{Q}_\theta(s,a)}}{\sum_{b\in\mathcal{A}} e^{\hat{Q}_\theta(s,b)}}$$

A method for estimating the gradient in Equation 5 is to use Monte Carlo estimation, as is done in the classic REINFORCE algorithm (Williams 1992). Let $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ denote the state and action performed at time step t, and define G_k as the total return obtained from time step k until the end of the episode:

$$G_t = \sum_{t=k}^{H} \mathcal{R}(s_t, a_t)$$

Derived from the policy gradient theorem (5), the REINFORCE update is defined as follows, after choosing action a_t from state s_t in time step t and observing total cost G_t afterwards:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi_\theta(s_t)(a_t)}{\pi_\theta(s_t)(a_t)}$$

B.1 Multi-agent reinforcement learning

Multi-agent reinforcement learning is a growing area of research but is still a relatively new field. The complex dynamics induced by multiple agents interacting with each other make theses types of problems difficult to analyze and often require game-theoretic techniques for reasoning about observed behaviours.

In Section 3 we introduced the real-time gross settlement (RTGS) payments system as an environment involving multiple independent agents, so it can be considered as a multi-agent reinforcement learning problem. However, our framing is different from the standard framing in the multi-agent reinforcement learning literature, where it is typically expressed via an MDP $\langle \mathcal{S}, \mathcal{A}^n, \mathcal{T}, \mathcal{R}, H \rangle$, where $\mathcal{A}^n = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$. Here, n agents observe the global state, but each agent independently selects their action. After each agent has made an action selection a_i from state s, the next-state transition dynamics are defined by $\mathcal{T}(s, (a_1, a_2, \cdots, a_n))$ and the reward dynamics by $\mathcal{R}(s, (a_1, a_2, \cdots, a_n))$. The interested reader is referred to Bowling 2003 and Foerster et al. 2016.

By contrast, our setting consists of n independent agents that do not get to observe a shared global state. Formally, our setup is an MDP $\langle \mathcal{S}^n, \mathcal{A}^n, \mathcal{T}, \mathcal{R}, H \rangle$, where $\mathcal{S}^n = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ and $\mathcal{A}^n = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$. Here, each agent i observes a state s_i and selects an action a_i. Importantly, there may or may not be overlap between $\mathcal{S}_i$ and $\mathcal{S}_j$ for $i \neq j$; that is, the states observed by agent i may contain no information about the states observed by the other agents. The transitions are governed by $\mathcal{T}((s_1, s_2, \cdots, s_n), (a_1, a_2, \cdots, a_n))$ and the rewards by $\mathcal{R}((s_1, s_2, \cdots, s_n), (a_1, a_2, \cdots, a_n))$, which means that even though the agents may not

be able to observe each other's states, their actions *do* affect each other's transitions and rewards/costs. This is a form of partial observability in a multi-agent setting, and the difficulty each agent faces is *independently* dealing with this partial observability.

C Benchmark for the Initial Liquidity Choice Using Brute Force Search

The results of Section 4 gave us benchmark results for the simplified problem with only two intraday periods. The closed form solution is not available for $T \geq 3$, therefore, to validate the training results of the problem with twelve intraday periods, we computed an empirical benchmark. Our benchmark is the planner solution that minimizes the sum of initial liquidity costs for both agents. With the current setup of two agents, twelve intraday periods, 380 payment profiles and the assumption of the optimal intraday payments policy, it is still feasible to calculate via brute force the sum of the initial liquidity costs for both agents in all possible outcomes. This brute force approach would not be feasible as soon as we add more agents and payment profiles.

We ran simulations to compute all feasible combinations of initial liquidity choices and payment profiles for both agents and then calculated the cost of each combination. Among these combinations, the planner's choice of initial liquidity is the pair of liquidity choices that yielded the minimum total cost for both agents A and B.

More formally, let $\mathcal{R}^i(\ell_0^i, \ell_0^{-i}, P_j)$ be the initial liquidity cost for agent $i \in \{A, B\}$ given the liquidity choice pair $(\ell_0^i, \ell_0^{-i}) \in \mathcal{L} \times \mathcal{L}$, and a given daily payment profile $P_j \in \mathcal{P}$. For a given payment profile P_j, the minimum system-wide total cost of initial liquidity is:

$$\mathcal{R}^*(P_j) = \min_{(\ell_0^A, \ell_0^B) \in \mathcal{L} \times \mathcal{L}} \mathcal{R}^A(\ell_0^A, \ell_0^B, P_j) + \mathcal{R}^B(\ell_0^B, \ell_0^A, P_j). \tag{6}$$

Then we define as our benchmark the average minimum system-wide total cost of initial liquidity:

$$\frac{1}{|\mathcal{P}|} \sum_{P_j \in \mathcal{P}} \Big(\mathcal{R}^*(P_j) \Big). \tag{7}$$

Correspondingly, the minimum and maximum cost across all payment profiles are:

$$\min_{P_j \in \mathcal{P}} \Big(\mathcal{R}^*(P_j) \Big), \tag{8}$$

and

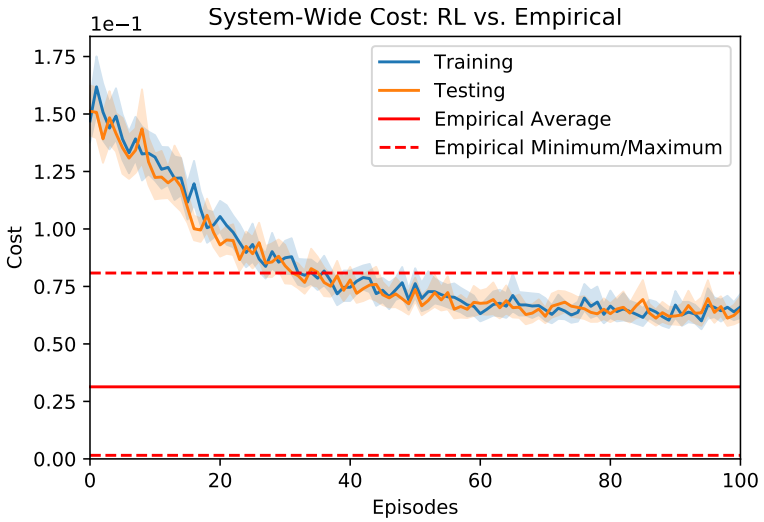
$$\max_{P_j \in \mathcal{P}} \left(\mathcal{R}^*(P_j) \right), \tag{9}$$

respectively.

We report these findings in Figure 10. The solid line is the average and the dotted red lines are the minimum and maximum possible total costs across all payment profiles. For the agents, the plot here shows the sum of the individual costs presented in Figure 5 to make a comparison at the system level.

Note that to treat Equation 7 as our benchmark, we implicitly assume the existence of a planner that implements the allocation of initial liquidity that minimizes total costs or some form of cooperation between the parties, for example via side payments to allow them to share the costs (Garratt 2019). Treated as a cooperative solution, if the sum of both agents costs are above the average costs, there would still be room for better learning performance, albeit one that might be hard to achieve when both agents learn independently.[17]

Figure 10: Evolution of total costs over the training of the initial liquidity choice problem with empirically calculated bounds. Curves show the sum of the costs incurred by both agents (independently) during training and testing. These are the sum of the costs shown in Figure 5. The solid red line is the average across payment profiles of the minimum system-wide cost for each payment profile. Dotted red lines are the minimum and maximum among all payment profiles.



[17] See Calvano et al. (2019) for an example of two reinforcement learning agents colluding to price above their marginal cost in an oligopoly model of price competition. The authors report that to achieve this result, training requires on average 500,000 episodes.

D Training Hyperparameters

In this appendix we provide further details on problem setup and hyperparameters used for training along with the pseudocode. Table 2 lists environment settings and hyperparameters used in training intraday payments and initial liquidity agents. These are base parameters selected for each of the training exercises unless otherwise specified.

Table 2: Environment and agents parameters selected for training

Parameters	**Intraday payments problem**	**Initial liquidity problem**
Algorithm	REINFORCE	REINFORCE
Number of agents	2	2
Number of episodes	50	100
Periods in each episode	12	12
Batch size	10	10
State space	4	12
Action space	21	21
Initial liquidity cost	0	0.1
Per-period delay cost	0.2	0.2
End-of-day borrowing cost	∞	0.4
Learning rate	0.1	0.1
Activation function	tanh	tanh
Optimization method	-	Adam
Policy network	Linear network	Linear/1-hidden layer network
Initial liquidity	Unrestricted collateral	Learning parameter
Intraday payments	Learning parameter	Send as much as possible

E Pseudocode

We provide the following pseudocode: Algorithm 1 for fixed payments policy, Algorithm 7 for RL agents payments policy, Algorithm 3 for fixed liquidity policy, Algorithm 4 for RL agents liquidity policy, Algorithm 5 for training intraday payments policy, Algorithm 6 for training initial liquidity policy and Algorithm 8 for brute force search method used for validation. We have chosen to separate out the pseudocode in this fashion to highlight the similarities and differences of each problem setup; in future work we will aim to use reinforcement learning to solve both problems simultaneously.

Algorithm 1: π_{fixedP}: Pseudocode for Fixed Payments Policy

input : $s_t^i = (t, p_t^i, P_t^i, \ell_t^i)$
output : x_t^i
initialization ;
if *last period* **then**
  send $x_T^i = P_T^i$ and borrow $P_T^i - \ell_T^i$ at cost r_b if $P_T^i > l_T^i$;
else
  send $x^{i'_t}$, the highest possible payment given liquidity up to l_t^i
end

Algorithm 2: $\pi_{\theta P}$: Pseudocode for RL Payments Policy

input : $s_t^i = (t, p_t^i, P_t^i, \ell_t^i)$
output : x_t^i
initialization ;
if *last period* **then**
  send $x_T^i = P_T^i$ and borrow $P_T^i - l_T^i$ at cost r_b if $P_T^i > \ell_T^i$;
else
  $x_t^i \sim \pi_\theta^i(s_t^i)$
  send $x^{i'_t}$, the highest possible payment given liquidity up to $\min(x_t^i, \ell_t^i)$
end

Algorithm 3: π_{fixedL}: Pseudocode for Fixed Liquidity Policy

input : $s_0^i = (p_0^i, ..., p_{12}^i)$
output : ℓ_0^i
initialization ;
allocate enough liquidity to clear all payments; in our case, because of normalization, this is always an $\ell_0^i = 1$

Algorithm 4: $\pi_{\theta L}$: Pseudocode for RL Liquidity Policy

input : $s_0^i = (p_0^i, ..., p_{12}^i)$

output : ℓ_0^i

initialization ;

choose liquidity $\ell_0^i \sim \pi_\theta^i(s_0^i)$

Algorithm 5: Pseudocode for Intraday Payments Problem

Result: Trained Policy Network

initialization ;

for *each episode* **do**

  set liquidity using $\pi^i_{fixedL}(s_0^i)$ for each agent i;

  for *each batch period* **do**

    send intraday payments concurrently using $\pi^i_{\theta P}(s_t^i)$ for each agent i

  end

  Perform batch gradient update to θ from experience collected using $\pi^i_{\theta P}$ after each batch ;

end

Algorithm 6: Pseudocode for Initial Liquidity Problem

Result: Trained Policy Network

initialization ;

for *each episode* **do**

  set liquidity using $\pi^i_{\theta L}(s_0^i)$ for each agent i;

  for *each batch period* **do**

    send intraday payments concurrently using $\pi^i_{fixedP}(s_t^i)$ for each agent i

  end

  Perform batch gradient update to θ from experience collected using $\pi^i_{\theta L}$ after each batch ;

end

Algorithm 7: $\pi_{\theta P}$: Pseudocode for RL Payments Policy

input : $s_t^i = (t, p_t^i, P_t^i, \ell_t^i)$

output : x_t^i

initialization ;

if *last period* **then**

  send $x_T^i = P_T^i$ and borrow $P_T^i - l_T^i$ at cost r_b if $P_T^i > \ell_T^i$;

else

  $x_t^i \sim \pi_\theta^i(s_t^i)$

  send $x^{i'_t}$, the highest possible payment given liquidity up to $\min(x_t^i, \ell_t^i)$

end

Algorithm 8: Pseudocode for Loop Structure in Intraday Payments Problem

Result: Robustness Visualization

for *each sample (independent training exercise)* **do**

  random initialize agents policy $\pi_{\theta P}^i$

  for *each episode (day)* **do**

    generate multiple trajectories (batch)

    for *each trajectory* **do**

      randomly sample new profile from the pool for each agent i

      set liquidity using $\pi_{fixedL}^i(s_0^i)$

      for *each period* **do**

        send intraday payments concurrently using $\pi_{\theta P}^i(s_t^i)$ for each agent i

        record each period experience: state, action and per-period cost

      end

      record each trajectory experience: state, action, cost

    end

    record batch experience: state, action, cost

    perform batch gradient update to θ using batch experience collected under $\pi_{\theta P}^i$

    record batch training loss and cost

  end

  record episode training loss and cost (averaged over batch)

end

plot episode training loss and cost across all samples (to get mean and CI)

Algorithm 9: Pseudocode for Loop Structure in Liquidity Problem

Result: Robustness Visualization

for *each sample (independent training exercise)* **do**
  random initialize agents policy $\pi^i_{\theta L}$
  for *each episode (day)* **do**
    generate multiple trajectories (batch)
    for *each trajectory* **do**
      randomly sample new profile from the pool for each agent i
      set liquidity using $\pi^i_{\theta L}(s^i_0)$ for each agent i
      for *each period* **do**
        send intraday payments concurrently using $\pi^i_{fixedP}(s^i_t)$ for each agent i
      end
      record each trajectory experience: state, action, cost
    end
    record batch experience: state, action, cost
    perform batch gradient update to θ using batch experience collected under $\pi^i_{\theta P}$
    record batch training loss and cost
  end
  record episode training loss and cost (averaged over batch)
end
plot episode training loss and cost across all samples (to get mean and CI)

F Robustness Exercises

Here we present the results of several robustness exercises (known as ablation studies in the machine learning literature). We study how changes to the hyperparameters of the system and the RL algorithm affect the learning performance. In general, the learning algorithm converges in every exercise we attempted, but in some cases, depending on the hyperparameters, the final policy might be different across exercises. More specifically, for the intraday payments problem, regardless of the hyperparameters chosen, the algorithm converges to the optimal policy at different speeds and variance of behaviour. On the other hand, for the initial liquidity problem, the algorithm converges to different policies depending on the hyperparameters chosen. Note that the training hyperparameters from Table 2 are kept constant across experiments unless otherwise specified.

F.1 Network size

We start with variants of the network size and structure. In general, for both the initial liquidity and the intraday payments problems, we found that increasing the network size tended to speed up the agents' learning, leading to faster convergence, all else being equal. For the intraday payments problem, Figure 11 shows that although a linear network is sufficient to find the optimal solution, we did not observe a problem of over-fitting when increasing the number of hidden layers. This may be due to the fact that the optimal strategy for this game is simply to send everything in any intraday period.

On the other hand, for the initial liquidity problem, Figure 12 shows that a linear network had some difficulty—larger number of episodes and larger variance during training—in converging to the optimal solution. Adding one hidden layer allowed us to converge significantly faster to the optimal solution, however we observed diminishing benefits to increasing the network size.

F.2 Learning rate

The robustness exercises of the learning rate (α) showed mixed results. For the intraday payments problem, an increase in the learning rate from 0.01 to 0.1 and 1 led to a faster convergence. The relationship between higher learning cost with faster convergence and narrower confidence intervals is shown in Figure 13 and 14. This is not too surprising given the simplicity of the rule that needs to be learned in this problem.

In contrast, for the initial liquidity problem, an increase in the learning rate led to a slower convergence as well as higher variance. This is likely due to the more complicated optimal strategy for the initial liquidity problem. With the highest learning rate, $\alpha = 1$, in every episode agents incorporate all the information provided by the gradient but eventually do not

Figure 11: Robustness to the variation of the network size for the intraday payments problem: Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the number of hidden layers in the neural network.

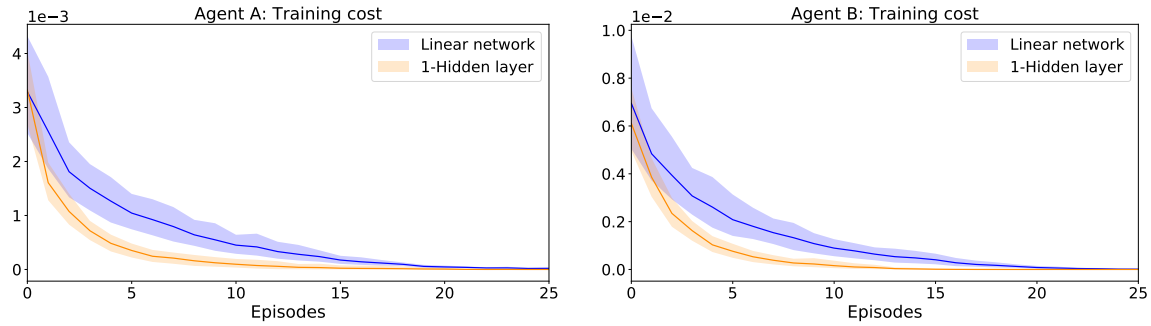
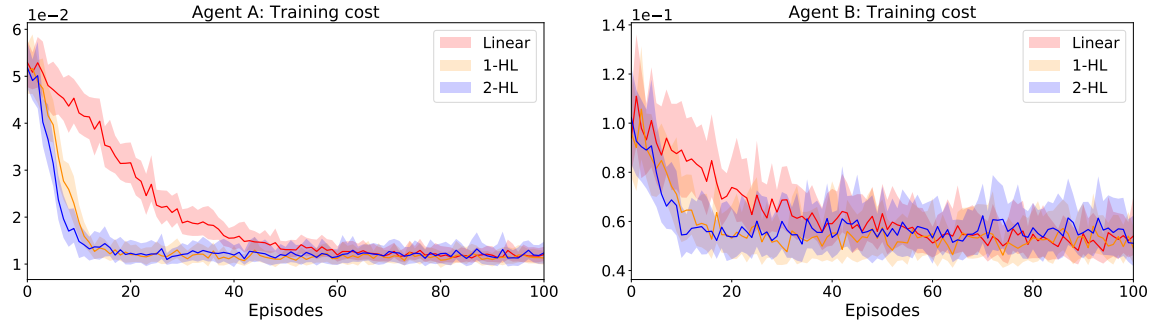


Figure 12: Robustness to the variation of the network size for the initial liquidity problem: Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the number of hidden layers in the neural network.



explore new policies. Consequently, both agents converge very quickly but to a high liquidity cost compared to the other lower learning rates.

F.3 Batch size

Similar intuitive results were found when we studied the effects of batch size on learning. Unsurprisingly, a higher batch size results in lower variance with diminishing returns as batch size increases. In some trials where batch size was singular, the learning fails to converge to the optimal behaviour entirely. A larger batch size also tended to lead to faster convergence. This is likely due to more exploration enabled by the larger batch sizes for the stochastic attribute of the reinforcement learning algorithm. See Figures 15 and 16.

Figure 13: Variation of the learning cost in the intraday payments problem: Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the learning rate in the learning algorithm.

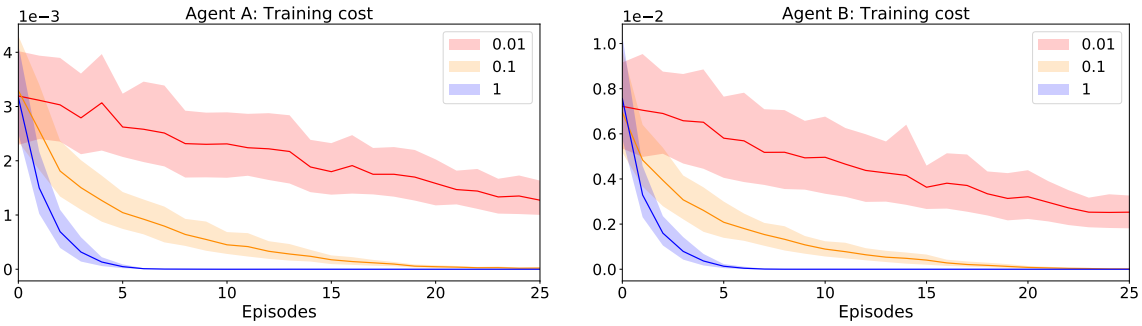


Figure 14: Variation of the learning cost in the initial liquidity problem: Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the learning rate in the learning algorithm.

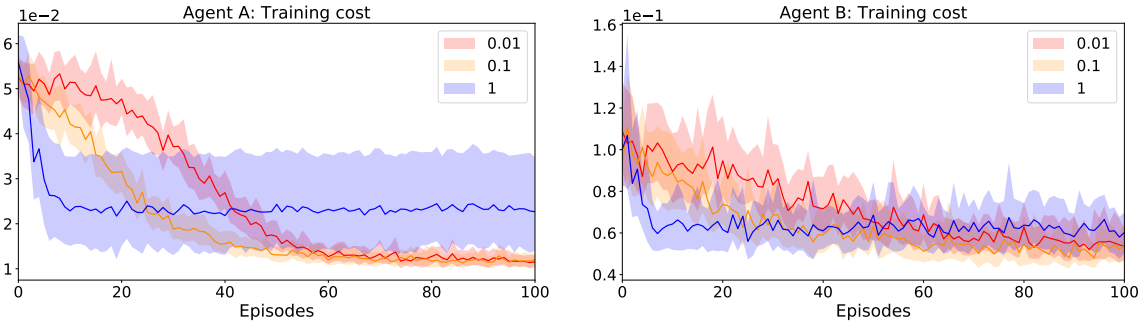


Figure 15: Batch size robustness for the intraday payments problem. Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the training batch size.

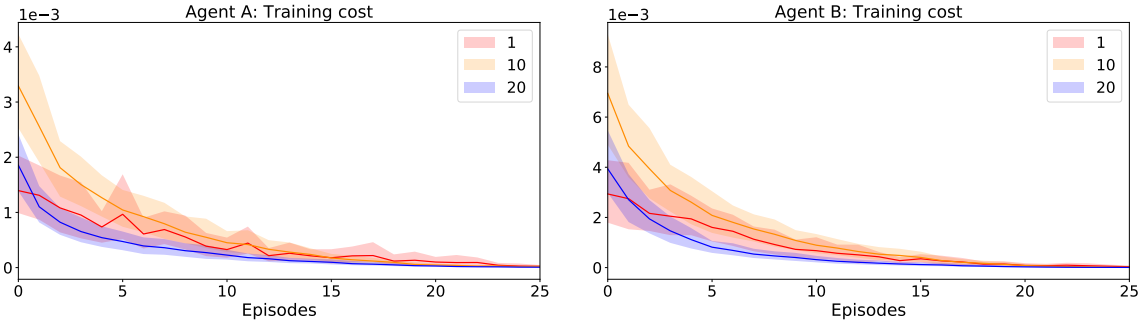
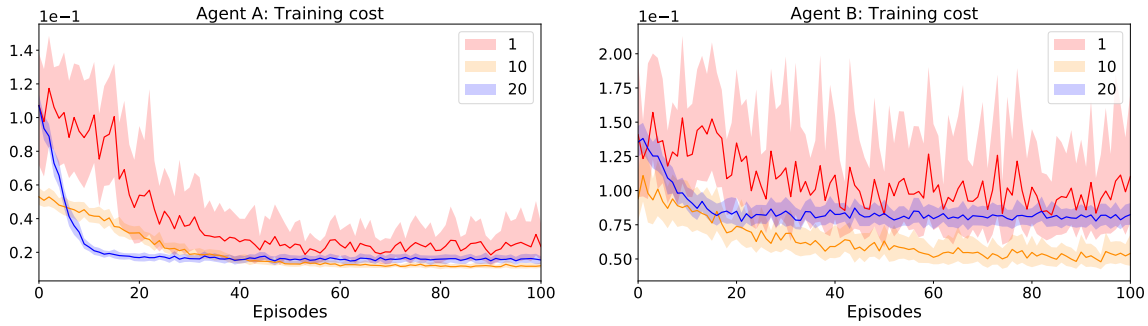


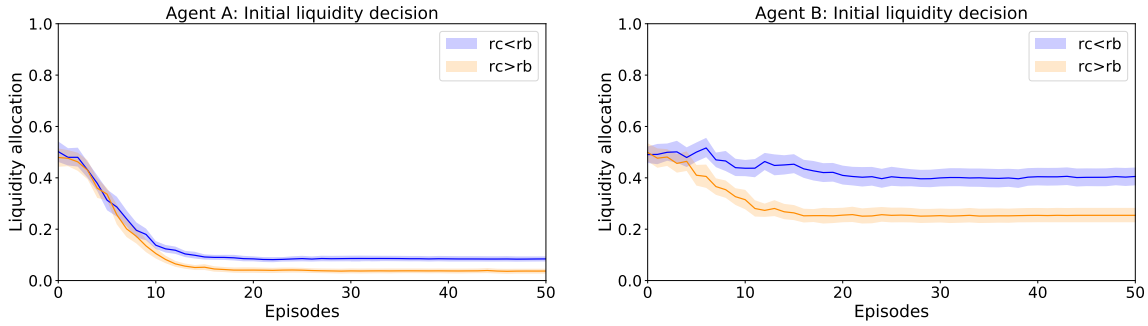
Figure 16: Batch size robustness for the initial liquidity problem. Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the training batch size.



F.4 Cost inequality relationship

In the exercises in the main body of the paper we assumed $r_c < r_d < r_b$, the per-dollar cost of initial liquidity, delay and end-of-day borrowing, respectively. Here we present an exercise reversing the relationship between the cost of borrowing and the cost of initial liquidity, $r_c = .4$ and $r_b = .1$. We found, not surprisingly, that agents allocated less liquidity when it was more beneficial to borrow at the end of the day, and that agents allocated more liquidity when borrowing was expensive. There is always some liquidity allocated regardless because there is still a cost associated with delaying payments, thus the agents could not solely rely on waiting for the opposing agent to send liquidity to them first as we fixed their intraday payments behaviour to send all payments in each period. See Figure 17.

Figure 17: Robustness of the parameter choices of the costs in the initial liquidity problem. Learning curves showing the cost incurred during the training for both agents. Solid lines are the average cost for the 50 independent trial runs with the same parameters. Shaded areas are the 99% confidence interval. Hyperparameters are consistent with Table 2, with the exception of the inequality relationship between the cost of borrowing and the cost of initial liquidity.



F.5 Payment Profiles

We performed additional robustness exercises to study the behaviour of agents that have different payment profiles. The payment profiles for two other pairs of banks are summarized in Figure 18. The results of the training are shown in Figure 19. To contrast with banks A and B examined in the main body, were chose other pairs to show banks with diverging flows late in the day (labelled C and D) and banks with homogenous flows throughout the day (labelled E and F). The results show that the total costs of D converge to a higher level than the costs of C. For the E and F pair, the total value of payments sent and received between each other is quite similar, and the total cost, as expected, converges to a similar magnitude.

We also performed the robustness exercise varying the delay cost. The box plots in Figure 20 presents the results for pair C and D, and Figure 21 presents the results for pair E and F. These show the importance of the payment profile for the sensitivity of the delay cost: for pair E and F that have very similar profiles, the variation of the delay cost affects the behaviour and the total cost only slightly. On the contrary, for pair C and D, the change in the delay cost matters substantially for C because it has the larger payment demand, in spite of this coming only in the last three intraday periods. Note that, like for the case of agents A and B, when the delay cost is below the liquidity cost, the sensitivity in total cost is quite large.

Figure 18: Payment demands for banks labelled C and D (left), and banks labelled E and F (right). Each plot overlays the hourly (bars, left axis scale) and cumulative (lines, right axis scale) sums of payments between each of the two agents over the sample. The solid lines are the average, and the shaded areas are the 99% confidence intervals. To contrast with banks A and B examined in the main text, these pairs were chosen to show banks with diverging flows late in the day (C and D) and banks with homogenous flows throughout the day (E and F).

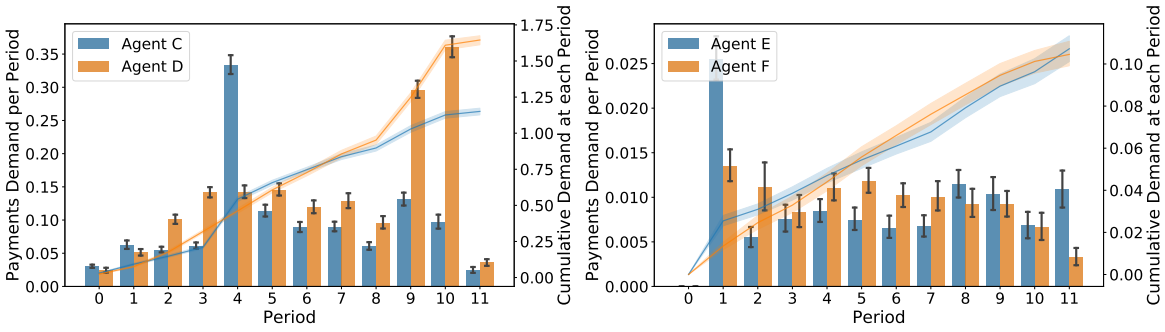


Figure 19: Robustness of the training of initial liquidity problem under different payment profiles: pair C and D on the left and pair E and F on the right. The solid lines are the average, and the shaded areas are the 99% confidence intervals. Hyperparameters are as outlined in Table 2.

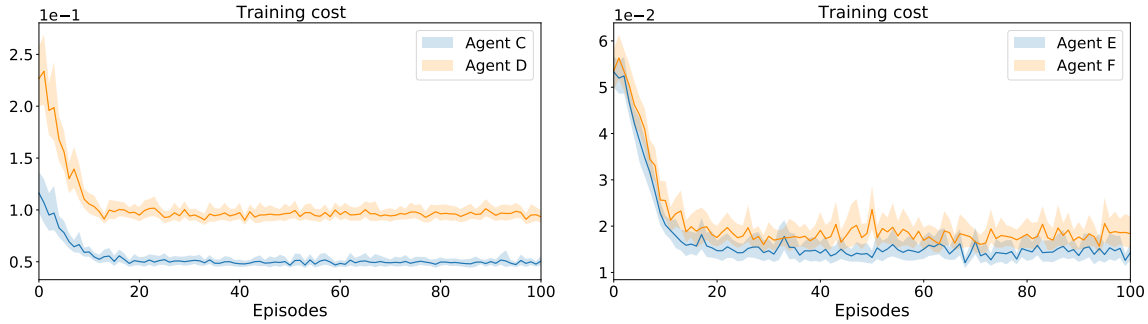


Figure 20: Boxplots showing the sensitivity of the total training cost to a variation in the cost of delay for agents C and D. Whiskers are defined as $1.5 \cdot$ IQR. Delay cost is varied between 0.01 and 0.3, while initial liquidity cost is maintained at 0.1. The rest of the hyperparameters are as outlined in Table 2.

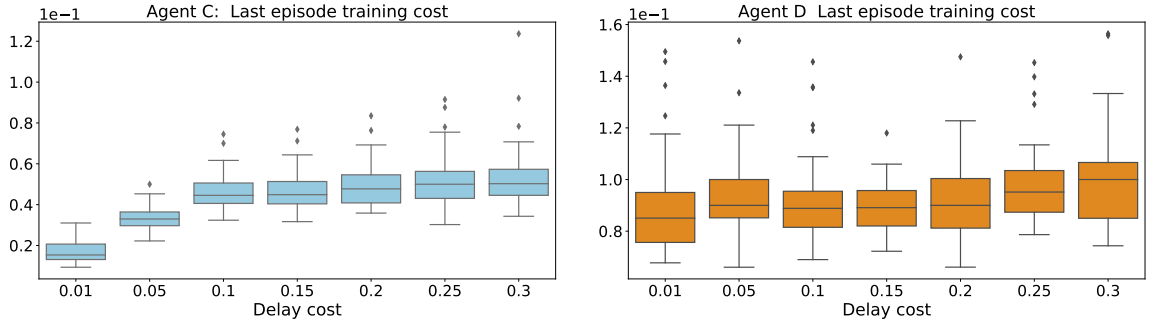


Figure 21: Boxplots showing the sensitivity of the total training cost to a variation in the cost of delay for agents E and F. Whiskers are defined as $1.5 \cdot$ IQR. Delay cost is varied between 0.01 and 0.3, while initial liquidity cost is maintained at 0.1. The rest of the hyperparameters are as outlined in Table 2.