Leveraging Open Source Software and Parallel Computing for Model Predictive Control of Urban Drainage Systems using EPA-SWMM5

Jeffrey M. Sadler[1], Jonathan L. Goodall, Ph.D.[2], Madhur Behl, Ph.D[3], Mohamed M. Morsy, Ph.D[4, 5], and Teresa Culver, Ph.D[6]

[1]*Graduate Research Assistant, Dept. of Engineering Systems and Environment, Univ. of Virginia, 151 Engineers Way, P.O. Box 400747, Charlottesville, VA 22904, jms3fb@virginia.edu*

[2]*Associate Professor, Dept. of Engineering Systems and Environment, Univ. of Virginia, 151 Engineers Way, P.O. Box 400747, Charlottesville, VA 22904, goodall@virginia.edu (Corresponding Author)*

[3]*Assistant Professor, Computer Science Dept, madhur.behl@virginia*

[4]*Research Associate, Dept. of Engineering Systems and Environment, Univ. of Virginia, 151 Engineers Way, P.O. Box 400747, Charlottesville, VA 22904, mmm4dh@virginia.edu*

[5]*Irrigation and Hydraulics Department, Faculty of Engineering, Cairo University, P.O. Box 12211, Giza 12613, Egypt, mohamedmorsy@eng.cu.edu.eg*

[6]*Associate Professor, Dept. of Engineering Systems and Environment, Univ. of Virginia, 151 Engineers Way, P.O. Box 400747, Charlottesville, VA 22904, tbc4e@virginia.edu*

Abstract

Active stormwater control may play an important role in mitigating urban flooding which is becoming more common with climate change and sea level rise. In this paper we describe and demonstrate swmm_mpc, software we developed for running model predictive control (MPC) for urban drainage systems using open source software (Python and the EPA Stormwater Management Model version 5 (SWMM5)). swmm_mpc uses an evolutionary algorithm as an optimizer and supports parallel processing. In the demonstration case, the control policy found by swmm_mpc for two storage units achieved its objectives of 1) practically eliminating flooding and 2) maintaining the water level at the storage units close to a target level. Although the current swmm_mpc workflow was feasible for a simple model using a desktop PC, a high-performance computer or cloud-based computer with more computational cores is needed for a more complex model.

Highlights

- Open-source implementation of model predictive control for EPA-SWMM5, swmm_mpc
- Evolutionary algorithm used to select effective control policy at each time step
- Parallel processing of genetic algorithm significantly reduces run-time
- Control policy from swmm_mpc minimizes flooding and maintains target water level
- Computational cost measured for personal, high-performance, and cloud-based computers

1 Introduction

Researchers have predicted that storm intensity will increase on average due to climate change (Berggren et al., 2012; Neumann et al., 2015). Coastal cities have an additional challenge as sea levels rise which makes it more difficult to drain storm runoff from streets. Coastal cities have already experienced increased flooding from high tidal events alone (Sweet and Park, 2014).

More intense storms and rising sea levels will put greater stress on urban drainage systems necessitating changes for urban drainage systems to perform at current levels. One possible adjustment is to make capital improvements such as increasing pipe size or constructing new storage units. Another option is to convert drainage systems from passive, gravity driven systems to active or "smart" systems (Kerkez et al., 2016). Active systems can increase performance of a urban drainage system at a lower cost than traditional capital improvements (Meneses et al., 2018). Actively controlling an urban drainage system does not increase the *actual* capacity of urban drainage infrastructure, but rather more efficiently uses the existing infrastructure, increasing its *effective* capacity. For example, one part of an active urban drainage system could be a valve at the outlet of a retention basin which can be automatically opened or closed based on system conditions and forecasts. With this setup, the valve could be closed more during a storm which would utilize the available storage better than would have been possible without the valve.

For an active urban drainage system to achieve its objective (e.g., minimize flooding, reduce combined-sewer overflows), an effective management strategy is required. Management decisions for a urban drainage system include which actuators (e.g., valves and pumps) in the system should change, when to change them, and to what setting. We refer to these decisions as a control policy (Vrabie et al., 2009; Mayne et al., 2005; Langson et al., 2004). An effective control policy for an active urban drainage system may depend on a number of factors such as antecedent moisture conditions, expected intensity and duration of oncoming rainfall, current water levels in the system, the condition of the drainage infrastructure, and other factors (e.g., tide levels in tidally influenced urban drainage systems).

A common approach for determining an effective control policy is model predictive control (MPC) (Camacho and Bordons, 2007). MPC has been used effectively in many control applications including automotive controls (Del Re et al., 2010), HVAC (heating, ventilation, and air conditioning) (Afram and Janabi-Sharifi, 2014), and other industrial applications (Qin and Badgwell, 2003). MPC has also been used effectively in urban drainage applications (Puig et al., 2009; Cembrano et al., 2004; Schütze et al., 2004; Gelormino and Ricker, 1994). In MPC, a process model is used to simulate the physical system and evaluate alternative control policies. Forecast data can be used as input for the simulation. During the control period, on-line optimization is performed, meaning that an optimal control policy is found and implemented at each time step (Camacho and Bordons, 2007).

While effective for finding effective control policies, implementing MPC for a urban drainage system is non-trivial due to the dynamics within the system. The fundamental governing equations for modeling urban drainage systems are the St. Venant equations which, when considered fully, are non-linear (Tayfur et al., 1993). This makes finding an optimal control policy for urban drainage systems challenging using MPC (Darsono and Labadie, 2007). To address this dilemma, two alternative approaches are found in the literature. The first is to simplify the governing equations of the process model to a linear system. This makes the optimization problem solvable using well-established procedures such as simplex (Nelder and Mead, 1965). Gelormino and Ricker (1994) took the approach of linearizing their system, converting their process model into a linear-time-invariant model to perform MPC for a large combined sewer system in Seattle, Washington USA.

The second approach is to retain the non-linear St. Venant equations and use a metaheuristic to find the best control policy at each time step. In this approach, a true optimization procedure is not possible because the system remains non-linear; instead, a metaheuristic (e.g., an evolutionary algorithm (EA)) can be used (Gandomi et al., 2013). The use of a metaheuristic precludes the possibility of determining a guaranteed optimal control policy and is typically computationally expensive. The advantage of this approach, however, is that the non-linear governing equations in the process model are retained. This approach was taken by Heusch and Ostrowski (2011) who used a dynamically dimensioned search for finding the best control policy and the United States Environmental Protection Agency's Stormwater Management Model Version 5 (EPA-SWMM5), which numerically solves the St. Venant equations, as their process model (Huber et al., 2005). Similar to Heusch and Ostrowski (2011), we have selected to follow the second approach so that the non-linearities in the process model can remain, and to leverage EPA-SWMM5 as the process model.

EPA-SWMM5 is an attractive choice as a process model for urban drainage systems for several reasons. EPA-SWMM5 is in the public domain making it free of charge and its source code is open-source making it customizable. The model simulates a wide variety of urban drainage structures including active controls such as orifices with variable openings and pumps. EPA-SWMM5 has been used in many research applications, as well as in engineering practice to model urban drainage systems (Burger et al., 2014). Notwithstanding the wide use and utility of EPA-SWMM5 for modeling urban drainage systems, and the established utility of MPC as a successful approach for determining effective control policies, to our knowledge, there is currently no software available for performing MPC using EPA-SWMM5. Although Heusch and Ostrowski (2011) developed software that implements MPC with EPA-SWMM5, that software was closed-source and is no longer available.

This study advances the work done by Heusch and Ostrowski (2011) by creating an open-source implementation of MPC for EPA-SWMM5, *swmm_mpc*, and by demonstrating swmm_mpc's parallel computing capabilities. By making swmm_mpc open source, other researchers will be able to use, improve, and build from the source code. Although, the software written by Heusch and Ostrowski (2011) supported the use of parallel computing, this capability, which is critical to the usability of such software given its associated computational costs, was never demonstrated or tested in the literature.

swmm_mpc was written in the Python programming language. Several third-party Python packages were necessary for the success of this project including pyswmm (https://github.com/OpenWaterAnalytics/pyswmm) and the Distributed Evolutionary Algorithms for Python (DEAP) (https://github.com/DEAP/deap). To evaluate swmm_mpc, it was applied to a use case model with two active control devices. The swmm_mpc results were compared to the results from a rules-based approach and a scenario with no active control. The swmm_ mpc software was run on a desktop personal computer (PC), a high-performance computer (HPC), and a rented, cloud-based machine to demonstrate and test the parallel processing capability of the software.

The remainder of this paper describes the methods used to implement swmm_mpc including a description of the MPC workflow and the interaction and role of the third-party Python libraries. The use case model is then described and the results of the evaluation are presented and discussed. As part of the results and discussion, the benefits of parallelization and the use of a high-performance and cloud-based computing for running swmm_mpc are quantified and discussed.

2 Methods

2.1 Overview of MPC for urban drainage systems

MPC for a urban drainage system consists of three main components as shown in Figure 1. The first component is the physical system, including the system states and system controls. The system states include hydraulic states such as water levels at system nodes and flow rates in system pipes, and hydrologic states such as watershed soil moisture and runoff. In a real system, these states would come from real-time sensors. The system controls are actuators that accept and implement the settings resulting from the MPC process at each time step.

The second component in MPC is a process model used to simulate the future states of the urban drainage system. The process model uses the states read from the urban drainage system as its initial states. The process model also takes future model inputs such as rainfall or tide level. Given the current state of the system and future disturbances, the process model is used to evaluate the effectiveness of control policy candidates.

A control policy consists of one setting for each actuator, for each control time step, for the duration of the control horizon. An individual setting can be a number as would be the case for a valve where the number would correspond to the percent open of the valve. An individual setting can also be a binary setting as would be the case for a pump that can either be on or off. As an example, consider a system with a variably opening valve and an on-off pump with a control horizon of three hours and a control time step of 15 minutes. A control policy for this system would consist of two arrays, an array of numbers between 0 and 1 to specify the percent open of the valve should be, and an array of "on" or "off" to specify the setting of the pump. Both arrays would have 12 settings (four settings per hour for three hours).

To evaluate the effectiveness of a given control policy, the settings in the policy are applied to simulated actuators in the process model and the process model simulation is executed. At the end of the simulation, a cost is determined for the policy. The cost is based on a user-defined cost function. In this study, we consider mainly the cost resulting from flooding but other costs could be considered withing this general framework including the costs of combined sewer overflows (CSO) and water quality. The cost may also be a factor of other process model outputs such as deviation from target water levels at certain points (Schütze et al., 2004).

The third component of MPC for a urban drainage system is an optimization routine to determine the best control policy for the system. Using the process model to assign a cost to a given control policy, the optimization procedure seeks to find the control policy that incurs the smallest cost. If the process model is linear, a true optimum can be found using traditional optimization procedures like simplex (Nelder and Mead, 1965). If the process model is non-linear, other approaches must be taken such as using a metaheuristic to find an effective control policy (Gandomi et al., 2013).

In summary, the chronological workflow for MPC for a urban drainage system is: 1) system states are read from the physical system, 2) using the system states as initial conditions and future disturbances as input, a process model is used to evaluate control policies, 3) the best control policy is selected through an optimization procedure, and 4) the best control policy is implemented in the real system. Although the best control policy is obtained for the entire control horizon, only the first step in the control policy is used since the procedure re-optimizes at every control time step.

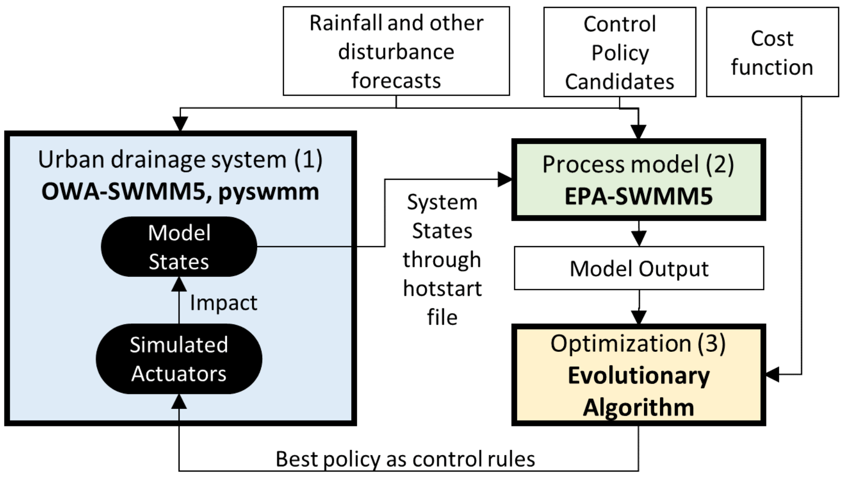
Rainfall and other disturbance forecasts
Control Policy Candidates
Cost function
Urban drainage system (1)
OWA-SWMM5, pyswmm
Model States
Impact
Simulated Actuators
System States through hotstart file
Process model (2)
EPA-SWMM5
Model Output
Optimization (3)
Evolutionary Algorithm
Best policy as control rules

Figure 1: Main components of MPC in swmm_mpc system

2.2 MPC for SWMM5: swmm_mpc

In this section we describe the implementation of the parts of MPC using Python and SWMM5. This implementation was done in the swmm_mpc Python package and uses the standard EPA-SWMM5 and an enhanced version of SWMM5 developed by Open Water Analytics, OWA-SWMM5. The software simulates online MPC for an urban drainage system using SWMM5 as the process model and as the simulated physical system. The current system could also be used in an offline mode where a control policy for a forecast storm event is found beforehand.

2.2.1 Simulated urban drainage system: OWA-SWMM5 and pyswmm

SWMM5 was used to simulate the physical urban drainage system. For this, an enhanced version of SWMM5, OWA-SWMM5 (https://github.com/OpenWaterAnalytics/Stormwater-Management-Model), was used via an accompanying Python library, pyswmm (https://github.com/OpenWaterAnalytics/pyswmm). Both OWA-SWMM5 and pyswmm were developed and are distributed by Open Water Analytics. Compared to EPA-SWMM5, OWA-SWMM5 contains additional C functions that are accessed by pyswmm.

OWA-SWMM5 and pyswmm provide three key functionalities needed to simulate the online optimization procedure required by MPC. First, unlike when a simulation is run via

EPA-SWMM5, when using pyswmm, custom Python routines can be executed between each time step of the simulation. This is critical to swmm_mpc because at each time step in the workflow three processes occur: 1) the states from the simulated urban drainage system need to be read and transferred to the process model; 2) the metaheuristic needs to be run; and 3) the best policy found by the metaheuristic needs to be implemented in the simulated urban drainage system. Using pyswmm, Python code can be run to perform each of these processes at each control time step.

Second, pyswmm enables the transfer of system states at each time step from the simulated urban drainage system to the process model. This is accomplished through a hotstart file. A SWMM5 hotstart file contains all of of the hydraulic and hydrologic states of the model at the time in the simulation when the hotstart file is saved. When a hotstart file is read into a simulation, that simulation's initial hydraulic and hydrologic states are the states represented in the hotstart file. This functionality is well-suited to transfer the states of the simulated urban drainage system to the process model in the swmm_mpc workflow.

Using EPA-SWMM5, a hotstart file can be saved only at the end of a simulation. This is a critical limitation because in MPC the system states need to be transferred at every time step. To address this limitation, we added new functionality to OWA-SWMM5 and pyswmm to enable hotstart files to be saved at any point in a SWMM5 simulation executed using pyswmm. This functionality allowed the system states of the simulated urban drainage system to be transferred to the process model at each time step.

Third, through pyswmm the best control policy found by the metaheuristic can be implemented at each time step. This is done using pyswmm to change the settings of the actuators in the model during the simulation. When a simulation is initialized in pyswmm, each object in a SWMM5 model (every node, link, subcatchment, etc.) can be read into a Python object via its element ID as defined in the SWMM5 input file. Each of these Python objects has attributes that can be read (e.g., depth at a node and flow in a link). Actuators in the model read into Python objects also have the "target_setting" attribute that can be written. To implement a control setting for an actuator via pyswmm, its "target_setting" is set to the first setting in the best control policy.

2.2.2 Process model: EPA-SWMM5

In addition to representing a real urban drainage system, SWMM5 was used as the process model. However, in contrast to using OWA-SWMM5 to simulate the physical urban drainage system, the standard EPA-SWMM5 was used as process models. This was necessary because the current version of pyswmm cannot run more than one simulation at a time. This is a functionality needed in swmm_mpc because at each time step during the simulation of the urban drainage system, at least one process model simulation is run in a predictive fashion to evaluate control policy candidates. EPA-SWMM5, unlike pyswmm, supports multiple simulations being executed simultaneously.

2.2.3 Active controls in EPA-SWMM5

EPA-SWMM5 simulates the active control of certain hydraulic structures including pumps, orifices, and gates. Each of these structures has a setting that can be assigned. For example, the setting for an orifice is a decimal number between 0 and 1 which corresponds to the percent open of the orifice (e.g., a 0.5 setting would mean the orifice was 50%). The user can also define an amount of time for a structure to implement a change in setting. This "time to change" parameter in EPA-SWMM5 represents the delay seen in reality for changing an actuator's setting.

Changing controls during an EPA-SWMM5 simulation is done using one or more control rules (see example in Figure 2). A control rule is specified in the SWMM5 input file before the simulation begins and consists of four parts. The first two parts of a control rule are the rule name and the condition. In the example, the rule name is "R1". The condition is "IF NODE J1 DEPTH <2", meaning that the program will check if the depth at the node with the ID of "J1" is less than 2 (the units being defined globally in the model input file as feet or meters). In EPA-SWMM5 the condition can be the state at any link or node and can also be related to global simulation states such as the model simulation time. The third part of a control rule defines which structure(s) should change if the specified condition is met. In the example the structure that will change is "ORIFICE R1" (note that the name of the rule

```
Rule R1
If NODE J1 DEPTH < 2
THEN ORIFICE R1 SETTING = 0.6
```

Figure 2: Example of a control rule in SWWM5

```
Rule R1
If SIMULATION TIME < 0.25
THEN ORIFICE R1 SETTING = 0.2

Rule R2
If SIMULATION TIME < 0.50
THEN ORIFICE R1 SETTING = 0.4

Rule R3
If SIMULATION TIME < 0.75
THEN ORIFICE R1 SETTING = 0.5

Rule R4
If SIMULATION TIME < 1.0
THEN ORIFICE R1 SETTING = 0.2
```

Figure 3: Example implementation of control policy as set of control rules

and of the orifice are the same coincidentally and can be different). Finally, the fourth part of the rule defines the setting to which the structure should change. In the example, this is "0.6", meaning that if the condition is met, the orifice should be set to 60% open.

In swmm_mpc a control policy is a time series of control settings (one control setting per control time step for the control duration). This is implemented in EPA-SWMM5 as a set of control rules. In the current version of swmm_mpc, only the control of orifices is supported while support for other controls such as pumps can be added in future versions. Since a control policy in MPC is a time series, each control rule's condition is based solely on the model's simulation time in decimal hours. For example, Figure 3 shows a control policy of four settings (0.2, 0.4, 0.5, and 0.2) for "ORIFICE R1" at a 15-minute control time step implemented as control rules. This text would be written to the EPA-SWMM5 process model input file under the "CONTROLS" heading.

2.2.4 Metaheuristic: evolutionary algorithm

Because we used EPA-SWMM5 as a black-box process model, a metaheuristic was used in place of a true optimization procedure to find an effective control policy at each time step in the MPC run. We chose an evolutionary algorithm (EA) for the metaheuristic since it has been shown to be successful in other urban drainage control applications (Zimmer et al., 2015, 2018) and it's inherent propensity for parallelization Maier et al. (2014). An EA begins with an initial population of individuals where, in our case, each individual is a control policy. A fitness score (or conversely a cost) is assigned to each individual in the population and certain individuals are selected to survive into the next generation based on their fitness score. Mechanisms for improving the fitness of the individuals from one generation to the next mimic natural processes including cross-over and mutation (Maier et al., 2014). The process of selection and improvement is repeated from generation to generation until a stopping criteria is met. Common stopping criteria include a user-defined number of generations or an acceptably low rate of improvement from one generation to the next. The use of an EA requires several user-defined parameters including the number of individuals in the initial population, the cross-over rate, the mutation rate, and the stopping criteria.

Since the EA searches for the policy that incurs the minimum cost, the way in which a cost is assigned to each individual control policy is very influential on the EA's effectiveness. In swmm_mpc, the cost of a control policy is determined using the process model and a cost function. First, each individual control policy is implemented in the process model input file as a set of control rules as described above. Once the control policy is implemented, the EPA-SWMM5 model is executed. Elements of the model output resulting from the process model execution become input for the cost function. The cost function used in swmm_mpc is

$$Cost = \alpha(\boldsymbol{a} \cdot \boldsymbol{v}) + \beta(\boldsymbol{b} \cdot \boldsymbol{d}) \tag{1}$$

where $\boldsymbol{a}$, $\boldsymbol{v}$, $\boldsymbol{b}$, $\boldsymbol{d}$ are each 1-dimensional vectors, and α and β are scalers. The members of $\boldsymbol{a}$ are user-defined weight values for flooding at any node in the system and the members of $\boldsymbol{v}$ are the magnitude of flooding at each node as calculated by the process model. The members of $\boldsymbol{b}$ are user-defined weights for deviation from user-defined target water levels at each node in the system and the members of $\boldsymbol{d}$ are the average absolute deviations from target water levels again as calculated by the process model. α, and β, are user-defined constants used to give overall weights to flooding costs compared to deviation costs.

We intentionally made this cost function flexible so that users can customize it to meet their objectives which may vary between use cases. A cost for flooding is obviously important as that is a major concern for many communities and the prevention of which is one of the main purposes for urban drainage systems. We also included a cost from deviations for target water levels because, in certain cases, it is desirable to maintain water levels close to a certain depth. For example, it may be important to keep a certain amount of water in a retention pond for aesthetic and/or ecological purposes. Although the cost function is flexible, when implemented in swmm_mpc, the user need only define what is important to the specific application. For example, default for $\boldsymbol{a}$ is a vector of all 1's. When one node is specified, the weight of any unspecified node becomes zero. The default for $\boldsymbol{b}$ is all zeros, since the user has to specify a target depth for a given node.

To execute EAs we used the Distributed Evolutionary Algorithms for Python (DEAP)(https://github.com/DEAP/deap) library. An advantage of EAs is that they can easily be run in parallel since they performs many independent evaluations (Maier et al., 2014). In DEAP parallel processing is supported through integration with the built-in "multiprocessing" Python library.

2.2.5 swmm_mpc Workflow

The MPC workflow in swmm_mpc was implemented using three main Python functions (see Figure 4). The function in the workflow called by the user is "run_swmm_mpc." This function runs the MPC workflow and calls the two other main functions. "run_swmm_mpc" takes 13 user inputs as shown in Table 1. Through these inputs, the user specifies the model input file to that represents the urban drainage system, the control inputs (i.e., which controls to find a policy for, the control time step, and the control horizon), and EA parameters (e.g., number of generations, cost function parameters).

The most complex of the user-supplied arguments are "target_depth_dict" and "node_flood_wgt_dict" (see Snippet 1 for examples). These two arguments define the $\boldsymbol{a}$ and $\boldsymbol{b}$ variables in the cost function. Additionally, the "target_depth_dict" argument is used to determine $\boldsymbol{d}$. These arguments map from Python data structures to the mathematical variables in the cost function. The "target_depth_dict" argument is a dictionary whose keys are node ids and whose values are dictionaries. The inner dictionary has two keys, the target depth of the node and the weight of the cost for deviations from the weight at the node. In Snippet 1, the "target_depth_dict" specifies that the target depths of Nodes St1 and St2 are 4.0 and 3.5, respectively. The weights are also specified: deviation from the target depth at Node St1 will be twice as costly as deviation from Node St2. The "node_flood_wgt_dict" is a simpler dictionary, the keys of which are node ids and the values are weights. In Snippet 1, the "node_flood_weight_dict" specifies that flooding at Node J3 is five times costlier than flooding at Node St1. Note that if one or more node is included in the "target_depth_dict" or the "node_flood_wgt_dict", other nodes are not included in the cost calculation (in terms of the cost function, the corresponding weights in $\boldsymbol{a}$ and $\boldsymbol{b}$ are zero). This is shown in Snippet 1, the weight of deviations from a water level at Node J3 and the weight of flooding at Node St2 would both be zero since they are not included in the dictionaries.

Snippet 1: Examples of "target_depth_dict" and "node_flood_wgt_dict"

```
target_depth_dict = {"Node St1": {"target": 4.0, "weight": 2},
                     "Node St2": {"target": 3.5, "weight": 1}}

node_flood_weight_dict = {"Node J3": 1, "Node St1": 0.2}
```

In the "run_swmm_mpc" function, the SWMM5 model simulating the urban drainage system is run step by step via pyswmm. At the beginning of the simulation, the SWMM5 input file representing the urban drainage system is copied. This copy serves as the input file used for the process model. To ensure that the states and simulation periods of process model remain in sync with the simulated urban drainage system, at each time step a hotstart file from the urban drainage system simulation is saved and then used as the initial states for the process model. The process model's simulation start date and time are also updated to match the urban drainage system simulation's current date and time.

Once the process model's simulation start date and time is same as the simulated urban drainage system and the hotstart file of the simulated urban drainage system is saved, the "run_ea" function is called. The "run_ea" function initiates the EA which starts by creating an initial population of individual control policies. In our case, an individual is a 1-dimensional vector, each member of which is a setting for an individual actuator for one control time step. The initial population for the first time step is a group of random individuals. For subsequent time steps, elitism is used where the best policy found in the previous time step is used to seed the initial population of the current time step.

The control policies initiated in the "run_ea" function are input into the third main function, "evaluate". The evaluate function makes a copy of the process model input file and the input hotstart file. To avoid file naming conflicts, a random string is appended to the hotstart and input file names. The control policy is then implemented in the newly created input file by adding corresponding control rules. Once the control policy is implemented in the input file, the simulation is executed with EPA-SWMM5. When the simulation run is completed, the "evaluate" function parses the output file to determine $\boldsymbol{v}$ and $\boldsymbol{d}$ in the cost function. The policy's cost can then be determined since the remaining cost function parameters (α, $\boldsymbol{a}$, β, and $\boldsymbol{b}$) are user-defined. The evaluation of an individual control policy is independent of all others, therefore, the "evaluate" function is the part of the workflow that is parallelized through Python's "multiprocessing" module.

Using the cost that has been assigned to each policy, the "run_ea" function selects the best individuals to retain in the population of policies for the next generation. After the user-defined number of generations are complete, the best policy found by the EA is implemented in the simulated urban drainage system in the "run_swmm_mpc" function. The policy is also saved and used to seed the population of the next time step. Finally, the setting for that time step is recorded so that at the end of the simulation, the best control policy for the entire simulation time is saved.

2.3 System demonstration

To demonstrate the utility and functionality of swmm_mpc, three control scenarios were used and compared for two cases in which the demonstration model and the cost function param-

Table 1: User inputs for "run_swmm_mpc" function

Parameter name	Data type	Description	Default value
inp_file_path	String	File path to SWMM5 input file (.inp)	N/A
control_horizon	Number	Control horizon in hours	N/A
control_time_step	Number	Control time step in seconds	N/A
control_str_ids	List of strings	IDs of control structures to be adjusted	N/A
work_dir	String	Path to directory where temporary files will be stored	N/A
results_dir	String	Path to directory where results should be stored	N/A
target_depth_dict	Dictionary	IDs of nodes and corresponding target depths and relative penalty weights	Null
node_flood_weight_dict	Dictionary	IDs of nodes and corresponding relative penalty weights	Null
flood_weight	Number	Overall weight of flood penalties	1
dev_weight	Number	Overall weight of deviation penalties	1
ngen	Number	Number of generations that the GA should perform	7
nindividuals	Number	Number of individuals in the inital GA population	100
run_suffix	String	suffix to be appended to results file	

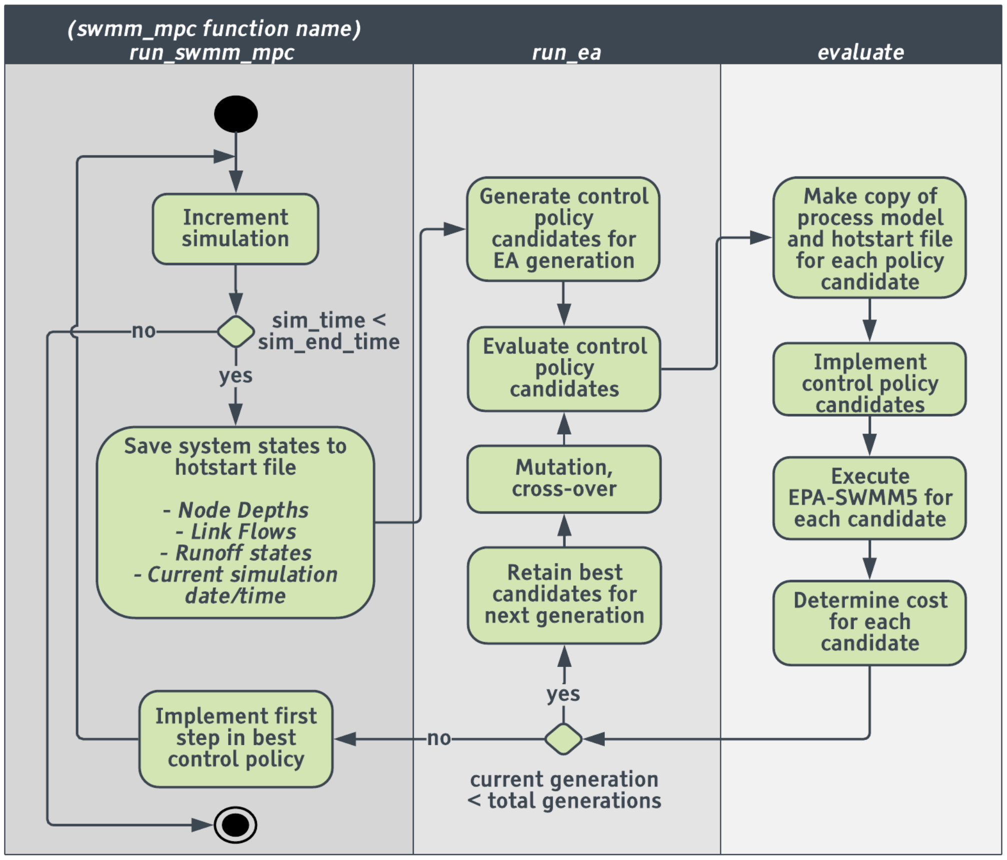
(swmm_mpc function name)
run_swmm_mpc
run_ea
evaluate
Increment simulation
sim_time < sim_end_time
no
yes
Save system states to hotstart file
- Node Depths
- Link Flows
- Runoff states
- Current simulation date/time
Implement first step in best control policy
Generate control policy candidates for EA generation
Evaluate control policy candidates
Mutation, cross-over
Retain best candidates for next generation
yes
no
current generation < total generations
Make copy of process model and hotstart file for each policy candidate
Implement control policy candidates
Execute EPA-SWMM5 for each candidate
Determine cost for each candidate

Figure 4: Activity diagram of MPC implementation using Python

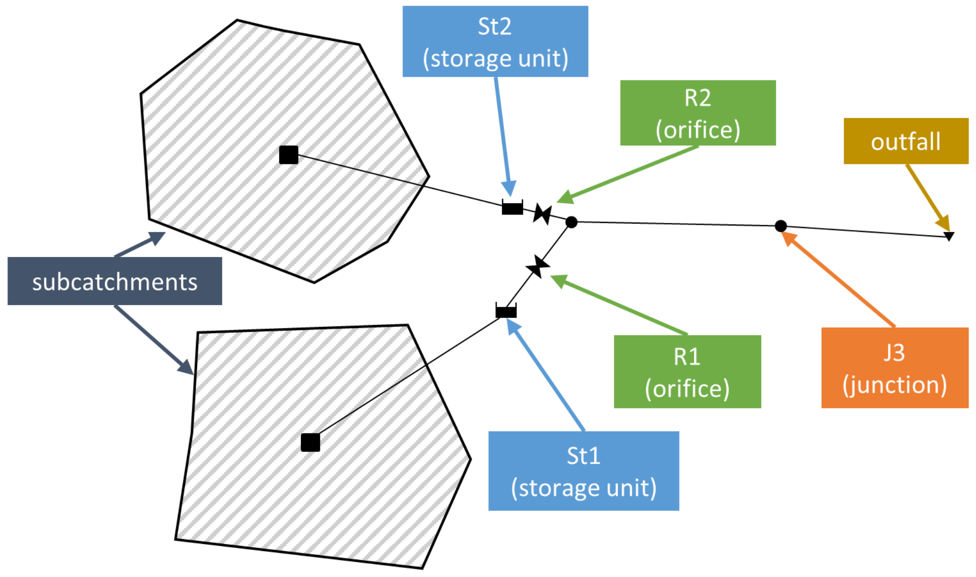

Figure 5: Demonstration model schema

eters differed slightly. The computational costs of running swmm_mpc were also quanitfied.

2.3.1 Demonstration model and rainfall event

The model used to demonstrate swmm_mpc (see Figure 5) has two orifices (R1 and R2) that control the flow out of two storage units in parallel in the system (St1 and St2, respectively). In SWMM5 models, storage units are generic and are used to represent both natural storage facilities such as a pond as well as man-made facilities such as an underground tank or retention pond. The two orifices from the storage units meet and flow through a junction, J3, before leaving the sytem through the outfall. For the example use case we used a 2-year, 12-hour rainfall event for Norfolk, Virginia, a coastal city that experiences frequent flooding (Mitchell et al., 2013). The rainfall event (see Figure 6) had 78.2 mm of total rainfall (Bonnin et al., 2018) with an SCS Type II temporal distribution (Mockus, 2012). The model simulation time was 24 hours.

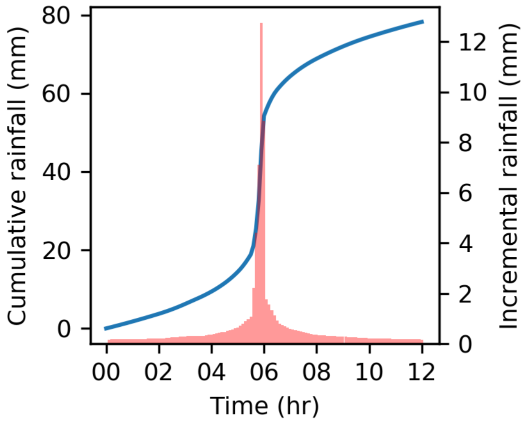

Figure 6: Design storm input to use case study

2.3.2 Control scenarios

Scenario 1: Passive In this scenario there is no active control with the outlets 100% open. Two outlets to the storage units were simulated, one at the bottom of the storage units, the other is near the top. Water continously leaves the retention pond through the lower outlet, and when the water level reaches a certain depth, the water flows out of the upper outlet as well to avoid overtopping.

Scenario 2: Rule-based control For this scenario, a simple logical rule controls the orifice openings and therefore the discharge from the storage units. In practice, such rules can be based on experience and knowledge gained by local stormwater personnel over time. Although, heuristic-based rules alter a dynamic, actuated system, the rules themselves are static, meaning that they do not change for the duration of an event. Furthermore, the rules do not adjust based on forecast conditions. The rule we chose was to set the percent open of the orifices equal to the percent full of the storage units. For example, if one storage unit were 30% full, the valve at the outlet would be set to 30% open. This rule was intended to minimize flooding by retaining some water so as not to flood the downstream nodes, but release enough water to avoid overtopping.

Scenario 3: MPC The MPC control policy was found using swmm_mpc as described in Section 2.2.5 above. One advantage of MPC over the rule-based control is the ability to adjust the actuators based on forecast conditions. For the use cases, we used a control time step of 15 minutes and a control horizon of one hour. Therefore, with two controls, a single control policy consisted of a vector of eight values (2 controls x 4 control steps per hour x 1 hour).

2.3.3 Model and cost function cases

We implemented and compared the three control scenarios for two cases in which model and cost function parameters were slightly different (see Table 2). For the passive and rules-based control scenarios, there is only one difference between Case 1 and Case 2. In Case 1, the maximum depth of the storage units is 1.52 m (5.00 ft) while in Case 2 the maximum depth is 1.37 m (4.50 ft). The maximum depth of the storage units was reduced in Case 2 in order to explore the effectiveness of the three scenarios in a more constrained situation.

For the MPC scenario, there is a difference in the cost function parameters between Cases 1 and 2. In Case 1, there are two objectives: 1) minimize flooding at the downstream Node J3, and 2) minimize deviations from a target water level of 1.22 m (4.00 ft) at the storage units. These objectives translated to the $\boldsymbol{a}$ and $\boldsymbol{b}$ parameters of the cost function as a vector of zeros except the last member (Node J3), and a vector of zeros except for the first two members (St1 and St2), respectively. In this case, to emphasize minimizing flooding at Node J3 over minimizing deviations from the target water depths at the storage units, we set the cost of flooding weight, α, six times larger than the cost of deviations, β (3 compared to 0.5). In Case 2, there was only one objective, minimize flooding at Node J3. In this case, $\boldsymbol{a}$ is the same as in Case 1, but β is zero since minimizing deviations from a target water level is not part of the objective.

2.3.4 Use of parallel, high-performance, and cloud computing

The EA used for selecting the best control policy is computationally expensive and therefore, some analysis of computational costs for executing the swmm_mpc workflow was performed.

Table 2: Differences in MPC cost function parameters and storage capacity between two test cases

	Case 1	Case 2
α	3	1
$\boldsymbol{a}$	[0, 0, 1]	[0, 0, 1]
β	0.5	0
$\boldsymbol{b}$	[1, 1, 0]	N/A
Storage depth at St1 and St2 (ft)	1.52 m (5.00 ft)	1.37 m (4.50 ft)

Table 3: Specifications of computational resources used for demonstration model

	PC	HPC	GCP
Max. # cores	8	28	64 (tested up to 32)
CPU speed	3.60 GHz	2.4 GHz	2.0 GHz
Processor type	Intel i7	Intel Xeon	Intel Xeon
RAM	16 GB	128 GB	7-120 GB (depending on # of CPUs)

The wall-clock times for Case 1 (the more complex of the two cases) were compared when using a typical personal computer (PC) and the University of Virginia's high-performance computing (HPC) system, Rivanna. Recogizing that many (likely most) municipalities will not have HPC resources available to them, we also explored the use of a commercial cloud computing service for running swmm_mpc. These services, such as Amazon Web Services, Google Cloud Platform, or Microsoft Azure, allow users to rent large, powerful computers, charging only for the time that the computers are being used. To explore the option of renting a cloud-based machine, we also executed Case 1 through Google Cloud Platform (GCP). The number of cores available, RAM, and processor speeds of the PC, HPC, and GCP machines are listed in Table 3. Case 1 was run with varying number of cores on each platform.

3 Results and Discussion

3.1 Results from Case 1 and Case 2

Figure 7 shows the results from the three control scenarios for Case 1. In the rules-based and MPC scenarios, the control policies kept the valves closed more, thus retaining more water in the storage units and preventing flooding at Node J3 (see Figures 7 A and B). The water level at St1 reaches much higher values in Scenarios 2 and 3 (max of 1.45 m (4.76 ft) and 1.38 m (4.54 ft)) compared to Scenario 1 (max of 1.26 m (4.15 ft)). In this case, both rules-based control and MPC practically eliminated flooding while in the passive scenario, flooding occurred (Figure 7D).

In addition to practically eliminating flooding in Case 1, the swmm_mpc control policy was able to maintain the water levels in St1 and St2 near the target of 1.22 m (4.00 ft). Because the cost of flooding was weighted more heavily than the cost of deviating from target water levels at St1 and St2, the storage units allowed the water levels to exceed the target depth. Following the peak of the storm, and therefore the largest risk of flooding, the algorithm could focus on the less-weightier objective of minimizing deviations from the target water levels. Therefore, following the peak of the storm, water was released from the storage units until the level reached near the target water level. The final water levels of St1 and St2 were 1.20 m (3.94 ft) and 1.17 m (3.83 ft), respectively.

Figure 8 shows the results for Case 2, in which less storage volume is available than in Case 1. In Case 2, the rules that completely eliminated flooding in Case 1 were not effective and in fact resulted in more flooding than the passive scenario. In both Case 1 and 2, the flooding in the passive scenario occurred at Node J3, downstream from the storage units. By contrast, in Case 2 the flooding in Scenario 2 occurred at the storage units. In this instance, the rules held back too much water so when the peak of the storm arrived, the storage units overtopped. As with Case 1, the policy found by swmm_mpc, was effective at practically eliminating flooding. This illustrates that there are conditions in which one set of simple rules is effective at achieving an objective and other conditions in which it is not. In Case 2, with less storage available, the more sophisticated, more computationally expensive control policy found through swmm_mpc was needed to achieve the objective of minimizing flooding.

In both Case 1 and Case 2, a major difference between the rules-based and the swmm_mpc scenarios is the smoothness that is seen in the control policy. The swmm_mpc control policies changes the valve position and therefore the upstream and downstream water depths much more abruptly than the rules-based policies. This abruptness could be smoother if the

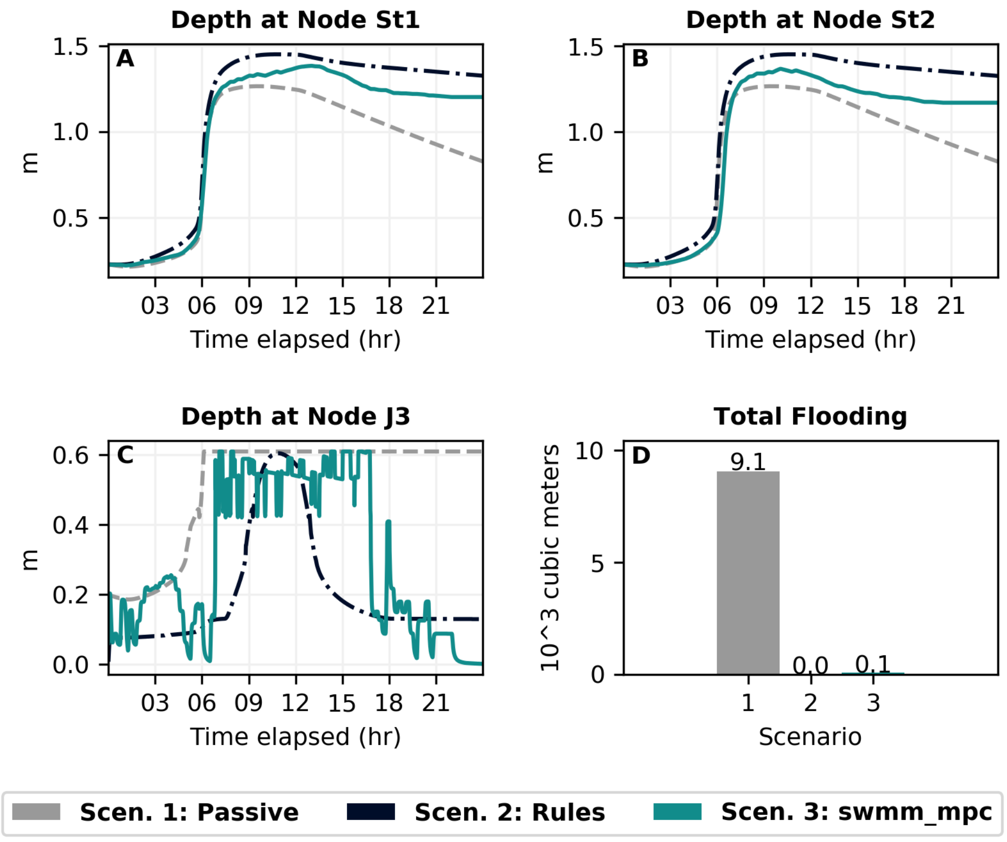

Figure 7: Depth and flooding at system nodes for Case 1

time for the orifices to implement a change in setting was increased. In our demonstation model these times were zero meaning that the orifice settings were changed instantly.

3.2 Computational cost of swmm_mpc

Figure 9 shows the wall-clock times for executing swmm_mpc for Case 1 on a PC, an HPC, and GCP machines with a varying number of processing cores. The simulation had 96 control time steps (15 minute resolution for 24 hours). If used for online MPC in a real case, the wall-clock time required for one time step would need to be less than the time step itself, otherwise, the setting for the next time step would not be determined before it would need to be implemented.

The fastest wall-clock time using the PC was 89.4 minutes using eight computational cores (the maximum available). Therefore the time required to find the best control policy

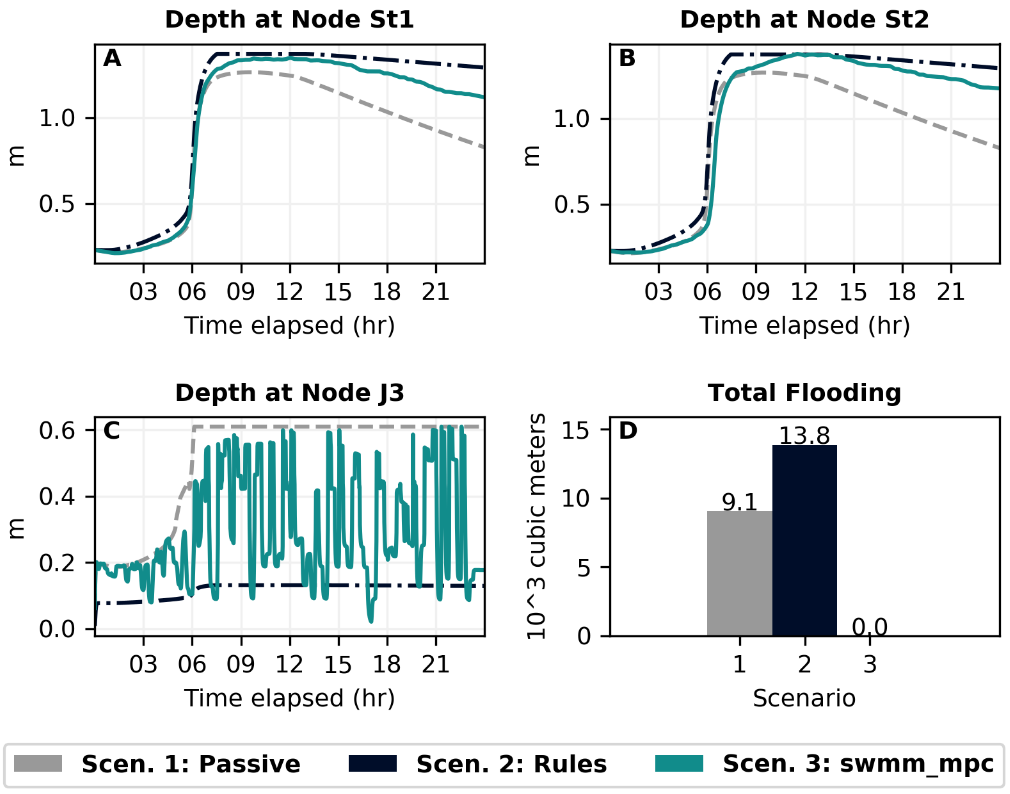

Figure 8: Depth and flooding at system nodes for Case 2

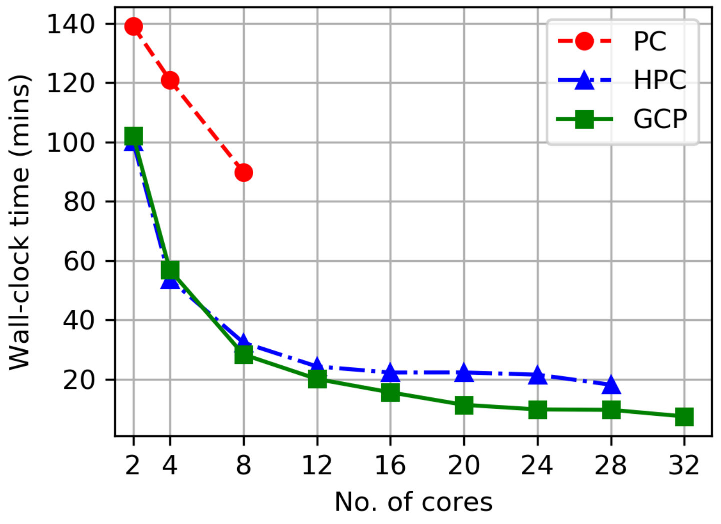

Figure 9: Wall-clock run times for Case 1 with varying number of computational cores using a PC and a high performance computer.

at each control time step was 0.93 minutes. In this case, the PC's computing power was sufficient (0.93 minutes per time step compared to 15 minute time step). For the HPC, the best case scenario was a wall-clock time of 18.2 minutes (0.19 minutes per time step) using the maximum of 28 computational cores. Although the minimum wall-clock time was achieved using all 28 cores on the HPC, the improvement in wall-clock time when increasing the number of cores past 16 was minimal. This is a relevant consideration when using a shared HPC resource where requesting more computational cores likely corresponds to a longer wait in the job queue.

The wall-clock times using GCP were much lower than the PC or HPC. In the best case, 32 vCPUs and 120 GB of RAM were used for a wall-clock time of 7.47 minutes (0.08 minutes per time step). This is a 2.4x speed-up compared to the fastest run using the HPC and almost a 12x speed-up compared to the fastest PC run. The financial cost of renting this machine was $1.71 per hour. The GCP hardware is newer which may explain why the wall-clock time is lower even when the same number of computational cores was used.

3.3 Pratical considerations

3.3.1 Computational costs

The execution times for our example use case were viable, however, a more complex model, a smaller control time step, or different EA parameters (more generations or individuals per generation) would increase the execution time. For example, the demonstration SWMM5 model required only one second or less to execute. The swmm_mpc workflow executed that model thousands of times. In our cases, the model was run more than 30,000 times (24-hour simulation x 4 time steps per hour x 5 generations per time step x approx. 70 individuals per generation) therefore requiring approximately 30,000 seconds of computation time (if each model takes approx. 1 second to run). If a more sophisticated SWMM5 model instance were used, the execution time would be much higher. For example, in related research we are using a more complex model of the stormwater infrastructure for a neighborhood in Norfolk, Virginia that requires close to 60 seconds to execute for a 24-hour simulation time period. The wall-clock time for swmm_mpc for this more complex model would increase by around a factor of 60 compared to the simple cases demonstrated here. Assuming a linear increase, the wall-clock time would be 55.8 minutes/time step using the PC, thus rendering it unfeasible for running on a PC. Again assuming linear scaling, using 32 cores on GCP, the same simulation would execute at a rate of 4.8 minutes/time step. Using this setup, the wall-clock time for a 24-hour simulation with a time step of 15 minutes would be approximately 7.68 hours.

Another factor to consider for practical use of swmm_mpc is the control horizon and the number of control structures whose policies will be found using swmm_mpc. These two parameters determine the size of the overall control policy and therefore the solution space that the EA will be searching. In our example use case, the control policy was a vector of eight integers between 0 and 10. Therefore, there were 11^8 possible solutions. This solution space, already large, would double if the control time step were 7.5 minutes instead of 15, or if the control horizon were two hours instead of one. A larger solution space would result in a larger computation time to reach an effective solution.

Given the computational cost of the current swmm_mpc approach, the required complexity (and thus the wall-clock time) of the process model is an important consideration. A scenario with a simple process model (approx. 1 second wall-clock time) can be feasibly executed with just a PC, as shown in the system demonstration. However, a simple model may not represent complex urban drainage systems well enough to produce an effective control policy. Defining what level of detail is sufficient in the process model may be difficult, however, as it may depend on the objective of the modeling, a certain storm event, or the system itself. There is a tradeoff among 1) model complexity, 2) model runtime, and 3) the model's ability to effectively simulate the relevant parts of the system. This tradeoff is very relevant to the use of process models in a receding control horizon approach such as MPC and needs further research.

If a more complex model is needed, municipalities or others needing cloud-based resources to run swmm_mpc must consider the financial cost of renting a machine. Using GCP, the cost of finding the control policy for the 24-hour time span in Case 1 in our system demonstration was very low, \$0.21. This was, however, for one simple case. The cost would be higher with more complex scenarios such as a more complex model, a shorter time step, or more controls. If we assume the use of a more complex model, which takes 60 seconds to run, would increase running time by a factor of 60, that would also increase the cost by a factor of 60 to \$12.60. However, the system would not only run if there were a storm in the forecast.

Given the computational cost of running the evolutionary algorithm, other, more efficient alternatives should be explored in future research. One possible alternative that is reinforcement learning (Kaelbling et al., 1996). This approach may be able to converge to a solution more quickly than an evolutionary algorithm and thus reduce runtimes. Another future improvement could be adding a penalty to changing actuator states and/or using another dynamic optimizer to have a less erratic behavior in the actuators.

3.3.2 Data and modeling uncertainty

The current design of swmm_mpc does not take into account the uncertainties in the system states,the forecast data and the process model. Because the SWMM5 engine is used to simulate both the urban drainage system and the process model, the process model assumes 1) perfect knowledge of the urban drainage system states, 2) perfect knowledge of future disturbances, and 3) perfect modeling of the urban drainage system. In a real implementation, there would be significant uncertainties in each of these aspects. In a real implementation, knowledge of the system states is available only from a limited number of sensors in the system. This data, limited in spatial and temporal resolution, would need to be interpolated, and likely extrapolated, to set the all the states in the system. More work will need to be done to investigate ways of incorporating sensor values to set the process model's initial conditions. Additionally, in the current case, the future disturbances (i.e., primarily rainfall) are known perfectly, when in reality, there is a large amount of uncertainty involved with forecasting such disturbances (see for example, Hong and Pai (2007), Valverde Ramírez et al. (2005), and Bellon and Austin (1984) regarding uncertainty in forecasting rainfall).

In addition to data uncertainties seen in reality, swmm_mpc does not currently consider gaps between the *simulated* behavior through the SWMM5 process model and the *actual* behavior of the urban drainage system, but assumes that simulation and reality are the same. In actuality, the gap between simulation and reality in urban drainage systems can be significant (see for example Mark et al. (2004)). On a related note, the ability for swmm_mpc to find a control policy that is effective for the urban drainage system is directly related to how well the process model represents the system. Given the simulation and data gaps seen in reality, the simulated results through policies found by swmm_mpc should be considered as the best case scenario and if the same policies were used in practice, any effects should be expected to be seen to a lesser extent. Further research is needed to determine the degree to which the results from the policies implemented in reality will differ compared to the simulation results.

4 Conclusions

A free and open-source software package, swmm_mpc, was developed which computes a control policy for controls within an urban drainage system model. The widely-used United States Environmental Protection Agency Stormwater Management Model Version 5 (SWMM5) is used to simulate the urban drainage system and the process model. A third-party Python library, pyswmm, is a critical component of the swmm_mpc workflow allowing a SWMM5 model to be run step-by-step in a Python environment. An evolutionary algorithm was used to find an effective control policy at each time step. When tested using a simple SWMM5 model, the swmm_mpc software was able to produce control policies that met objectives including minimizing flooding and minimizing deviation from target water levels at certain nodes in the system.

swmm_mpc leverages parallel computing to run the computationally expensive evolutionary algorithm more quickly. The wall-clock time for a simple SWMM5 model for a 24-hour simulation was reduced from 139 minutes to 89.4 minutes when the computational cores on a desktop PC were increased from two to eight. The wall-clock time was reduced even further to 18.2 minutes on a 28-core high-performance computer and to 7.47 minutes on a 32-core machine rented through the Google Cloud Platform. Parallel computing makes swmm_mpc feasible for use in real-time control with complex process models.

As the average storm intensity is projected to increase, and sea levels are expected to continue to rise, cities globally and especially on the coasts, can expect more flood conditions. Active control of urban drainage systems may be one of an array of approaches that can be used to confronting these challenges. The swmm_mpc software we have developed can be used, built-from, and improved upon as a tool to assist decision-makers and researchers in finding effective control policies for urban drainage systems.

5 Acknowledgments

We also acknowledge the support of awards 1735587 (Critical and Resilient Infrastructure Systems - CRISP) and 1737432 (Smart and Connected Communities - S&CC) from the United

States National Science Foundation. We thank Bryant McDonnell, one of the main developers of pyswmm, for his assistance in adding the hotstart file functionality to OWA-SWMM5 and pyswmm. We also thank the consultants of the University of Virginia Graduate Writing Lab including Kelly Cunningham, its director, for their helpful feedback in preparing the manuscript.

6 Software Availability

The swmm_mpc software is open-source and available for use and improvement on GitHub at https://github.com/UVAdMIST/swmm_mpc (note to reviewers/editor - once revisions are complete, we will make a release inf the repository and include a DOI in this section). A Docker image of swmm_mpc is also available at https://hub.docker.com/r/jsadler2/swmm_mpc/. The demonstration model will also be available on HydroShare (Sadler, 2018).

References

Afram, A. and Janabi-Sharifi, F. (2014). Theory and applications of HVAC control systems – A review of model predictive control (MPC). *Building and Environment*, 72:343–355.

Bellon, A. and Austin, G. (1984). The accuracy of short-term radar rainfall forecasts. *Journal of Hydrology*, 70(1-4):35–49.

Berggren, K., Olofsson, M., Viklander, M., Svensson, G., and Gustafsson, A.-M. (2012). Hydraulic Impacts on Urban Drainage Systems due to Changes in Rainfall Caused by Climatic Change. *Journal of Hydrologic Engineering*, 17(1):92–98.

Bonnin, G., Martin, D., Lin, B., Parzybok, T., Yekta, M., and Riley, D. (2018). POINT PRECIPITATION FREQUENCY ESTIMATES, Norfolk, Virginia, USA. https://hdsc.nws.noaa.gov/hdsc/pfds/pfds_printpage.html?lat=36.8661&lon=-76.2890&data=depth&units=english&series=pds. Online; accessed 2018-10-04.

Burger, G., Sitzenfrei, R., Kleidorfer, M., and Rauch, W. (2014). Parallel flow routing in SWMM 5. *Environmental Modelling & Software*, 53:27–34.

Camacho, E. F. and Bordons, C. (2007). *Model predictive control*. Springer London Limited.

Cembrano, G., Quevedo, J., Salamero, M., Puig, V., Figueras, J., and Martı, J. (2004). Optimal control of urban drainage systems. A case study. *Control Engineering Practice*, 12(1):1–9.

Darsono, S. and Labadie, J. W. (2007). Neural-optimal control algorithm for real-time regulation of in-line storage in combined sewer systems. *Environmental Modelling & Software*, 22(9):1349–1361.

Del Re, L., Allgöwer, F., Glielmo, L., Guardiola, C., and Kolmanovsky, I. (2010). *Automotive model predictive control: models, methods and applications*, volume 402. Springer.

Gandomi, A. H., Yang, X.-S., Talatahari, S., and Alavi, A. H. (2013). 1 - metaheuristic algorithms in modeling and optimization. In Gandomi, A. H., Yang, X.-S., Talatahari, S., and Alavi, A. H., editors, *Metaheuristic Applications in Structures and Infrastructures*, pages 1 – 24. Elsevier, Oxford.

Gelormino, M. S. and Ricker, N. L. (1994). Model-predictive control of a combined sewer system. *International Journal of Control*, 59(3):793–816.

Heusch, S. and Ostrowski, M. (2011). Model Predictive Control with SWMM. *Journal of Water Management Modeling*, 19:237–247.

Hong, W.-C. and Pai, P.-F. (2007). Potential assessment of the support vector regression technique in rainfall forecasting. *Water Resources Management*, 21(2):495–513.

Huber, W. C., Rossman, L. A., and Dickinson, R. E. (2005). EPA storm water management model, SWMM5. *Watershed Modeling, CRC Press, Boca Raton, FL*, pages 339–361.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kerkez, B., Gruden, C., Lewis, M., Montestruque, L., Quigley, M., Wong, B., Bedig, A., Kertesz, R., Braun, T., Cadwalader, O., Poresky, A., and Pak, C. (2016). Smarter Stormwater Systems. *Environmental Science & Technology*, 50(14):7267–7273.

Langson, W., Chryssochoos, I., Raković, S., and Mayne, D. (2004). Robust model predictive control using tubes. *Automatica*, 40(1):125–133.

Maier, H., Kapelan, Z., Kasprzyk, J., Kollat, J., Matott, L., Cunha, M., Dandy, G., Gibbs, M., Keedwell, E., Marchi, A., Ostfeld, A., Savic, D., Solomatine, D., Vrugt, J., Zecchin, A., Minsker, B., Barbour, E., Kuczera, G., Pasha, F., Castelletti, A., Giuliani, M., and Reed, P. (2014). Evolutionary algorithms and other metaheuristics in water resources: Current status, research challenges and future directions. *Environmental Modelling & Software*, 62:271–299.

Mark, O., Weesakul, S., Apirumanekul, C., Aroonnet, S. B., and Djordjević, S. (2004). Potential and limitations of 1D modelling of urban flooding. *Journal of Hydrology*, 299(3-4):284–299.

Mayne, D., Seron, M., and Raković, S. (2005). Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224.

Meneses, E., Gaussens, M., Jakobsen, C., Mikkelsen, P., Grum, M., and Vezzaro, L. (2018). Coordinating Rule-Based and System-Wide Model Predictive Control Strategies to Reduce Storage Expansion of Combined Urban Drainage Systems: The Case Study of Lundtofte, Denmark. *Water*, 10(1):76.

Mitchell, M., Hershner, C., Julie, H., Schatt, D., Mason, P., and Eggington, E. (2013). Recurrent Flooding Study for Tidewater Virginia. *Virginia Institute of Marine Science, Center for Coastal Resources Management*, (January).

Mockus, V. (2012). Storm Rainfall Depth and Distribution. In *National Engineering Handbook*, chapter 4. Natural Resources Conservation Service.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

Neumann, J. E., Price, J., Chinowsky, P., Wright, L., Ludwig, L., Streeter, R., Jones, R., Smith, J. B., Perkins, W., Jantarasami, L., and Martinich, J. (2015). Climate change risks to US infrastructure: impacts on roads, bridges, coastal development, and urban drainage. *Climatic Change*, 131(1):97–109.

Puig, V., Cembrano, G., Romera, J., Quevedo, J., Aznar, B., Ramón, G., and Cabot, J. (2009). Predictive optimal control of sewer networks using CORAL tool: application to Riera Blanca catchment in Barcelona. *Water Science & Technology*, 60(4):869.

Qin, S. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764.

Sadler, J. (2018). swmm_mpc demonstration model. HydroShare. https://www.hydroshare.org/resource/73b38d6417ac4352b9dae38a78a47d81/.

Schütze, M., Campisano, A., Colas, H., Schilling, W., and Vanrolleghem, P. A. (2004). Real time control of urban wastewater systems—where do we stand today? *Journal of Hydrology*, 299(3-4):335–348.

Sweet, W. V. and Park, J. (2014). From the extreme to the mean: Acceleration and tipping points of coastal inundation from sea level rise. *Earth's Future*, 2(12):579–600.

Tayfur, G., Kavvas, M. L., Govindaraju, R. S., and Storm, D. E. (1993). Applicability of St. Venant Equations for Two-Dimensional Overland Flows over Rough Infiltrating Surfaces. *Journal of Hydraulic Engineering*, 119(1):51–63.

Valverde Ramírez, M. C., de Campos Velho, H. F., and Ferreira, N. J. (2005). Artificial neural network technique for rainfall forecasting applied to the São Paulo region. *Journal of Hydrology*, 301(1-4):146–162.

Vrabie, D., Pastravanu, O., Abu-Khalaf, M., and Lewis, F. (2009). Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 45(2):477–484.

Zimmer, A., Schmidt, A., Ostfeld, A., and Minsker, B. (2015). Evolutionary algorithm enhancement for model predictive control and real-time decision support. *Environmental Modelling & Software*, 69:330–341.

Zimmer, A., Schmidt, A., Ostfeld, A., and Minsker, B. (2018). Reducing Combined Sewer Overflows through Model Predictive Control and Capital Investment. *Journal of Water Resources Planning and Management*, 144(2):04017091.