Smart Stormwater Control Systems: A Reinforcement Learning Approach

Cheng Wang
Department of Engineering Systems and Environment, University of Virginia
cw8xk@virginia.edu

Benjamin Bowes
Department of Engineering Systems and Environment, University of Virginia
bdb3m@virginia.edu

Arash Tavakoli
Department of Engineering Systems and Environment, University of Virginia
at5cf@virginia.edu

Stephen Adams
Department of Engineering Systems and Environment, University of Virginia
sca2c@virginia.edu

Jonathan Goodall
Department of Engineering Systems and Environment, University of Virginia
goodall@virginia.edu

Peter Beling
Department of Engineering Systems and Environment, University of Virginia
pb3a@virginia.edu

ABSTRACT

Flooding poses a significant and growing risk for many urban areas. Stormwater systems are typically used to control flooding, but are traditionally passive (i.e. have no controllable components). However, if stormwater systems are retrofitted with valves and pumps, policies for controlling them in real-time could be implemented to enhance system performance over a wider range of conditions than originally designed for. In this paper, we propose an autonomous, reinforcement learning (RL) based, stormwater control system that aims to minimize flooding during storms. With this approach, an optimal control policy can be learned by letting an RL agent interact with the system in response to received reward signals. In comparison with a set of static control rules, RL shows superior performance on a wide range of artificial storm events. This demonstrates RL's ability to learn control actions based on observation and interaction, a key benefit for dynamic and ever-changing urban areas.

Keywords

Reinforcement Learning, Stormwater, Flooding Control.

INTRODUCTION

Flooding poses a significant and growing risk for many urban areas, with the potential to disrupt normal and emergency operations, damage infrastructure, and cause loss of life (Galloway et al. 2018). Stormwater infrastructure provides critical services in such cities by collecting and moving water (Viessman et al. 1998), with one of the main goals being flood prevention. Traditionally, stormwater systems are designed as static systems with capacity for a specified size of storm. However, increasing urbanization (UN 2018) and changing precipitation patterns from climate change (Wuebbles et al. 2017) could stress stormwater systems beyond their designed capacity, leading to reduced system performance and increase flooding.

One way to adapt traditional stormwater systems to changing conditions is to make them larger (e.g. replacing small pipes with larger ones). These capital improvement projects are typically costly and disruptive for the normal operation of a city. In fact, research suggests that such piece-wise improvements can degrade total system performance (Emerson et al. 2005; Petrucci et al. 2013).

Instead of capital improvement to increase the actual capacity of a stormwater system, real-time control of these systems could increase their effective capacity in a more cost efficient way. With the current growth in the applications of Internet of Things (IoT) and ubiquitous sensing, stormwater systems can be retrofitted with sensors, valves, and pumps that are capable of real-time operation. By using the data provided from the sensors and monitoring and controlling these structures in real-time, existing stormwater systems can handle larger storms (Kerkez et al. 2016); this is known as active control.

One method of automating active control is with a type of machine learning called Reinforcement Learning (RL). In RL, an agent interacts with its environment, and learns to optimize its behaviour in response to reward signals. Some previous research in water resources engineering has used RL as an alternative to dynamic programming, in part because of the ability of certain RL algorithms to overcome the curse of dimensionality associated with dynamic programming (Delipetrev et al. 2016). For example, Lee and Labadie 2007 used Q-Learning, a specific RL algorithm, in a two-reservoir system and successfully outperformed a stochastic dynamic programming approach. Castelletti et al. 2013 have used batch reinforcement learning to design the optimal operation of a water reservoir to meet multiple objectives. In their study, by using Q-iteration in combination with decision trees, they have shown the feasibility of using such models for real-time control.

While RL has had some use for multi-objective reservoir management, there has been very little research using RL for stormwater system control. Mullapudi and Kerkez 2018 have recently demonstrated the use of RL, with a Deep Q-Network, for stormwater system control, providing new insights and challenges for future research. The presented study advances knowledge of the use of RL for stormwater systems by using an RL agent suitable for continuous action spaces.

The main contribution of this paper is to design and analyze an RL-based stormwater system for flooding control. In particular, we propose a reward structure for environments where reward signals are sparse. The effectiveness of our approach is demonstrated by the RL agent's superior out-of-sample performance compared to a set of simple benchmarks.

PROBLEM DESCRIPTION

In order to evaluate the use of RL for flood reduction, a simple stormwater system (Fig. 1) is simulated using the U.S. Environmental Protection Agency's Stormwater Management Model, version 5 (SWMM5). SWMM provides an integrated environment for designing study areas, editing input data, applying user-defined control policies, running hydrologic and hydraulic simulations, and tracking a variety of measurements (such as depth and flooding at a storage unit) during simulations. The inputs to the SWMM simulation are time series of rainfall data; during the simulation, rain falls on the subcatchments and is transformed into volumes of water in the storage units. From the storage units, water flows through a single pipe to the system outfall where it exits the system. Flow out of the storage units is controlled by two orifices (valves). The pyswmm (https://github.com/OpenWaterAnalytics/pyswmm) Python wrapper for SWMM is used to enable the step-by-step running of a stormwater simulation as required for RL.

In this system, flooding can occur at either storage unit (St1, St2) or at the downstream junction (J3). If water enters the storage units faster than it drains out, the storage units can over-top and cause flooding. However, if flow out of the storage units is not controlled, their combined discharge can overwhelm the capacity of the pipes downstream and cause flooding. The problem, therefore, is to develop policies (mapping of stormwater system conditions to valve positions) that can minimize or eliminate flooding for a variety of rainfall events.

A BRIEF INTRODUCTION TO REINFORCEMENT LEARNING

In machine learning, reinforcement learning (RL) (Sutton and Barto 1998) attempts to learning an optimal control policy for an agent through sequential interactions with an environment. In the standard RL paradigm, an agent observes their current state and takes an action based on its current control policy. The agent then receives a reward from the environment and observes a state transition. The process then repeats. The ultimate goal of the agent is to learn a policy through these interactions that maximizes expected future discounted reward. A key concept in RL is exploration during the learning phase, where an agent takes actions simply to observe the associated reward. RL can often require a large number of interactions due to the stochastic nature of the environment.

Sequential decision-making problems are often modeled as Markov decision processes (MDPs) which are composed of the states, actions, rewards, a transition function, and a discount factor. Let $\mathcal{S}$ represent the set of I states, and let $\mathcal{A}$ represent the set of K actions. Further, let $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The reward function, which is represented by $r(s, a, s')$, is assumed to be dependent on the state transition and the action, where s' represents the state after the

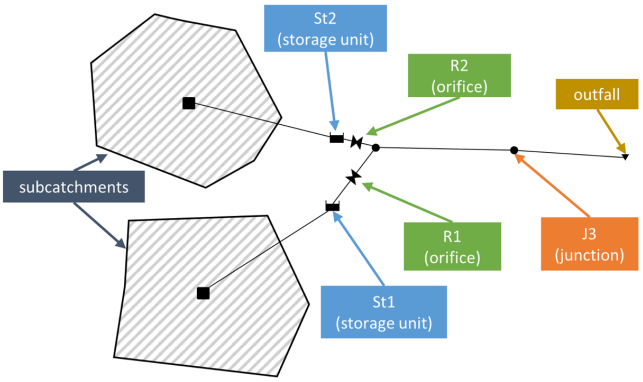

Figure 1. A simple stormwater system

transition [1]. The transition function represents the probability of a state transition and is written as $P^a_{s,s'} = \mathbb{P}(s'|s,a)$. A discount factor $\gamma \in [0,1]$ is used to determine the present value of future rewards. (For example, a reward of 10 received in n time steps is only worth $10\gamma^{n-1}$ now.) The sum of discounted future rewards at time t can be written as $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $r_t = r(s_t, a_t, s_{t+1})$. A policy maps states to actions and can be written as $\pi(a|s)$ and represents the probability of taking action a in state s.

The expected future discounted reward when starting in s and following π is the value function and is written as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}(G_t|s) \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \left(r(s,a,s') + \gamma \sum_{s'\in\mathcal{S}} P^a_{s,s'} V^\pi(s') \right), \end{aligned} \tag{1}$$

where $\mathbb{E}[\cdot]$ is the expectation. The expected future discounted reward when starting in s, then taking a, and following π thereafter is called the Q-value function[2] and is written as

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}(G_t|s,a) \\ &= r(s,a,s') + \gamma \sum_{s'\in\mathcal{S}} P^a_{s,s'} \sum_{a'\in\mathcal{A}} \pi(a'|s') Q^\pi(s',a'). \end{aligned} \tag{2}$$

An optimal policy is found by maximizing either the value function or the Q-value function

$$V^*(s) = \max_a \left(r(s,a,s') + \gamma \sum_{s'\in\mathcal{S}} P^a_{s,s'} V^*(s') \right), \tag{3}$$

and

$$Q^*(s,a) = r(s,a,s') + \gamma \sum_{s'\in\mathcal{S}} P^a_{s,s'} \max_{a'} Q^*(s',a'). \tag{4}$$

An optimal policy can be estimated using dynamic programming if all of the components of the MDP are known and the state and action spaces of the MDP are small enough to efficiently compute the value or Q-value functions. If this is not the case, reinforcement learning algorithms can be used to estimate the optimal policy.

In practice, it is often more convenient to work with Q-value functions. For instance, suppose we have found the optimal Q-value function $Q^*(s,a)$, then an optimal policy can be obtained by choosing the action that produces the largest Q-value at the current state s, i.e., $\pi^*(s) = \arg\max_a Q^*(s,a)$.

Many RL algorithms use the Bellman equation

$$Q^\pi(s,a) = \mathbb{E}\big[r(s,a) + \gamma \mathbb{E}_{a\sim\pi} \left[Q^\pi(s',a) \right] \big], \tag{5}$$

to calculate Q-value function recursively:

$$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha\delta, \tag{6}$$

[1] For some MDPs, the reward function may only depend on the current state s and the current action a. In such cases, the reward function can be denoted as $r(s,a)$

[2] Also called the action-value function.

where α is the learning rate and $\delta = y - Q^{\pi}(s,a)$ is the temporal difference error. In particular, Q-learning algorithm (Watkins and Dayan 1992) uses $y = r + \gamma \max_a Q^{\pi}(s',a)$ to estimate the optimal Q-value function $Q^*(s,a)$. For continuous state and action spaces, the Q-value function can be approximated by a neural network parameterized by θ^Q. However, apply Q-learning to continuous action spaces is not straightforward, as calculating $\max_a Q^{\pi}(s',a)$ requires an optimization over the entire action space at every timestep.

Instead of finding the optimal policy through value functions, policy gradient methods learn a parameterized policy π_θ by optimizing a performance measure $J(\theta)$ directly. Specifically, let $J(\theta)$ denote the expected return under policy π_θ from an initial state distribution, then θ can be updated through gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$. Note that the estimated gradient in actor-only methods may have a large variance, as no information about previous estimates is used in computing the gradient (Sutton, McAllester, et al. 2000; Grondman et al. 2012).

Actor-critic methods combines value function approximations with a parameterized policy function (Konda and Tsitsiklis 2000). Specifically, the policy function (the actor) generates and executes an action a for the current state s, and the critic updates the value function using the observed sample (s,a,s'). The value function is then used in the policy gradient to update actor parameters. For example, consider a deterministic policy π_θ, and a critic $Q^w(s,a)$ that approximates the action-value function $Q^{\pi_\theta}(s,a)$, then the off-policy deterministic actor-critic algorithm (Silver et al. 2014) updates the actor and critic parameters as follows :

$$\begin{aligned} \delta &= r + \gamma Q^w(s', \pi_\theta(s')) - Q^w(s,a) \\ w &\leftarrow w + \alpha_w \delta \nabla_w Q^w(s,a) \\ \theta &\leftarrow \theta + \alpha_\theta \nabla_a Q(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) \end{aligned} \tag{7}$$

Compared with actor-only methods, actor-critic methods can reduce the variance of policy gradients by making use of the critic's estimates of the expected returns, while still being able to generate continuous actions.

PROPOSED APPROACH

In this section, we will formally define the reinforcement learning environment for the stormwater system in Figure 1. The definitions of states, actions, and the reward function can be easily extended to more complex systems. For control problems where reward signals are sparse, we propose a simple way to modify the reward function that could lead to faster learning process and more robust policy. We also briefly describe the learning algorithm for our experiments.

State and Action Spaces

At each time step, the state of the environment, $\mathbf{s}$, is defined as observations of current depth and flooding amount at different nodes in the stormwater system. Specifically, in our example system, $\mathbf{s}$ is a vector in $\mathbb{R}^6$: $\mathbf{s} = (d_1, d_2, d_3, f_1, f_2, f_3)$, where d_1, d_2, d_3 are current depths at storage unit 1 (St1), storage unit 2 (St2), and junction 3 (J3), respectively, and f_1, f_2, f_3 are current flooding amounts at St1, St2, and J3, respectively.

Based on the state information and the policy, the agent then determines opening levels for both orifice R1 and R2: $\mathbf{a} = (a_1, a_2)$, where $0 \le a_1, a_2 \le 1$. Note that under the current setting, the action space is continuous, so orifice opening can be adjusted to any level between fully open (1) and fully closed (0).

Reward Function

As our objective is to minimize the total cumulative amount of flooding at St1, St2 and J3, it is natural to use the following reward function for transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$: $r = -(f_1' + f_2' + f_3')$. That is, the reward r is the negative of total flooding observed at the next time step. However, since flooding may only happen occasionally, reward signals during a learning trial will mostly be zeros. This will lead to a slow learning process and may increase the chance of overfitting. In addition, as any arbitrary orifice setting will have the same impact during no-flood periods, the learned policy could be too volatile and risky. For example, the valve could be fully closed at time t, fully open at $t+1$, and close again at $t+2$, when there is no flooding or any rainfall at all. Policy of this type may be too difficult or costly to implement in practice.

To accelerate the learning process and to make the RL-based policy more robust, we propose a new reward function that will make use of a baseline control policy. Specifically, let $\mathbf{a}^b$ denote the action for state $\mathbf{s}$ under the baseline policy (e.g., a policy from an expert), then the reward for transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ is defined as

$$r = -(f_1' + f_2' + f_3') - \beta \cdot \|\mathbf{a} - \mathbf{a}^b\|_1 = -(f_1' + f_2' + f_3') - \beta \cdot (|a_1 - a_1^b| + |a_2 - a_2^b|), \tag{8}$$

where $\beta > 0$ is a hyper-parameter that controls the degree of impact from the baseline on the reward function (we choose $\beta = 1$ for our experiments described in the next section). It is easy to see that in order to maximize the cumulative reward, the agent will try to follow the baseline policy if there is little flooding (i.e., $f_1', f_2', f_3' \approx 0$), which should help make the final policy more stable. When the amount of flooding is not negligible, the agent will update its current policy according to the flooding situation, and will no longer be restricted to the baseline policy.

Learning Algorithm

As the action space in our problem is continuous, applying traditional value function-based RL algorithms (such as Q-learning) is not straightforward. Finding the orifice opening levels that produce the largest $Q-$value will require an optimization process over a continuous space at every time step. Discretizing the action space may create more problems than it solves: we would need to decide what ranges of actions should be clustered together as one single action and what its value should be. In addition, the total number of actions grows exponentially with the number of control units, which significantly limits the scalability of the control system. While policy gradient methods can be used for continuous control problems, as mentioned in the previous section, they may have a large variance, leading to slow learning.

Therefore, to deal with continuous action space while reducing the variance, we will use an actor-critic algorithm, Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015), for its simplicity and popularity, to train our RL agent. DDPG uses deep neural networks to approximate the Q-value function (the critic) and the policy function (the actor). During each time step, an action is selected by the actor and executed, and the transition $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ is stored in a replay buffer. Then a mini-batch of transitions is sampled and used to update the critic by minimizing the loss between outputs from the critic network and Q-values calculated from the target network. The actor network is updated by following the policy gradient. Finally, the weights of target critic and actor networks are updated softly: $\theta' \leftarrow \tau\theta + (1-\tau)\theta$, where θ' is the new weight and $\tau << 1$.

Recall that for our example, the state at a given time step is represented as a vector in $\mathbb{R}^6$ and the action is a vector in $\mathbb{R}^2$. Due to the simplicity of the state and action spaces, for experiments in the following section, we will use the feedforward neural network structure for both the actor and the critic. Specifically, the actor has two layers with 16 neurons in each layer, and the critic consists of three layers with 32 neurons in each layer.

EXPERIMENTS

In this section, we demonstrate the effectiveness of reinforcement learning based stormwater control systems. First, we introduce the experiment settings and the dataset. Then, we investigate some simple baseline policies, and show that it is not always easy to outperform a common-sense baseline such as keeping the valves open at 50% all the time. We then evaluate the RL agent's learning and generalization ability through in-sample and out-of-sample analysis.

Data and Environment

Our stormwater system is created using the Storm Water Management Model (SWMM) developed by the U.S. Environmental Protection Agency. We then generate 100 artificial storm events, and use them to create 100 SWMM models that only differ in the rainfall input. The 100 storms are carefully selected such that: (i) they have a wide range of intensities and (ii) the magnitude of each individual storm is large enough to cause flooding under the baseline policy. Figure 2 shows two simulated storms where the rainfall time series for the two rain gages are plotted. The pyswmm package is used to interact with the SWMM model (for example, to control the valve openings during simulations). Simulation of each model lasts for 24 hours, and each time step during a simulation represents 15 minutes.

Baseline Policy

The simplest type of control policy for our stormwater system is to keep the two valves open at (x%,y%) level at all times. Due to the symmetric nature of the system topology and model inputs, we consider the following baseline policies: 10%, 25%, 50%, 75%, and 90% opening of both valves. Table 1 summarizes the results from running these policies on all 100 SWMM models. As we can see, keeping the valves at 50% level generated the least amount of flooding on average with a relatively small variance. Figure 3 shows depths and flooding at both storage units and the junction node during the first storm under baseline policy (50%,50%). The cumulative amount of flooding over the simulation period is 70.6 thousand cubic feet (MCF).

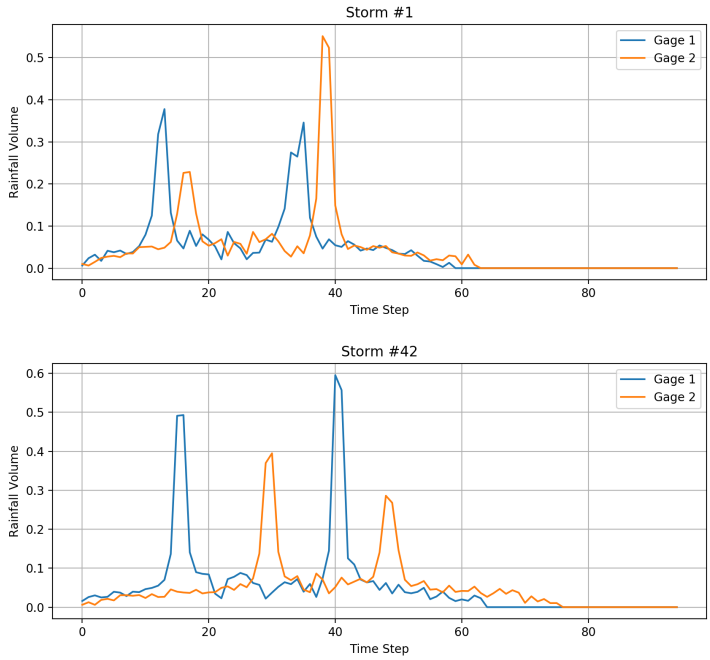

Figure 2. Rainfall data from artificial storm events

Table 1. Flooding from Baseline Policies

Policy	Average of Total Flooding (10^3 ft^3)	Standard Deviation (10^3 ft^3)
(10%, 10%)	198.4	77.7
(25%, 25%)	96.3	64.8
(50%, 50%)	88.6	52.7
(75%, 75%)	98.8	50.7
(90%, 90%)	100.2	50.5

Rule-based Control Policies

Before we evaluate the RL-based policy against the static baseline policy (50%,50%), we would like to examine another type of policies that could be described as control rules, and to see if we could obtain a better baseline by simply switching to a dynamic policy. Expert knowledge is often required for designing complex control rules that can utilize more than just local information. Extensive experiments may also be needed for tuning parameters in order to achieve satisfying results. However, here we will just consider two simple policies that choose the target setting for each orifice based solely on the depth of the corresponding storage unit.

In particular, as water level increases in a storage unit, to avoid or to mitigate flooding at the same unit, the corresponding valve will be opened more to let water flow out. Under the first policy, valve opening, a, is linearly dependent on the depth of storage unit, d, until the depth exceeds a threshold H=4 [3], in which case opening will be set to *max_open*=0.95. [4] Specifically, we have

$$a = \begin{cases} 0.95 \times d/H & \text{if } d < H, \\ 0.95, & \text{if } d \geq H. \end{cases}$$

The second policy has four predetermined settings, corresponding to different intervals for the current depth of storage unit. For example, if the current depth of storage unit 1 is between 1ft and 3ft, then the opening of orifice 1 is set to be 0.5. Figure 4 plots both policies.

[3]The full depth of each storage unit is set to be 5 (ft).

[4]Fully open a valve (i.e., set *max_open* to one) will send more flow to the downstream node J3, which could increase the risk of flooding. As a compromise, here we choose 95% as the maximum opening level.

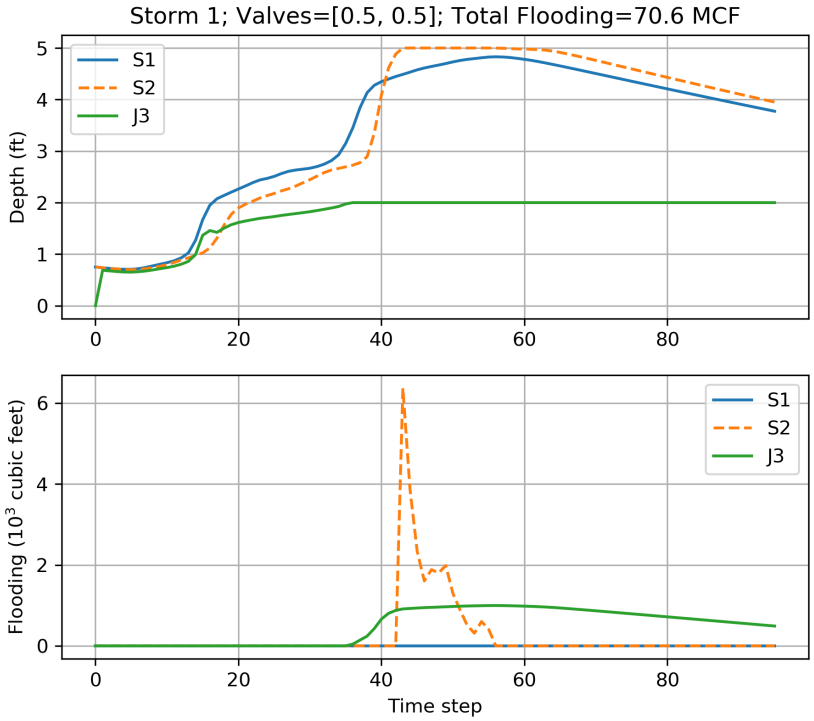

Figure 3. Performance of the baseline policy during the first storm

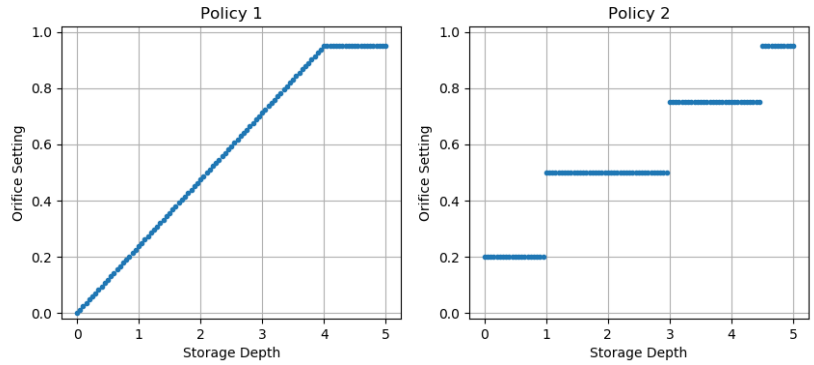

Figure 4. Rule-based policies

We then test each policy against the fixed [50%,50%] baseline on all 100 storm events. As can be seen from Figure 5, both policies fail to reduce the total flooding compared with the [50%,50%] baseline.

Upon further investigation, the worsening of the situation is mainly due to more severe flooding at the downstream node J3. Figure 6 shows that under both policies, the amount of flooding at J3 increases by roughly 20 thousand cubic feet compared with the baseline for every storm in the dataset. This should not be surprising, since both policies are sending more water (and at a higher speed) to the downstream node as water level in storage units increases. Although this may lower the risk of flooding at storage units, their capacities may not be fully utilized. In other words, water may be let out of the storage units unnecessarily and/or too early. During the peak of storms, water level in J3 quickly surpasses its maximum depth (2 ft), causing intense flooding.

Overall, these experimental results from applying rule-based control policies illustrate that it is not always easy to beat the common sense baselines (e.g., the [50%,50%] fixed policy). To find successful control rules, one need to experiment with more structures and parameter values , while also taking more than local information into account.

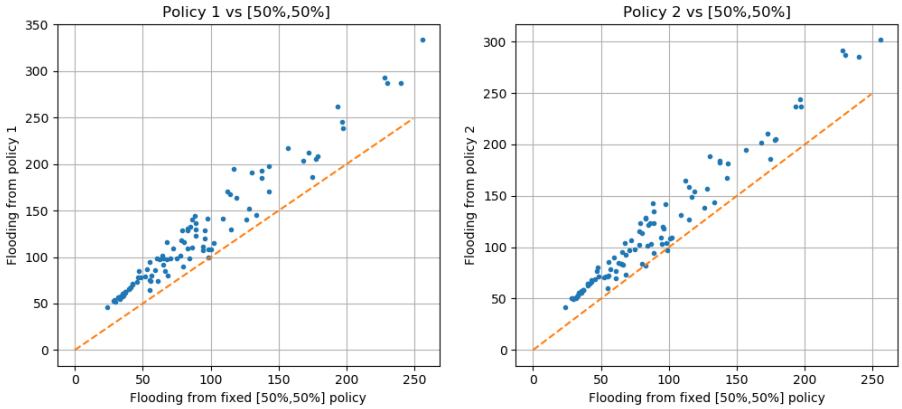

Figure 5. Comparison of total flooding

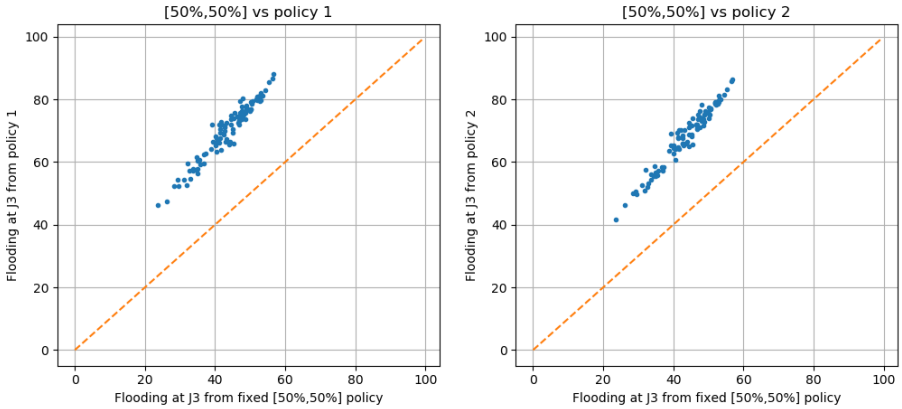

Figure 6. Comparison of flooding at J3

Performance of the RL-based Policy

We first experiment with training and testing the RL agent on the same storm event. In Figure 7, we plot the learned policy (top panel) and the amount of flooding (bottom panel) during the first storm after 50,000 steps of training. Both valves are kept approximately half open until the storm intensifies around t=40. As water accumulates more rapidly in storage unit 2 than in unit 1 (see rainfall time series in Figure 2), valve 2 is opened more to let water flow out. At the same time, valve 1 is fully closed to keep additional water from flowing to the downstream node J3. When the capacity of storage unit 1 is eventually reached and flooding starts to happen around t=50, valve 1 is quickly opened to mitigate local flooding while keeping the impact on node J3 low. The rainfall stops after t=65, and each valve is turned into a partially open position to gradually reduce the water level in the corresponding storage unit.

By dynamically controlling the valve openings, the RL agent is able to reduce the total amount of flooding to just 28.8 thousand cubic feet from 70.6 thousand resulted from the baseline policy (50%,50%) . It is worth noting that flooding is distributed more evenly across all three nodes in the system. Compared with the baseline result (Fig 3), although there is still some flooding at storage unit 1, the magnitude and duration of flooding at storage unit 2 and the junction node J3 have been dramatically reduced by the RL-based policy. In particular, flooding at J3 stops after 60 time steps. In contrast, under the baseline policy, J3 remains flooded until the end of the simulation.

We also experimented with more training steps, and found similar results, indicating the policy shown in Fig 7 is close to the optimal solution. Note that flooding is inevitable during some heavy rainfall events, given the physical limitations of the stormwater system (for instance, the maximum depth of J3 in our model is only 2 ft).

To examine the learning ability of the RL agent and the impact of storm magnitude on its performance, we train the RL agent for 10,000 steps separately for each of the 100 simulated storm events. The agent is then tested on the same training storm, and total flooding is compared with the one from the baseline policy (50%,50%). Figure 8

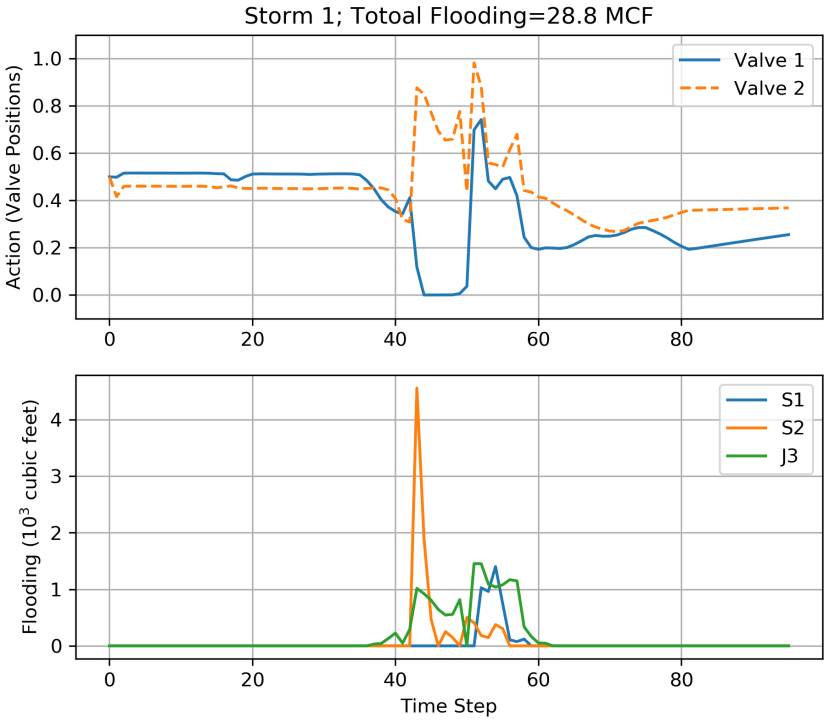

Figure 7. Policy and performance of the RL agent during the first storm

shows percentages of flooding reduced by the RL agent and the corresponding total flooding from the baseline policy on all 100 storms. Table 2 reports the improvement of performance by the RL agent under different storm strengths. We can see that when rainfall intensity is low and the system is less flooded under the baseline policy (for example, when total flooding is less than 50 thousand cubit feet), the RL agent is able to significantly reduce the amount of flooding, if not eliminate it altogether. For moderate storms that could cause a flooding of volume between 50 and 100 thousand cubic feet , the agent can still reduce the amount of flooding by 53.8%. However, under extremely severe storms, the effect of redistributing flow using RL is limited, as system capacity is quickly reached to its maximum.

Table 2. Flooding reduced by RL agent

Storm Intensity	Baseline Flooding (10^3 ft^3)	Avg. Reduction	Std. Reduction
Low	[0,50]	94.0%	5.2%
Medium	(50,100]	53.8%	17.0%
High	(100,150]	32.2%	10.0 %
Severe	(150, ∞)	13.2%	11.3%

Testing the RL-based Policy on Unseen Storms

To evaluate RL agent's generalization ability, we train the agent using the first 50 storm events. During each iteration of the training process, a sample storm is randomly selected, and the agent is trained for N steps. We then validate performance of agents trained with different training time on the next 10 storms. After the validation process, the agent with 150 iterations of 1000 steps of training is selected for testing on the remaining 40 events. Again, the baseline policy (50%,50%) is used for comparison. In Fig 9, we plot percentages of reduction in flooding by the agent against total flooding from the baseline policy on the whole testing set. Table 3 summarizes the results for different storm magnitudes.

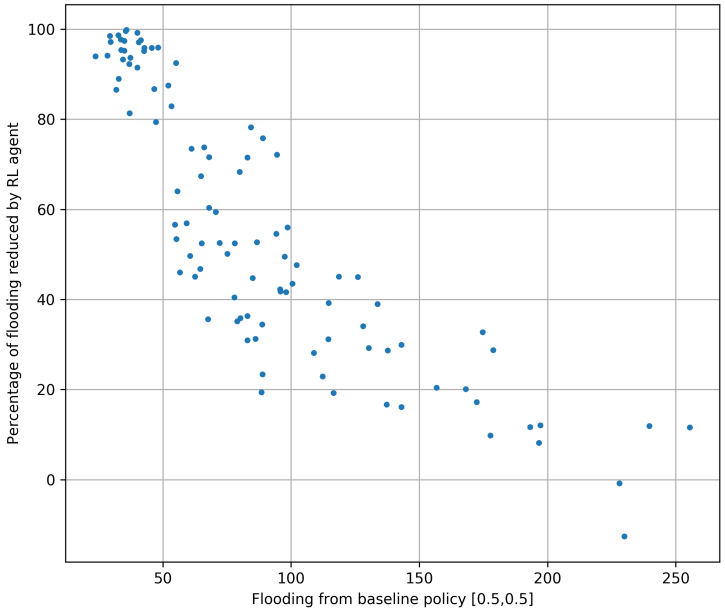

Figure 8. Percentage of flooding reduced by RL agent on 100 storm events

Table 3. Flooding reduced by RL agent on the testing set

Storm Intensity	Baseline Flooding (10^3 ft^3)	Avg. Reduction	Std. Reduction
Low	[0,50]	88..5%	21.3%
Medium	(50,100]	45.8%	18.6%
High	(100,150]	35.6%	16.9%
Severe	(150, ∞)	16.6%	8.4%

As we can see, the RL agent is able to reduce flooding considerably for low intensity storms. In some scenarios, flooding is completely eliminated while the baseline policy leads to a total amount of flooding around 30 thousand cubic feet. For medium and high intensity storm events, using RL-based dynamic control policy can still alleviate flooding by 46% and 36% on average, respectively. Under severe storms, the system will quickly reach its full capacity, and it will become more and more difficult for the RL agent to mitigate the flooding.

We find the testing results on unseen storm events quite encouraging, considering that only depths and current flooding information has been taken into account by the agent when determining the valve positions. The current policy could be further improved by having a more detailed representation of the state space . For example, incorporating forecasts of rainfall into the state space could help the agent make better and timely decisions.

CONCLUSION

In this paper, we have developed a reinforcement learning based stormwater control system to mitigate flooding during storm events. To accelerate learning and make the learned policy stable, we introduced a reward function that trades off flooding reduction from the RL agent's current policy with the risk of deviating from a known baseline policy. We also investigated different control rules, and found that a static and neutral policy such as keeping the valves open at 50% level could consistently outperform other simple baselines. This policy is then used as a benchmark for comparison with the RL-based control policy. The out-of-sample results showed that the RL agent is able to reduce the total flooding volume significantly for most synthetic storms in our dataset, indicating good generalization ability of the RL agent and the overall effectiveness of our approach. The results of this study revealed the potential of applying RL to stormwater control systems in order to reduce the risk of flooding during

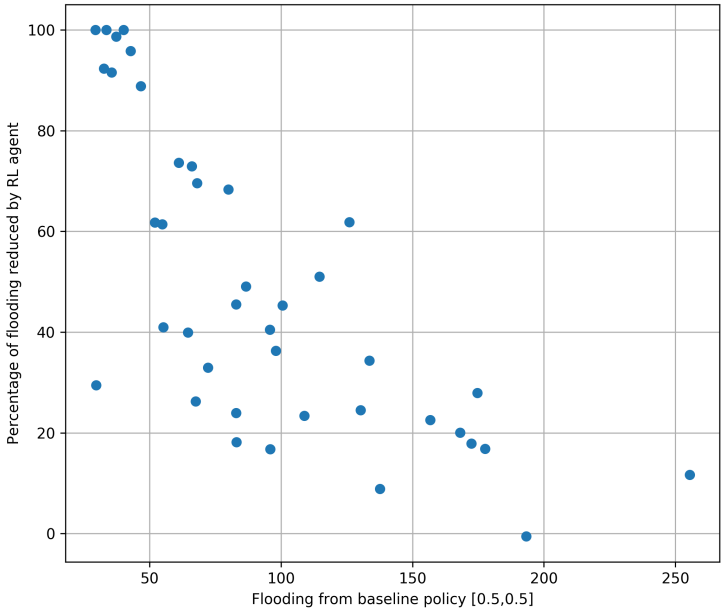

Figure 9. Percentage of flooding reduced by RL agent on the testing set

storm events. An optimal control policy could be learned automatically, without relying on expert knowledge to design complex control rules.

A significant assumption in this work is that observations of depth and flooding amount at every node comprise complete information about the state of a stormwater system, which could be an oversimplification. Additional measurements (e.g., flow rate/velocity in pipes) may need to be included in state definition as well. On the other hand, it could also be impossible to have complete information about the true underlying state, no matter how many types of measurements are considered. In this case, in order to better understand the environment and to perform well, the RL agent may need to consider long sequences of past observations (and actions).

We also assumed that the objective of the RL agent is to minimize the total amount of flooding during a storm event. Under this assumption, the duration of flooding could be ignored: Some place may remain flooded for a long period of time, although the magnitude at any given moment is small. In addition, each part of the system is treated equally. This design may be questionable in the real world. For instance, populated areas should require higher priorities than remote parks: any flooding occurred in the former should have heavier penalties. Future work in this direction would consider alternative objectives in flooding control and correspondingly, reward functions with more complex structures.

Future work would also include applying RL to more sophisticated and realistic models of stormwater systems. For example, in coastal cities, tidal changes and storm surge can cause water to back up in pipes and prevent water flow out of the system. A more challenging environment for the RL agent could then be built by incorporating tidal movements into a SWMM model with multiple nodes and a complex network structure.

REFERENCES

Castelletti, A., Yajima, H., Giuliani, M., Soncini-Sessa, R., and Weber, E. (2013). “Planning the optimal operation of a multioutlet water reservoir with water quality and quantity targets”. In: *Journal of Water Resources Planning and Management* 140.4, pp. 496–510.

Delipetrev, B., Jonoski, A., and Solomatine, D. P. (2016). “A novel nested stochastic dynamic programming (nSDP) and nested reinforcement learning (nRL) algorithm for multipurpose reservoir optimization”. In: *Journal of Hydroinformatics* 19.1, pp. 47–61.

Emerson, C. H., Welty, C., and Traver, R. G. (2005). “Watershed-scale evaluation of a system of storm water detention basins”. In: *Journal of Hydrologic Engineering* 10.3, pp. 237–242.

Galloway, G. E., Reilly, A., Ryoo, S., Riley, A., Haslam, M., Brody, S., Highfeld, W., Gunn, J., Rainey, J., and Parker, S. (2018). *THE GROWING THREAT OF URBAN FLOODING: 2018*. Tech. rep. College Park: University of Maryland, Center for Disaster Resilience, Texas A&M University, Galveston Campus, Center for Texas Beaches, and Shores.

Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. (2012). "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6, pp. 1291–1307.

Kerkez, B., Gruden, C., Lewis, M., Montestruque, L., Quigley, M., Wong, B., Bedig, A., Kertesz, R., Braun, T., Cadwalader, O., et al. (2016). "Smarter Stormwater Systems". In: *Environmental Science and Technology* 50, 72677273.

Konda, V. R. and Tsitsiklis, J. N. (2000). "Actor-critic algorithms". In: *Advances in neural information processing systems*, pp. 1008–1014.

Lee, J.-H. and Labadie, J. W. (2007). "Stochastic optimization of multireservoir systems via reinforcement learning". In: *Water resources research* 43.11.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

Mullapudi, A. and Kerkez, B. (2018). "Autonomous Control of Urban Storm Water Networks Using Reinforcement Learning". In: *EPiC Series in Engineering* 3, pp. 1465–1469.

Petrucci, G., Rioust, E., Deroubaix, J. F., and Tassin, B. (Apr. 2013). "Do stormwater source control policies deliver the right hydrologic outcomes?" In: *Journal of Hydrology* 485, pp. 188–200.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). "Deterministic policy gradient algorithms". In:

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*. Vol. 2. 4. MIT press Cambridge.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*, pp. 1057–1063.

UN (May 2018). *68 percent of the world population projected to live in urban areas by 2050, says UN | UN DESA Department of Economic and Social Affairs*.

Viessman, W., Hammer, M. J., Perez, E. M., and Chadik, P. A. (1998). "Water supply and pollution control". In:

Watkins, C. J. and Dayan, P. (1992). "Q-learning". In: *Machine learning* 8.3-4, pp. 279–292.

Wuebbles, D., Fahey, D., Hibbard, K., Dokken, D., Stewart, B., and Maycock, T. (2017). *Climate Science Special Report: Fourth National Climate Assessment, Volume I*. Tech. rep. Washington, DC: U.S. Global Change Research Program (USGCRP), p. 470.