Lattice Evolution

Jan Kierzkowski

July 2024

1 Introduction

1.1 Rock erosion described by a lattice

A rock with eroding channels can be described by a graph in which each edge represents a channel, and the nodes represent the endpoints of those channels. In this simulation, a triangular lattice is used as the starting point. All edges are randomly widened, and if any two neighboring edges satisfy a specific condition, they are merged as shown in the figure below:

$$d_i + d_j \geq L_k \tag{1}$$

where d_i and d_j are the diameters of neighboring edges, and L_k is the length of the edge connecting their endpoints.

The merged edges are represented by two elements: one normal edge with diameter $d = d_i + d_j$, and one edge with infinite conductivity (non-resistance), which effectively represents the merging of two nodes into one. This solution helps avoid the complications of managing parallel edges, as illustrated in the figure below.

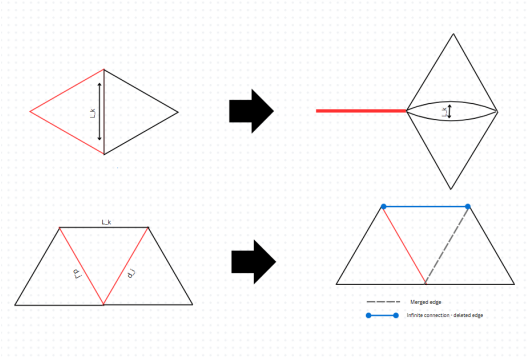

Figure 1: The mechanism of merging edges.

The main challenge with parallel edges is node management—removing a node and reconnecting all attached edges is complex. Additionally, track-

ing distances between such merged nodes would require unnecessary data structures.

1.2 Flow equations

The flow through each edge connecting the i-th and j-th nodes is described by:

$$q_{ij} = \frac{K_{ij}}{l_{ij}}(p_i - p_j) \tag{2}$$

where conductivity $K_{ij} = \frac{d^4}{128 \cdot \mu}$ (with d being the edge diameter and μ the constant viscosity for all edges), l_{ij} is the edge length, and p_i, p_j are the node pressures.

The boundary conditions are such that pressures at nodes in the first row are set to $p_{\text{in}} = 1$, and those in the last row to $p_{\text{out}} = 0$. Additionally, a continuity equation is applied—net flow at every node must be zero.

2 The Algorithm

The simulation is divided into two stages:

1. Lattice graph computation,
2. Flow computation.

The first part is implemented in C++ for higher performance compared to Python. The triangular lattice graph is initially generated using the NetworkX Python library. Node positions are then passed into the C++ code for further simulation. A single time step consists of the following:

- An edge is randomly selected and its diameter is increased,
- Neighboring edges are checked for the merging condition (Equation 1),
- If the condition is met, edges are merged.

During merging, three edges are involved: - The enlarged edge, - A neighboring edge, - The connecting edge between them (used to evaluate L_k).

This third edge becomes the non-resistant edge; the second edge is removed from the graph, and the diameter of the first is increased by the diameter of the removed one.

The simulation can be configured by changing the enlargement factor, the lattice size (20×20 in this case), and the number of time steps.

The second stage, flow calculation, is implemented in Python. The flow equations are solved using QR decomposition (NumPy's 'np.linalg.qr'). The pressure vector is calculated as [1]:

$$\mathbf{p} = R^{-1}\left(Q^{\mathsf{T}}\mathbf{b}\right), \tag{3}$$

where $\mathbf{p}$ is the pressure vector, Q and R are the matrices from QR decomposition, and $\mathbf{b}$ is the right-hand side of the system of equations. Flows are then calculated based on these pressures.

3 Results

3.1 First stage – lattice evolution

The main difficulty in this stage was the performance of Python scripts, due to frequent condition checking during each time step. This problem was solved by offloading the core computation to C++. The results were passed to the second stage for further analysis.

3.2 Second stage – flow computation

Initially, with all edges having equal diameters, pressure is evenly distributed across all nodes.

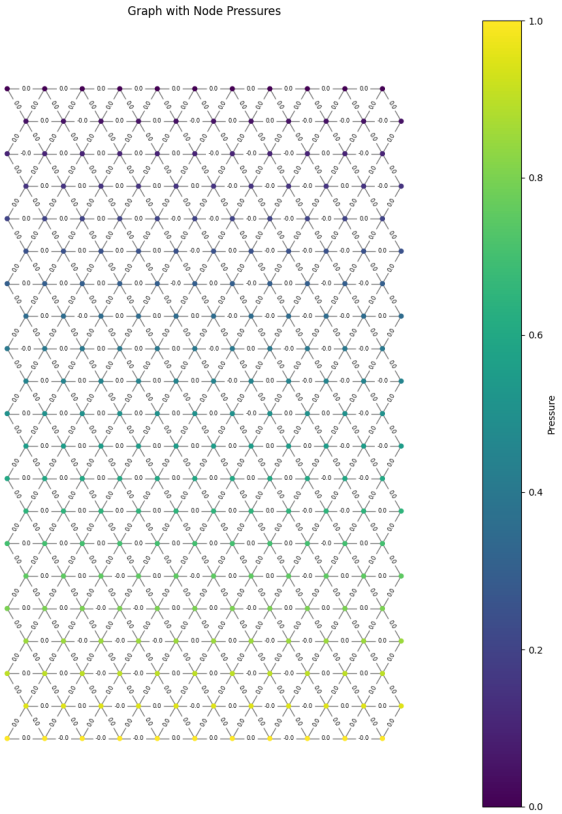

Figure 2: Pressures and flow at the beginning of the simulation.

The next image shows the lattice after 73 modifications. Input nodes (with pressure = 1) are connected via non-resistant edges to several other nodes, making them part of the input. Similarly, more output nodes appear, shortening the flow path. This mechanism continues (Figure 4) until a closed circuit of non-resistant edges forms (Figure 5), at which point the system of equations becomes unsolvable. In Figure 5, one input node no longer has a pressure of 1. To obtain a solution in such a scenario, only the most independent equations are retained, and others are removed from the system.

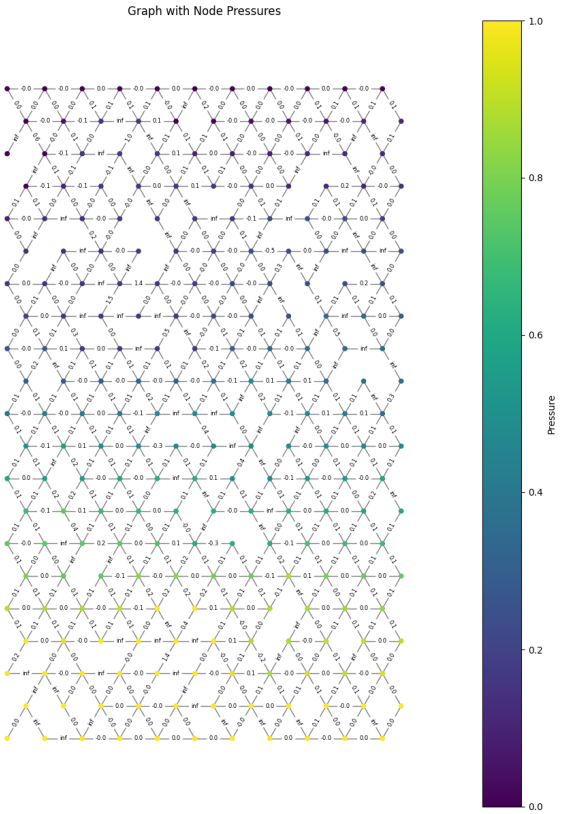

Figure 3: Pressures and flow after 73 changes to the lattice.

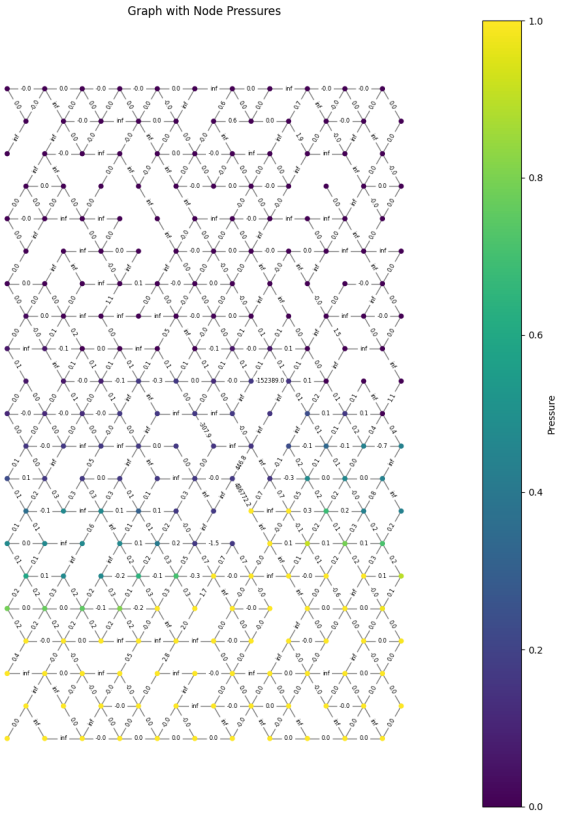

Figure 4: Pressures and flow after 109 changes to the lattice.

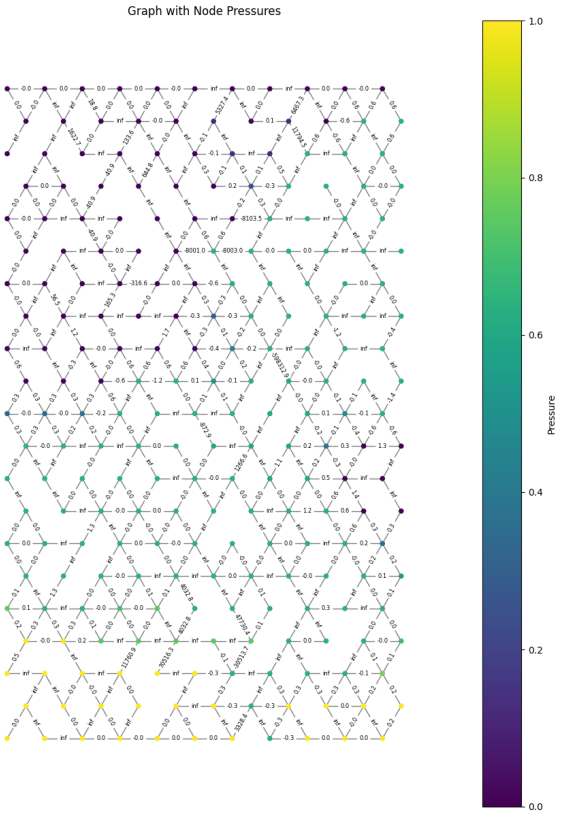

Figure 5: Pressures and flow after 146 changes to the lattice.

4 Summary

In this paper, I presented a method and simulation results addressing the problem of merging edges in a graph representing a rock with eroding and connecting channels. The dual-stage simulation enabled efficient lattice evolution and accurate flow calculation. Key challenges such as node management during merging and maintaining solvability of flow equations were addressed with practical algorithmic solutions.

References

[1] *Using QR decomposition for solution to linear inverse problems*. URL: https://en.wikipedia.org/wiki/QR_decomposition#Using_for_solution_to_linear_inverse_problems.