Report: Wi-Fi Hotspot Interference Minimization

Introduction

This project addresses the problem of minimizing interference among 1,000 Wi-Fi hotspots distributed across a 5x5 km area. Each hotspot operates on one of five channels, and interference occurs when two hotspots within 275 meters of each other use the same channel. The objectives are to generate hotspot locations ensuring a minimum separation of 50 meters, optimize channel assignments to reduce interference, and visualize the results through plots. The solution is implemented in Python, using object-oriented programming, SQLite for data storage, and matplotlib for visualization.

Approach

The program is structured as a single Python script with modular functions, leveraging libraries such as NumPy, SciPy, Matplotlib, SQLite3, and random. Below is an overview of the key components:

- **Hotspot Generation**:
    - A `Hotspot` class encapsulates the x-coordinate, y-coordinate, and channel of each hotspot.
    - The `generateHotspots` function creates 1,000 hotspots with random positions in a 5x5 km area, ensuring no two hotspots are closer than 50 meters. Positions are generated using `random.uniform`, and distances are checked using the Euclidean distance formula.
    - Hotspots are assigned a random channel (1 to 5) using `random.randint`.
- **Data Storage**:
    - Hotspots are saved to an SQLite database (`hotspots.db`) using `saveHotspotsToDb`, which creates a table with columns for id, x, y, and channel.
    - The `loadHotspotsFromDb` function retrieves hotspot data, enabling persistence and reuse.
- **Interference Detection**:
    - The `findInterferingHotspots` function identifies pairs of hotspots that are within 275 meters and share the same channel. It uses SciPy's `distance_matrix` for efficient distance calculations, returning interfering pairs and a set of affected hotspot indices.

- **Channel Optimization**:
    - The `optimizeChannels` function iteratively reduces interference by randomly selecting an interfering hotspot and assigning it a new channel. The process runs for up to 100 iterations or until no interference remains, tracking the number of interfering hotspots per iteration.
- **Visualization**:
    - The `plotInterferenceTrend` function generates a plot showing the number of interfering hotspots over iterations, saved as `interference_trend.png`.
    - The `plotHotspots` function creates a scatter plot of hotspots, with colors indicating channels, red borders for interfering hotspots, and red lines connecting interfering pairs, saved as `hotspot_map.png`.

Results

The program successfully generates 1,000 hotspots, optimizes their channels, and produces visualizations. The interference trend plot (Figure 1) typically shows a rapid decrease in the number of interfering hotspots within the first few iterations, stabilizing as the algorithm converges. The hotspot map (Figure 2) visually confirms the spatial distribution, channel assignments, and remaining interference, with red lines highlighting problematic pairs.

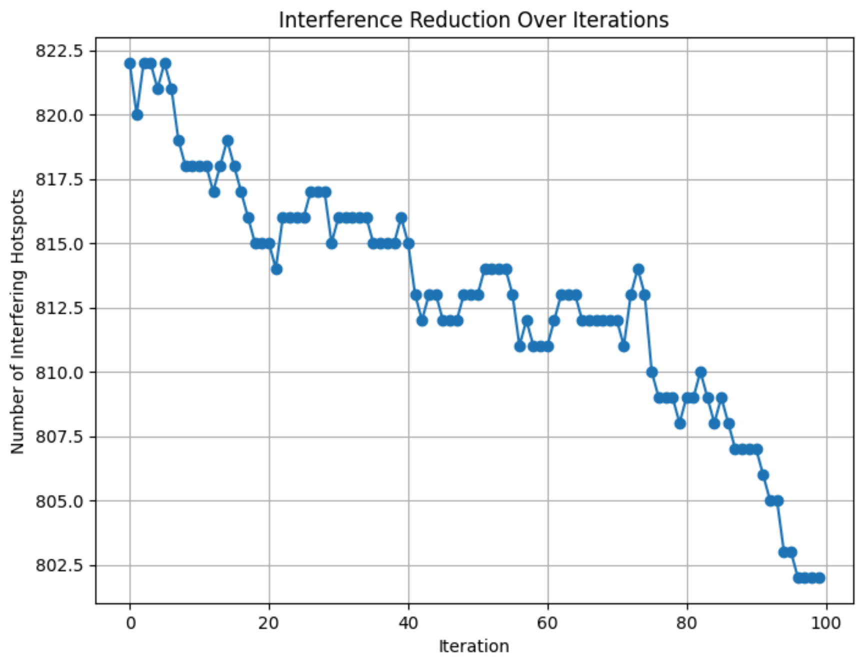

Figure 1: Interference Trend Interference Trend This plot shows the number of interfering hotspots decreasing over iterations, indicating effective channel optimization.

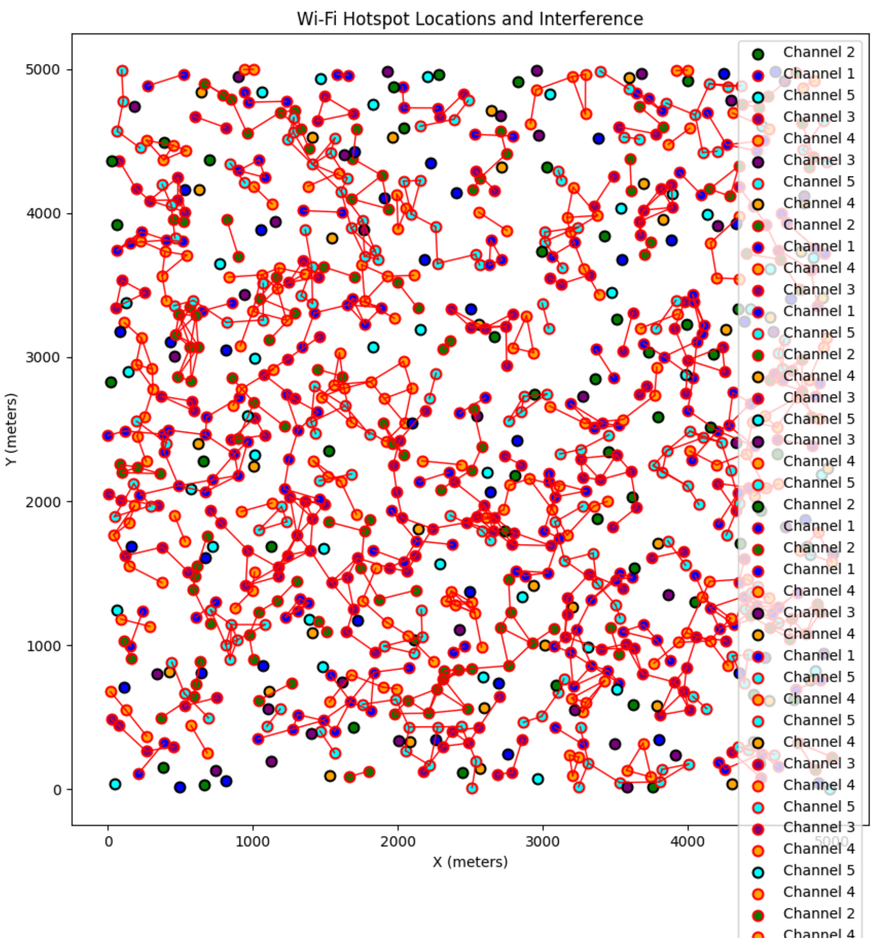

Figure 2: Hotspot Map Hotspot Map This scatter plot displays hotspot locations, with colors representing channels, red borders for interfering hotspots, and red lines connecting interfering pairs.

Discussion

The approach is effective due to its simplicity and modularity. The use of SQLite ensures data persistence, while SciPy's distance matrix optimizes interference detection. The random channel reassignment strategy is computationally lightweight but may not always find the global minimum interference. Future improvements could include:

- Implementing a more sophisticated optimization algorithm, such as simulated annealing or graph coloring.
- Parallelizing distance calculations for larger hotspot counts.
- Adding user-configurable parameters for area size, minimum distance, or iteration limits.

The program is robust, handling edge cases like failed hotspot generation (via a maximum attempt limit) and ensuring all hotspots are saved and loaded correctly. The visualizations provide clear insights into the optimization process and final configuration.

Conclusion

This project demonstrates a practical solution to Wi-Fi hotspot interference minimization, combining random generation, iterative optimization, and clear visualization. The code is modular, well-documented, and extensible, making it suitable for further enhancements or real-world applications.