A WeightedCA for Flood Modeling

Andre Vacha

22/12/2021

1 Introduction

In this project, we will propose a cellular automaton for flood simulation. We will first outline the model, before implementing the model over a study area with fascinating characteristics: the state of Kerala, India, which simultaneously receives an average of $2000m$ of rainfall during the monsoon season while the majority of its terrain is at mean sea level.

2 Model: Weighted CA

The WeightedCA model extends the CA2D model (Guidolin et al., 2016) to simulate flooding over surface terrain using a 2D CA and 'weight-based' transition rules. The model is quasi-physical, solving simple shallow-water equations while using a weighting system to speed up the simulation. The pseudocode for updating the CA is given below:

```
def update(grid: np.array(N,N,5))
    grid_copy = grid.copy()
    for i in range(N):
        for j in range(N):
            central_cell = grid_copy[i,j]

            find downstream cells relative to central_cells

            compute total available storage volume v_tot
            compute weight of each cell using v_tot
            compute inter-cellular volume using weights ic

            update cell states
                central_cell -= ic
                for each downstream neighbor:
                    neighbor += weight[i]*ic
```

The terms are fairly esoteric, and are better explained by specifying the model (Cells and Transition Rules) and a demonstration with simple examples (Basin and Dam-Basin models) below.

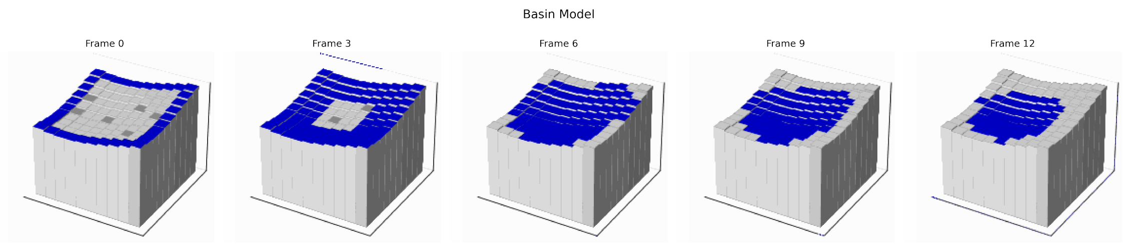

Figure 1: Toy Model 1. Basin Model. A gently sloping, concave basin with predictable flow from the edges to lowest point.

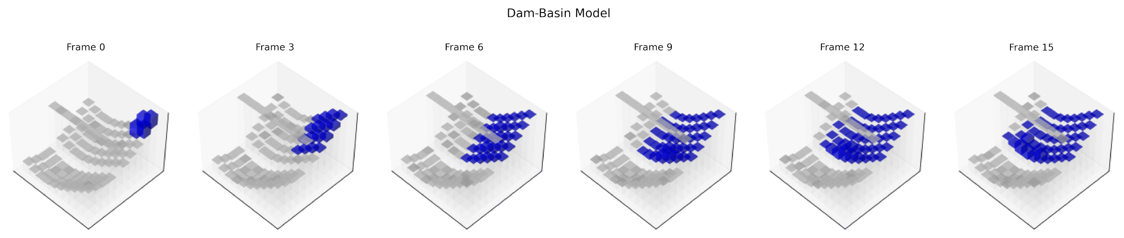

Figure 2: Toy Model 2. Dam-Basin Model. The Basin Model with a tall Dam constructed through the center. Bars are made transparent and the y-scale is adjusted to focus on the flow of water. In frame 15, water does not cross the dam.

2.1 Cells

The model takes a Digital Elevation Model, stored as a numpy array, as input. For an `(N,N)` input, the preprocessing stage involves the addition of 4 layers to an expanded representation of `(N,N,5)`:

1. Layer 1: Ground Level.

   This layer is the DEM itself, with no_data values (e.g. -9999) masked out. A DEM represents surface terrain of a landscape using a grid comprising cells, where each cell has a raster size of, say, `30m x 30m`. Data was downloaded using, in one-off excellence for a .gov website, EarthExplorer and converted to a `numpy` array using the `geotiff` module.

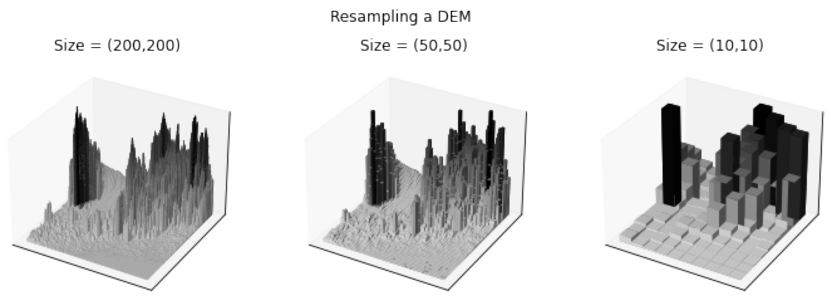

Figure 3: Re-sampling a large DEM: the southern Western Ghats, India

For simulating very large catchments `(>10,000, >10,000)`, the DEM could be re-sampled using the `resample_array()` utility function to yield a smaller, more computationally tractable grid size while retaining the overall topology. As seen in the figure above, however, aggressive re-sampling creates sheer drops between adjacent cells, which we almost never find in real-world data.

2. Layer 2: Water Columns.

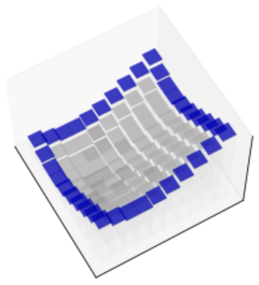

Figure 4: A ‘border’ initialization for the Basin model

This layer records the water level of each cell, relative to Ground (DEM) Level. This is the state we will update each time step in the CA. The layer can be initialized to dry cells (i.e. all cells with a water level of 0), or preexisting water levels. Precipitation can be thought of making minor additions to this layer each time step.

3. Layer 3: Slope Field.

   This records the slope (degrees) of each cell. It measured by considering the Moore neighborhood of each cell, measuring the change along the ‘row‘ and ‘column‘ directions of the 3x3 window. This is data cached to help approximate Manning flow while calculating state transition rules.

4. Layer 4: Directions.

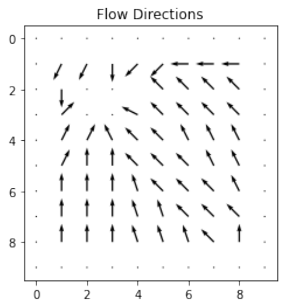

Figure 5: Flow Directions for the Basin model.

This records the directions of flow for each cell, encoded as an integer between `[0, 494]`. Water in a cell may flow to any of 8 neighboring cells, as long as they are downstream. As such, to find the directions for each cell, we first find the downstream cells, and encode each of their positions in binary. For example, a cell with no downstream neighbors would be stored as `[00000000] = 0`, while a cell with a downstream neighbor in the North and West would be stored as `[00001010] = 10`.

0	2	4
8	16	32
64	128	256

Figure 6: Position codes for Flow Directions.

5. Layer 5: Previous Intercellular Volume.

   The previous intercellular volumes for each cell are stored here. At the start of the simulation, this is initialized to 0. This layer is needed to compute the intercellular flow at time step 't+ Δ t'.

The concept of an 'Intercellular Volume' will be explained in the next section on transition rules.

2.2 State Transition Rules

We are interested in updating the water height of a central cell d_c at time $t+\Delta t$ considering the state of the central cell and its neighbors at time t, where Δt is a simulation parameter.

We first consider the height differences between a central cell c and each of its neighbors i,

$$\Delta d_{c,i} = d_c - d_i \qquad \forall i \in \{1 \cdots 8\}$$

d (m) is the water *height* at a given cell. The water height is the sum of a cell's elevation and its water column w_c at time t. Since elevation is recalled from a DEM, measured as meters from mean sea level, this can also be thought of as the water level relative to mean sea level. As such, $\Delta d_{c,i}$ measures the height difference between water levels in the central cell c and its neighbour i.

$$\Delta V_{c,i} = A \max(\Delta d_{c,i}, 0) \qquad \forall i \in \{1 \cdots 8\}$$

A (m^2) is the area of each cell. This is a constant, as each cell in the grid is a square with length (l) equal to the resolution of the DEM. Typical raster sizes are (10x10) (m^2) or (30x30) (m^2).

$\Delta V_{c,i}$ (m^3) calculates the volume we need to add to cell i such that the water levels between the central cell and the cell i are balanced. Implicating mass conservation, this is the available storage volume for the cell i.

$$\Delta V_{\text{tot}} = \sum_{i=1}^{8} \Delta V_{c,i}$$

ΔV_{tot} (m^3) sums over all $\Delta V_{c,i}$ to yield the total available storage volume across all neighbors. The largest contribution to this term will come from the largest $\Delta V_{c,i}$. Since area is constant, this coincides with the lowest downstream neighbor or, equivalently, the neighbor with the largest height difference with the central cell. We will make use of this fact in the weighting system, an algorithm for determining flow proportions to downstream neighbors.

Consider a simple physical intuition: when water flows downstream, it chooses the path of least resistance. This rules out water flowing upwards (as it loses energy in the process), so an efficient path must be one of steepest descent. In continuous space, we have a wide span of directions to choose for this path. In a CA, however, we have discretized all possible paths into a finite set of (8) directions. To compensate for this loss of granularity, we don't consider a *single* path for the flow, but rather, a series of flow proportions to downstream neighbors. These flow proportions are such that the steepest downstream neighbor receives the largest fraction, while the remaining neighbors receive smaller and smaller fractions in proportion to their lower and lower steepness. In the simplest case, consider the following height differences across 8 neighbors '[0,10,0,5,0,5,0,0]'. The flow proportions are '[0,0.5,0,0.25,0,0.25,0,0]'.

$$w_i = \frac{\Delta V_{c,i}}{\Delta V_{\text{tot}} + \Delta V_{\min}}, w_c = \frac{\Delta V_{\min}}{\Delta V_{\text{tot}} + \Delta V_{\min}}$$

In this formulation, downstream neighbours may reach heights above the central cell, which causes oscillations in the simulation (Guidolin et al., 2016). To minimize these, we permit the central cell retain a fraction of the total volume, ΔV_{min}, the minimum available volume found in neighbours. Unlike the Guidolin et al. formulation, we also set a minimum volume of (1e-4) and maximum volume (1e4) in the event that no downstream neighbor is found. This was found to reduce oscillations in the simulation dramatically.

While the weighting system alone can serve as a simple, "water balancing" transition rule, it excludes physical terms that might improve the modeling of flood dynamics. For example, there are flooding regimes where a small, local area will experience a large influx of water and become completely flooded, before the water dissipates after a short interval of time. To introduce these terms, we introduce a new quantity, total intercellular volume I_{tot} (m^3). In contrast to total available storage volume, total intercellular volume is the total outflow from the central cell over the timestep Δt and must strictly be $I_{\text{tot}} < \Delta V_{\text{tot}}$,

$$I_{\text{tot}}^{t+\Delta t} = \min(d_c A, \frac{I_M}{w_M}, \Delta V_{\text{min}} + I_{\text{tot}}^{t})$$

Of the three terms, the first one is a simple limitation: the outflow cannot exceed the total available volume in the cell ($d_c A_c$). The third term is an 'inertial' volume: we consider the transfer out of the cell in the previous iteration, and add the volume that would be transferred out of the central cell. The second term represents the maximum possible intercellular volume I_M, which flows into the downstream neighbor with the largest weight w_M,

$$I_M = v_M d_c l \Delta t$$

Here, ($v_M d_c l$) (m^3/s) computes the maximum permissible volume change in a second. l is the cell length (m), d_c is the water height (m), and v_M is the maximum permissible velocity (m/s) at the current time step. Δt (s) is the time step, which is a simulation parameter.

$$v_M = \min(\sqrt{2gd_c}, \frac{1}{n} d_c^{\frac{2}{3}} \sqrt{\frac{\Delta d_{c,M}}{l}})$$

To calculate the maximum permissible velocity, we consider a maximum theoretical velocity term $\sqrt{2d_c g}$, when water is theoretically in free fall: all its stationary Gravitational Potential Energy is converted to Kinetic Energy ($mgh = 1/2mv^2; 2gh = v^2; \sqrt{2gh} = v$). The next term is Manning's formula, an estimation for average velocity of a liquid flowing in an open channel. $\sqrt{\frac{\Delta d_{c,M}}{l}}$ computes the gradient (-) between the central cell height, while d_c is the height of the water column in the central cell. n is Manning's coefficient, which is set to 0.02, the value often used for floodplain simulations over smooth earth.

Tying everything together, we can update water columns using the following equations:

$$d_c^{t+\Delta t} = d_c^t - \frac{I_{tot}^{t+\Delta t}}{A}$$

First for the central cell's water column, we simply subtract out the total intercellular volume divided by the cell area.

$$d_i^{t+\Delta t} = d_i^t - \frac{w_i I_{tot}^{t+\Delta t}}{A} \qquad \forall i \in \{1 \cdots 8\}$$

Then, for the neighbors, we subtract out the flow proportion times the total intercellular volume, divided by the cell area.

2.3 Benchmark Model: Flow-Accumulation Matrix

While we have good intuitions about how water flows in small cases (see Toy Examples, above), we need another heuristic for how well the simulation will work in larger cases. To do so, we estimate a flow accumulation matrix.

Unlike the CA, this simulation does not have a time component, and does not solve any equations. Instead, we are using information from the Direction Layer to suppose where water would end up. We do this by repeatedly asking a unit of water to flow down terrain, and tally the cells it passes through:

```
def make_flow_acc(grid, limit = 10, iterations = 10000):

    let N be the number of rows/columns in the grid
    Initialize flow accumulation matrix Fa = (N,N)

    for i in range(iterations):
        let grid[i,j] be a randomly selected current_cell in grid

        visit the cell in Fa matrix Fa[i,j] += 1
        initialize visited_cells to 1

        while visited_cells < limit:
            get the direction vector of the cell c[i,j, 2]
            unpack the direction vector to a list of directions
            select a random direction from list (dx, dy)

            current_cell = grid[i + dx, j + dy] += 1
            visit the cell in Fa matrix Fa[i + dx, j + dy] += 1

    return Fa
```

Recall that a direction code [0-494] stores up to 8 possible downstream directions. In `make_flow_acc`, the next visit is chosen at random, making this simulation stochastic.

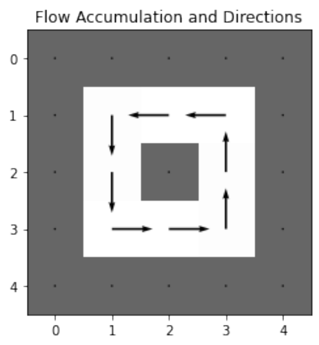

Figure 7: Penrose Accumulation (Infinite Staircase).

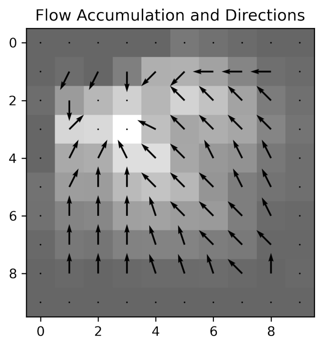

Figure 8: Basin Accumulation. Arrows point to the direction of steepest descent, a vector sum of all downstream directions.

While the visualization of the flow accumulation matrix is seemingly equivalent to the height map in simple examples, it is much more useful for complex terrains. Additionally, it can be used to simulate local patterns of rainfall. For example, we simulate persistent rainfall at the borders of the Beauford Watershed first before applying uniform rainfall,

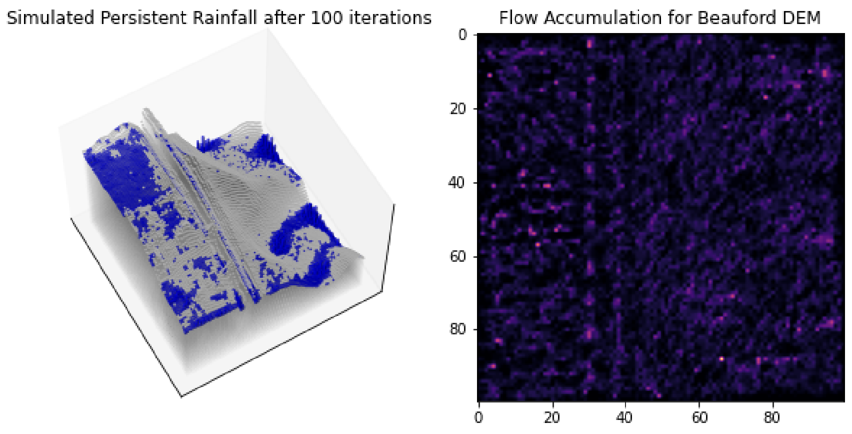

Figure 9: CA vs Flow Accumulation for Beauford Watershed.

Notice how both the CA and the flow accumulation matrix trace the straight ridges and suggest similar regions of high flooding risk, but deviate significantly in segments with more complex geometry. The CA, for example, suggests that flooding will occur in the

Northwest quadrant, with flooding occurring the Southeast quadrant.

2.4 Metrics

To ground the discussion of metrics, we will refer to two toy models introduced earlier: the Basin Model and the Dam-Basin Model.

1. Total Mass (m^3).

   This metric primarily checks that the total mass is conserved across iterations of a simulation with no rainfall. When we simulate with rainfall, this metric proxies for a cumulative precipitation record. The units are a misnomer, as we technically need to rescale this by density (1000 kg/m^3) to get the correct units (in kg).

2. Average Flow Rate (m^3/s).

   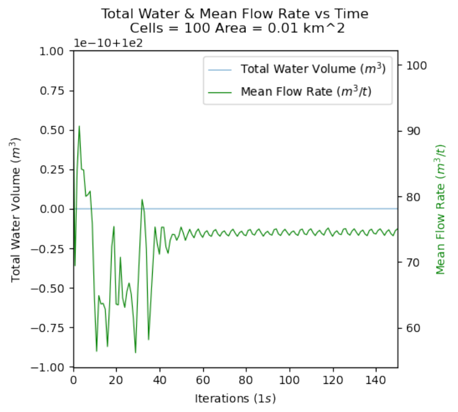

   Figure 10: Total Mass and Average Flow Rate in Basin.

   This metric averages the intercellular volumes across each cell each iteration of the simulation. It answers the question: "How fast is the water flowing at the current iteration of the simulation?". Realizing this metric is the main benefit of adding physical terms to the model: along with inundation risk, we can also estimate the risk of flash flooding.

   A shortcoming of the CA approach is that this metric is prone to oscillations, as small amounts of water pass back and forth between neighbors in the stable regime. To limit these oscillations, we have set minimum and maximum volumes (`v_min` and `v_max`) to `1e-4` and `1e4` respectively.

3. Fraction of Cells Flooded [0,1].

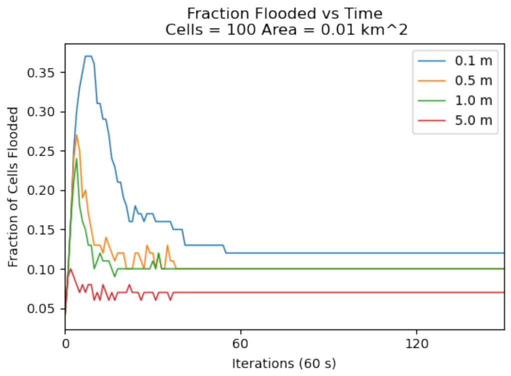

Figure 11: Fraction of cells flooded in Basin.

This metric measures the fraction of cells that are flooded according to a set of thresholds (by default, this is `[0.1m, 0.5m, 1.0m]`). This allows us to estimate the risk of flooding at different levels of severity.

4. Central Cell(s).

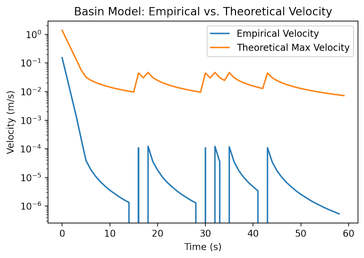

Figure 12: Fraction of cells flooded in Basin.

Records the total height of the water column at a cell/cells of interest across each iteration in the simulation. By default, this is the cell with position [5,5]. In simulating with the study catchment, we will choose areas of importance (points with arterial highways, high-density housing etc.). By calculating the difference between successive points, this can also be used to check that water is flowing below a theoretical maximum of $\sqrt{2gh}$.

2.5 Rainfall Model

To simulate rainfall, we implement a 'Poisson-Cascade Model'. Using an average monthly rainfall (in mm) as input, we simulate 30 days of rainfall, then 12 hours from a sampled day, and then 60 minutes from a sample hour. Each of these is Poisson-distributed as it returns discrete rainfall values and we only need a single mean parameter to define it.

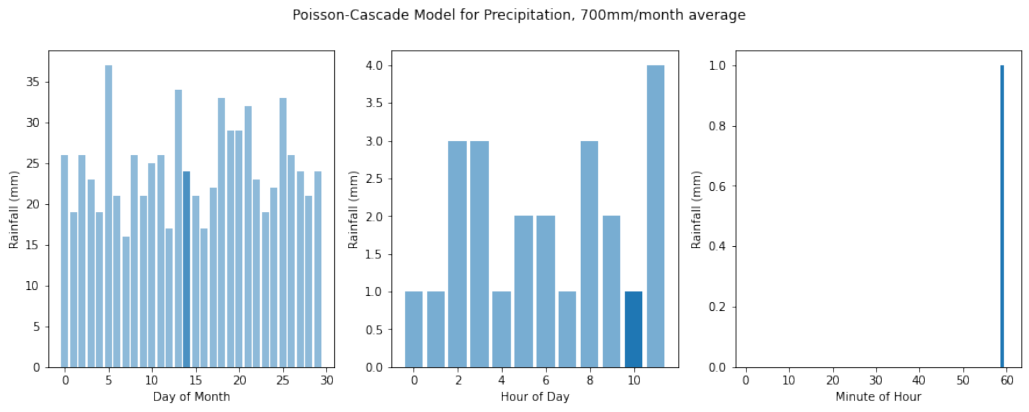

Figure 13: Poisson-Cascade Rainfall Model, for

This model is stochastic, and yields moderately different results for simulations initialized with the same parameters. This model also assumes even rainfall over the Study Area, as opposed to one that models rain clouds moving over time and affecting small locales.

2.6 Implementation

The WeightedCA model was implemented in Python primarily using numpy. Optimizations were won using the `numba` package's JIT compiler for especially expensive routines. Simulation performance statistics were saved in the `pref_resuls.txt` file. A suite of tests were implemented in `Tests.ipynb`, greatly benefiting from plotting utilities in `plotting_utils.py` and functional programming, which allowed tests to be performed on small, intuitively predictable cases. Animated test cases are stored in the `media` folder.

As a performance thumb rule, to simulate a `100x100` grid with `60*12 = 720` iterations and a `60`s timestep (a full rainy day), the model takes an average of about `140s` to run. While this is reasonably efficient, it is sub-optimal for large grids and necessitates grid re-sampling, demonstrated in an earlier section.

3 Case Study: Kerala, India

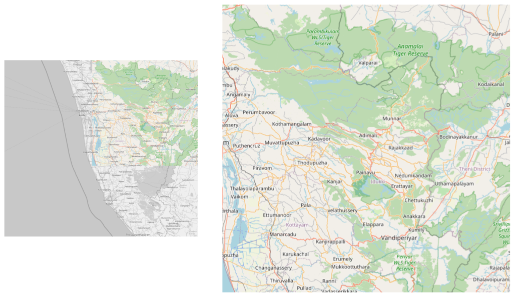

Figure 14: Left: SRTM DEM data. Right: Study Area in Kerala, India

The Study Area (color, above) is a square quadrant of a DEM loaded from the SRTM database with a 3-arc second resolution (`90m x 90m` per cell). This quadrant measures `72 km x 72km`, cropped from the overall DEM size of `108 km x 108km` to focus on inland areas more prone to flooding.

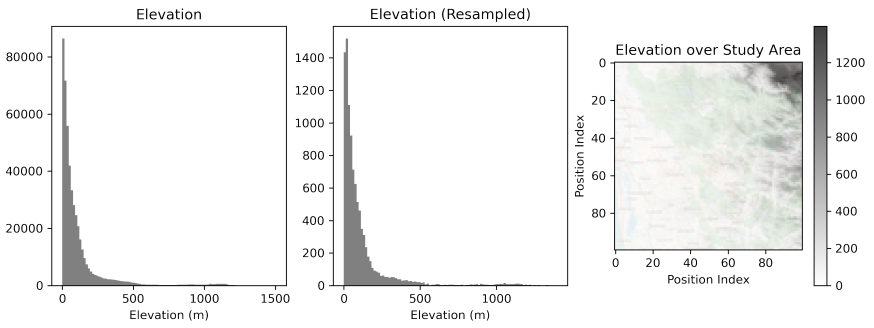

Figure 15: DEM Resampling for Study Area

After resampling, each cell has a size of `720m x 720m`. These large cells introduce a new tradeoff: as we simulate larger areas, we lose out on possible targeted interventions. The Hoover Dam, for instance, stretches only 380m or about half the length of a cell. As such, a proposed dam just 1 cell wide would be a very large dam in real life.

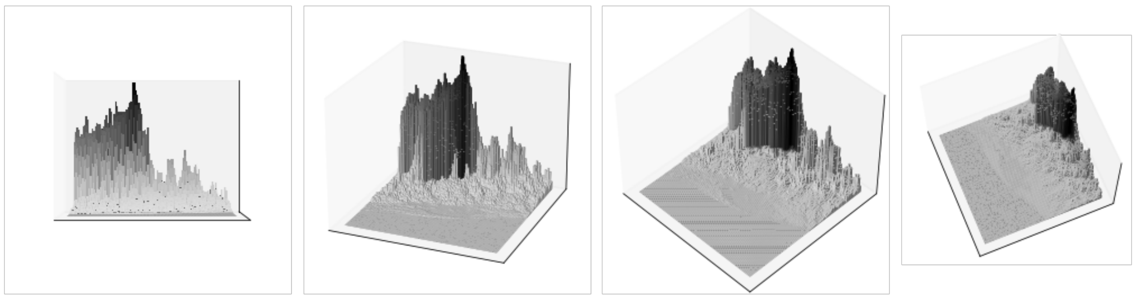

Figure 16: Topography of Study Area

The topography describes mountain ranges in the Northeast that rapidly dissipate to a floodplain almost entirely at sea level. In combination with strong precipitation during the monsoon, these features have created an abundance of still freshwater that has been a boon for rice paddy cultivation, but also made the state prone to flooding in unpredictable ways.

3.1 Simulation Results

We simulate a day of heavy rain (`avg = 50mm`) over 12 hours, with a time step of 60 seconds. Rainfall is applied homogeneously over all cells in the grid. To estimate uncertainty over simulation results, we repeat the experiment 5 times.

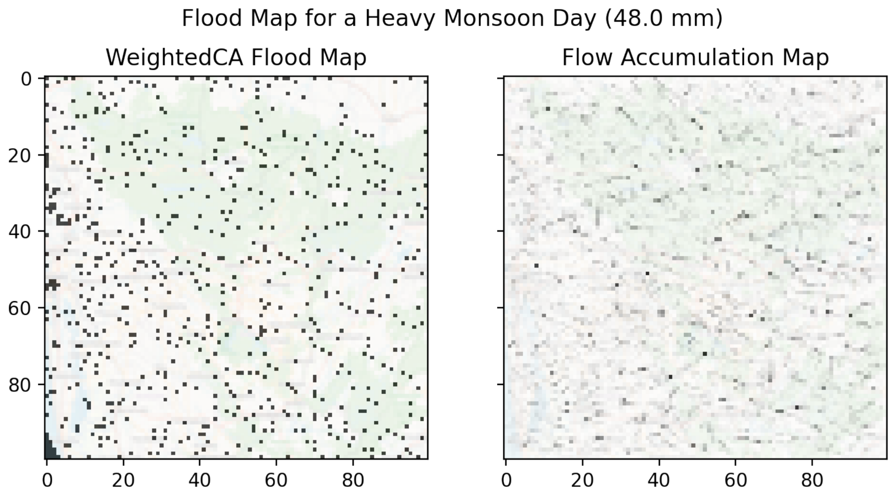

Figure 17: Flood Maps for Heavy Rain

In the figure above, the darkness of the dots corresponds to the extend of the flooding in the cell. Cells without dots generally contain water, albeit in very minute quantities.

Comparing the plots, there is correspondence between the WeightedCA and Flow Accumulation matrix: over a very flat floodplain, ponds form over individual cells or small clusters of cells, but large scale flooding patterns do not seem to occur. This might be because the simulation time scale was too short to allow water to reach stable locations,

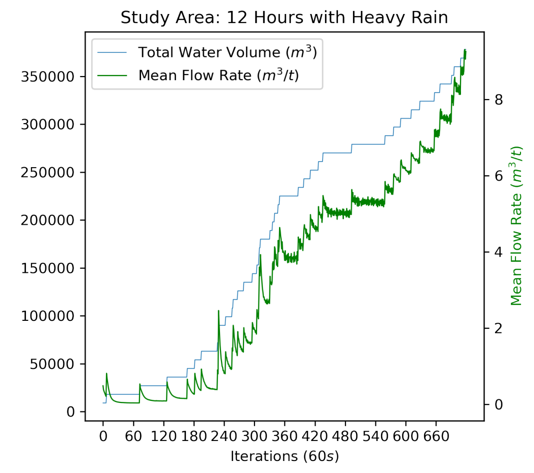

Figure 18

In a sample run, for example, we can see that the mean flow rate continues to rise. In a stable regime where water has settled on average, we would see a much lower mean flow rate oscillating about a long-term mean (Figure 8).

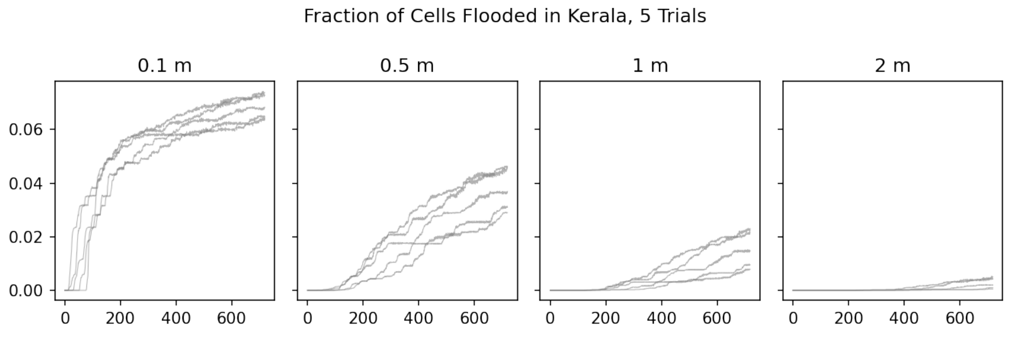

Figure 19: Fraction of Cells Flooded, 5 Runs

According to the figure above, at the end of the simulated day approximately 6% of cells reach 0.1m of water, which can be thought of as a mild or negligible flood. About 2% to 4% experience moderate flooding at 0.5m, while about 1% of cells experience significant flooding of 1m or more.

In each threshold, the fraction of flooded cells increases monotonically (as there is persistent rainfall over the course of the day), albeit at a reducing rate (as outlet or drainage cells collect water). This suggests that there are outlet cells that have higher flooding risks because of the local topography.

3.2 Dam Proposals

While the current study area is quite large, we demonstrate how two strategies for flood mitigation could be implemented. The area these dams will try to protect is the city of Kottayam, marked with a black rectangle below.

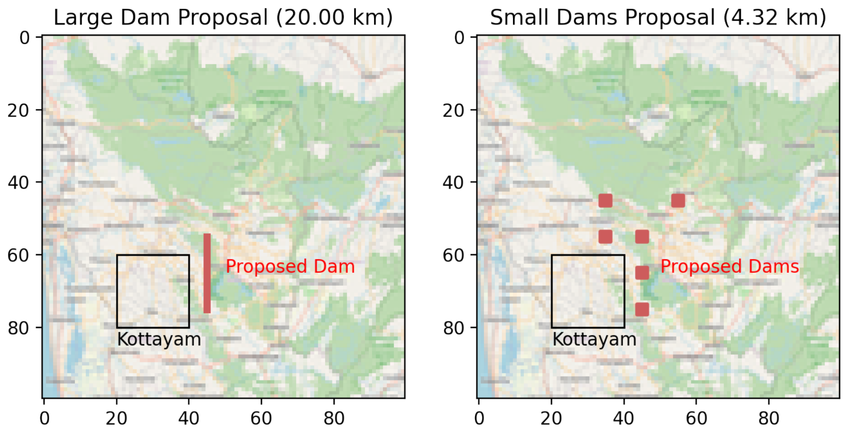

Figure 20: 2 Example Strategies for Flood Mitigation: Large Dam and Small Dams

In the first strategy, an unrealistically large dam embanks the right side of the city. In the second, 6 dams, each occupying one cell, are placed spaced apart towards the North-East of the city. For each strategy, the simulation was repeated for 50 trials (a runtime of about 3 hours in total) with average rainfall of about 50mm as before.

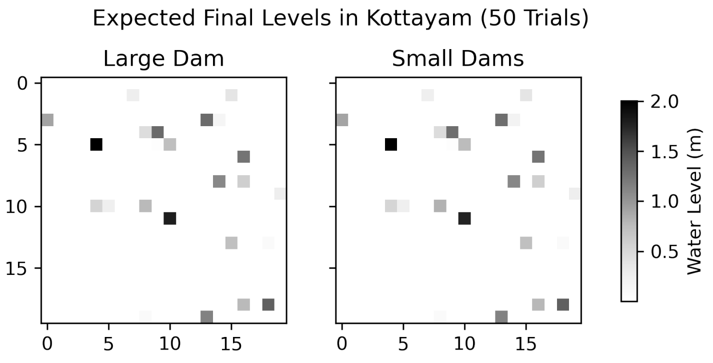

Figure 21: Comparison of Strategies: Expected Flooded Cells

Averaged over all runs, there is seemingly no difference between the strategies. In both cases, the same cells are susceptible to flooding. In the following plot, the minor differences between the strategies can be analyzed in another way,

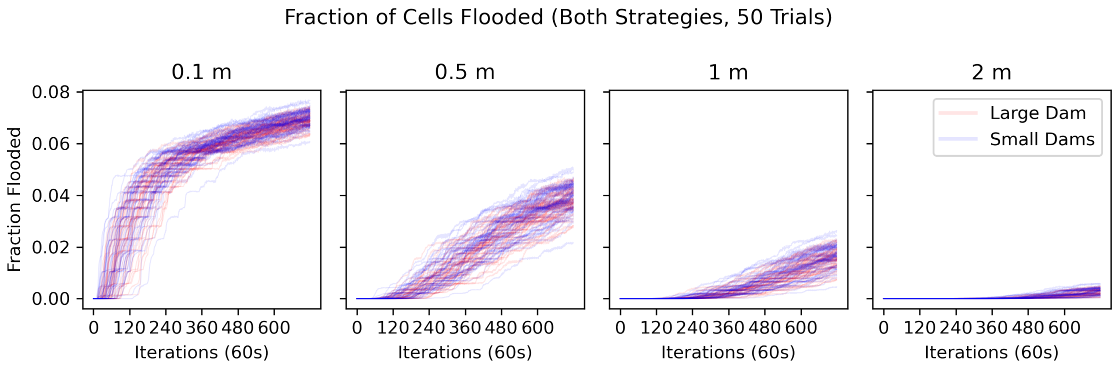

Figure 22: Comparison of Strategies: Fraction of Flooded Cells.

Over 12 hours, the fraction of cells flooded across each threshold exhibit the same dynamics. Interpreted another way, the confidence bands of both strategies envelop each other. If we consider the final iteration of all simulations, we can see if there is in fact a difference between these strategies,

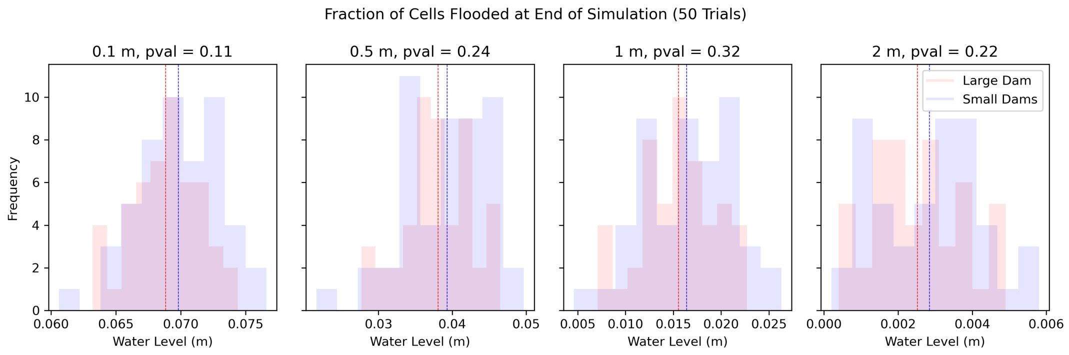

Figure 23: Comparison of Strategies: Fraction Flooded at End of Day

On average, the small dam strategy narrowly reduces flood risk when compared to large dams. However, the difference between these is statistically insignificant, with no p-value falling below 0.11. These flooding fractions are also comparable to the fractions seen without any interventions (Figure 18), and are not very effective solutions. As made apparent by the histograms, the uncertainty over these expected values are nearly identical between strategies, perhaps as an artefact of i.i.d. sampling from a Poisson distribution.

4 Conclusion

Over very flat terrain and with a rainfall model that distributes rainfall evenly over the catchment, the WeightedCA provides detailed information about which cells are at risk of flooding, and the risk of flooding at the simulation scale. However, in simulating over a large study area and cell size, we lack the precision to describe possible intervention strategies. In a following study, we might simulate smaller catchments with smaller cell sizes and realize better results. Additionally, an improved rainfall model that includes moving clouds (and subsequent local rainfall patterns) might improve the model.

5 References

1. Michele Guidolin et al. (2016). A cellular automata 2D inundation model for rapid flood analysis.

2. Coppola, E., et al. (2007). Cellular automata algorithms for drainage network extraction and rainfall data assimilation.

3. Cirbus, J., Podhoranyi, M. (2013). Cellular Automata for the Flow Simulations on the Earth Surface, Optimization Computation Process

   The paper suggests that it uses the D8 model for flow direction (pick the Direction of the 8 neighbors with the lowest elevation). However, when I replicated Figure 3 in the paper, I found that it was instead doing a "pick the sum of the directions of all lowest neighbors" technique, which I haven't seen in similar papers.

   There was a miscalculation in the slope field. I corrected this and tested my calculations against a DEM-manipulation package, and got exactly the same results.

   Nonetheless, I have so much concern about the update rules and how they are defined. Each iteration, and each cell, the water in the cell changes by (water in from neighbors [to whom the central cell is the D8 neighbor]) - (water out to D8 neighbor). There is an idea about transfers happening in 'active cells' in the control flow diagram, but there is no mention of what makes a cell active in the rest of the paper.

4. Barnes, Richard. 2016. RichDEM: Terrain Analysis Software