Computational Analysis of Percolation Thresholds using Union-Find Algorithms and Monte Carlo Simulation

Vivekjit Das
Department of Computer Science and Engineering
IIT Madras

December 18, 2025

Abstract

This report details the design and implementation of a high-performance percolation simulator. The project models the flow of fluid through a porous medium using an $N \times N$ grid. By leveraging the Weighted Quick-Union with Path Compression (Union-Find) data structure, we efficiently solve the dynamic connectivity problem. A Monte Carlo simulation was conducted to determine the critical probability threshold p^*, confirming the phase transition behavior at $p^* \approx 0.592746$. The project concludes with a real-time visual simulation developed in C++ using the SFML library.

Contents

1 Introduction

Percolation theory is a branch of statistical physics and mathematics that studies the movement of fluids through porous materials. A fundamental question in this field is: *"If sites in a lattice are independently open with probability p, what is the threshold p^* at which a giant connected component spans the lattice from top to bottom?"*

This project addresses this problem computationally. Rather than analytical derivations, which are often intractable for finite grids, we employ an algorithmic approach. The system is modeled as a dynamic graph where nodes represent grid sites and edges represent open connections. The core challenge is to efficiently track connectivity as the grid evolves, requiring advanced data structures to handle millions of operations in real-time.

2 Graph Theory Modeling

2.1 The Lattice as a Graph

We model the $N \times N$ percolation grid as a Graph $G = (V, E)$.

- **Vertices (V):** The set of N^2 sites, indexed by (row, col) where $0 \leq row, col < N$.
- **Edges (E):** An undirected edge exists between two vertices u and v if and only if:
    1. u and v are physical neighbors (Up, Down, Left, Right).
    2. Both u and v are **Open**.

2.2 The Virtual Sites Optimization

A naive check for percolation would require iterating through every site in the top row and running a search algorithm (like BFS or DFS) to see if any site in the bottom row is reachable. This yields a time complexity of $O(N^2)$ per check, or $O(N^4)$ for a full simulation, which is computationally prohibitive.

To optimize this, we introduce two **Virtual Nodes**:

1. **Virtual Top (S):** Connected to all open sites in the first row.
2. **Virtual Bottom (T):** Connected to all open sites in the last row.

The percolation problem is thus reduced to a single connectivity query:

$$\text{Percolates} \iff \text{connected}(S, T) \tag{1}$$

3 Data Structures and Algorithms

The efficiency of this simulation relies entirely on the underlying data structure used to manage disjoint sets of connected sites.

3.1 The Union-Find Data Structure

We utilize the **Disjoint-Set Union (DSU)** data structure. It supports two primary operations:

- `union(p, q)`: Merge the set containing p with the set containing q.
- `find(p)`: Return the canonical identifier (root) of the set containing p.

3.2 Optimizations: Reaching Near-Constant Time

Standard implementations of Union-Find (like Quick-Find or Quick-Union) are too slow ($O(N)$ per operation). We implemented two specific optimizations:

3.2.1 1. Weighted Quick-Union

When merging two trees, we always link the root of the smaller tree to the root of the larger tree. This ensures the height of any tree is bounded by $\log_2 N$.

3.2.2 2. Path Compression

During the `find()` operation, we make every node in the path point directly to its grandparent. This flattens the tree structure dynamically.

```
int UnionFind::find(int p) {
    while (p != parent[p]) {
        parent[p] = parent[parent[p]]; // Path Compression
        p = parent[p];
    }
    return p;
}
```

Listing 1: Path Compression Implementation

3.3 Time Complexity Analysis

With these two optimizations, the amortized time complexity for any combination of M union/find operations on N objects is:

$$O(M \cdot \alpha(N)) \tag{2}$$

Where $\alpha(N)$ is the Inverse Ackermann function. For all practical values of N, $\alpha(N) \leq 5$, making the operations effectively constant time $O(1)$.

4 Monte Carlo Simulation

4.1 Methodology

To estimate the percolation threshold p^*, we employ a Monte Carlo simulation. The analytic calculation for a specific N is intractable, so we rely on the law of large numbers.

Simulation Steps:

1. Initialize an $N \times N$ grid with all sites blocked.
2. Randomly pick a site (i, j).
3. If the site is closed, open it.
4. Perform `union` operations with adjacent open neighbors.
5. Check `percolates()`. If true, record the fraction of open sites $p = \frac{\text{open sites}}{N^2}$.
6. Repeat for T trials and average the results.

4.2 The Phase Transition

The experimental results demonstrate a sharp phase transition.

- For $p < 0.59$, the probability of percolation is near 0.
- For $p > 0.59$, the probability of percolation is near 1.

This confirms the critical threshold for a 2D square lattice is approximately 0.592746.

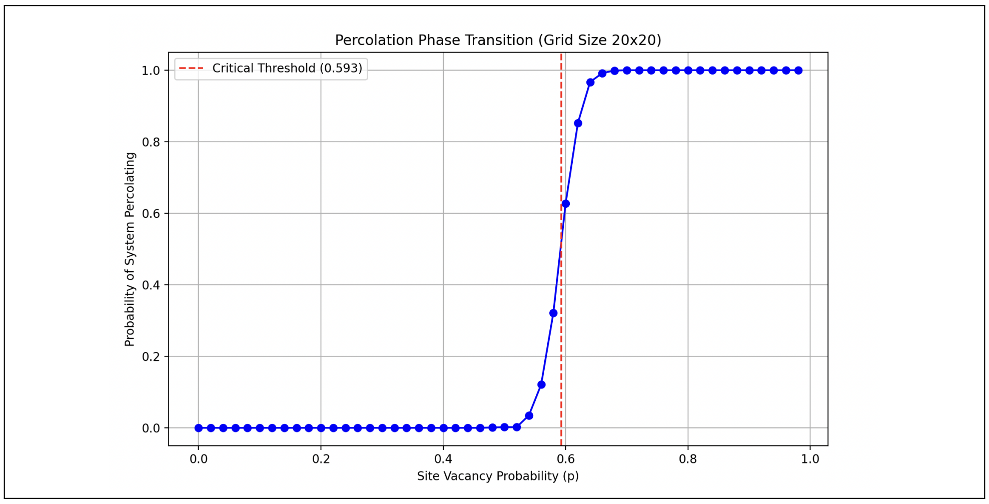


Figure 1: Probability of percolation vs. Site Vacancy Probability (p). The vertical red line marks the critical threshold.

5 Visual Simulation Implementation

The final component of the project is a real-time visualizer built in C++ using the **SFML (Simple and Fast Multimedia Library)**.

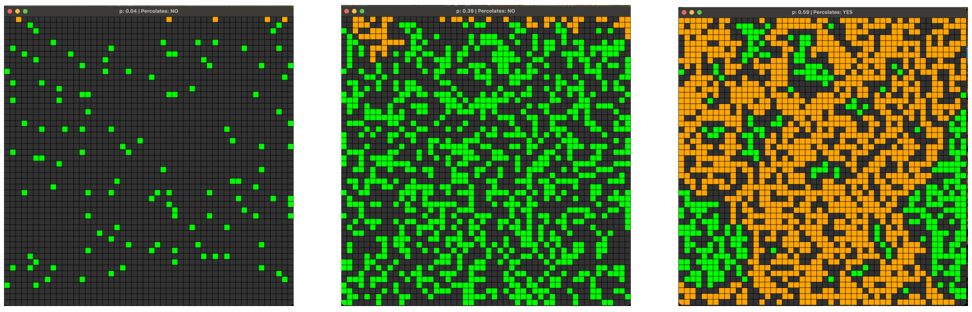
5.1 Architecture

The visualizer runs a continuous loop at 60 FPS:

1. **Input Handling:** Listens for keyboard events (Spacebar to "rain" sites, 'R' to reset).
2. **Physics Update:** Opens random sites and updates the Union-Find structure.
3. **Rendering:** Draws the grid state.
   - **Black:** Blocked Site.
   - **Green(denoting trees):** Open Site (Empty).
   - **Orange(denoting Fire):** Full Site (Connected to Virtual Top).

5.2 Handling SFML 3.0 Updates

The project was developed using the latest SFML 3.0 library, which required adapting standard patterns (e.g., using `sf::VideoMode({w, h})` and `std::optional` for event polling).



(a) $p = 0.1$ (b) $p = 0.59$ (c) $p = 0.8$

Figure 2: Evolution of the grid at different probabilities.

Figure 3: Real-time visualization of the percolation process. Orange cells represent the "Full" component connected to the top.

6 Conclusion

This project successfully implemented a highly optimized system to solve the percolation problem. By utilizing the Union-Find data structure with path compression, we achieved near-linear performance, allowing for large-scale simulations. The Monte Carlo experiments verified the theoretical critical probability of $p^* \approx 0.593$, and the visual simulation provides an intuitive demonstration of the phase transition phenomenon.

Future work could involve implementing the "Backwash" fix using dual Union-Find structures or extending the simulation to 3D lattices.

7 References

1. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.
2. Stauffer, D., & Aharony, A. (2018). *Introduction to Percolation Theory.* Taylor & Francis.
3. SFML Development Team. *Simple and Fast Multimedia Library 3.0 Documentation.*