Rapport: Frequency Assignment for Antenna Interference Mitigation

Awen Desbois, Julien Calisto

November 30, 2023

Contents

Chapter 1

Introduction

In the rapidly evolving landscape of telecommunications, the effective management of radio frequencies is paramount to ensuring seamless communication. One of the challenges faced by telecommunication companies involves strategically assigning frequencies to antennas distributed across a geographical area. The objective is to minimize interference between antennas, as overlapping frequencies can lead to signal degradation and operational inefficiencies.

We present here a specific scenario in which they have a network of antennas spread over a given geographical area. Each antenna uses a separate radio frequency to communicate. The problem arises when antennas are close to each other, which can lead to interference problems. The challenge is compounded by the limited number of frequencies available. Consequently, we are looking for an optimal solution that involves allocating the minimum number of distinct frequencies to the antennas while simultaneously mitigating interference.

The foundation of this assignment lies in devising a frequency assignment strategy that strikes a balance between minimizing interference and efficiently utilizing the limited pool of available frequencies. To address this, a careful analysis of the spatial distribution of antennas and the potential for interference is essential. This report describes a systematic approach to this problem and presents a viable solution.

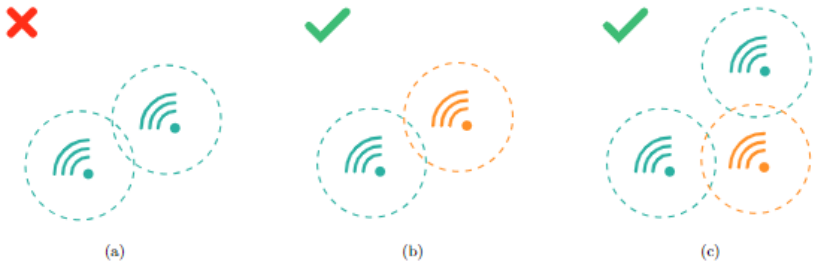

Figure 1.1: Interference between antennas

Chapter 2

Materials

The information is provided by 8 antennas distributed over a geographical area, each specified by its coordinates (x, y) in kilometers:

Antenna 1: (0, 0)
Antenna 2: (3, 5.2)
Antenna 3: (6, 0)
Antenna 4: (9, -5.2)
Antenna 5: (9, 0)
Antenna 6: (9, 5.2)
Antenna 7: (9, 10.4)
Antenna 8: (12, 0)

Antennas are considered to interfere with each other if they are less than 8.7 km apart. The geographical coordinates provide the spatial framework necessary for analyzing potential interference and devising an effective frequency assignment strategy.

Chapter 3

Assignment

3.1 Step 1: Modeling Antenna Frequency Assignment as a Graph Coloring Problem on Unit-Disk Graphs

To address the telecommunication challenge, we model the antenna frequency assignment problem as a graph coloring problem on unit-disk graphs. This approach involves representing antennas as nodes in a graph, where edges connect antennas that are within the interference radius of each other. The goal is to assign colors to nodes (antennas) in such a way that no two connected antennas share the same color.

Our supplied Python code implements this model using the NetworkX library. Here is an overview of the main components:

1. `AntennaGraph` Class:
   - `__init__`: Initializes the AntennaGraph with an empty adjacency list, antenna colors, and a default color palette.
   - `add_connection`: Adds a connection between two antennas to the adjacency list.
   - `add_connections`: Adds multiple connections to the adjacency list.
   - `assign_colors`: Assigns colors to antennas based on a given pattern.
   - `print_graph`: Creates a NetworkX graph and visualizes it using Matplotlib.
   - `create_networkx_graph`: Converts the adjacency list to a NetworkX graph.
   - `draw_graph`: Visualizes the graph using NetworkX and Matplotlib.
2. **Functions for Modeling**:
   - `calculate_adjacency_list`: Computes the adjacency list based on antenna coordinates and interference radius.
   - `calculate_distance`: Computes the Euclidean distance between two antenna coordinates.
   - `model_antenna_frequency`: Main function to model the antenna frequency assignment problem, returning the adjacency list and the number of antennas.
   - `create_antenna_coordinates`: Creates a list of antenna coordinates.
3. **Example Usage**:
   - `antenna_coordinates`: Provides the (x, y) coordinates of the 8 antennas.
   - `interference_radius`: Specifies the radius within which antennas interfere with each other.
   - `adj_list, num_antennas`: Calls the `model_antenna_frequency` function to calculate the adjacency list and the number of antennas.
   - `graph_model`: Creates an instance of the AntennaGraph class, adds connections based on the calculated adjacency list, and visualizes the graph.

This step establishes the foundational framework for addressing the frequency assignment problem. Subsequent steps will involve refining this model to achieve an optimal solution.

3.2 Step 2: Algorithm for Graph Coloring using Repeated Applications of Maximum Independent Set

To address the frequency assignment problem, a graph coloring algorithm is designed based on repeated applications of the Maximum Independent Set (MIS) concept. The goal is to iteratively find colorings for the antennas, ensuring that antennas with shared interference radii do not share the same color.

3.2.1 Key Components of the Algorithm

- **Conversion Functions**:
  - `int_to_binary`: Converts an integer to a binary string with a specified size.
  - `have_common_bit`: Determines if two binary strings have common bits.
- **Distance Calculation**:
  - `calculate_distance`: Computes the Euclidean distance between two antenna coordinates.
- **Cost Function**:
  - `cost_function`: Evaluates the cost (disutility) of a given coloring based on interference and assigned colors.
- **Coloring Functions**:
  - `find_min_cost_coloring`: Identifies the coloring with the minimum cost using a dynamic programming approach.
  - `remove_edges`: Removes edges connected to a specific vertex from the adjacency list.
  - `remove_vertices_and_edges`: Removes vertices and associated edges based on a binary string representing the selected vertices for coloring.
- **Color Counting**:
  - `count_color`: Counts the number of colors used in the coloring process.
- **Graph Coloring Algorithm**:
  - `graph_coloring_algorithm`: Iteratively applies the MIS-based coloring process until all antennas are assigned colors. The algorithm selects colorings with minimal cost to ensure an optimal solution.
- **Example Usage**:
  - `color_list`: Calls the `graph_coloring_algorithm` function, which returns a list of binary strings representing the colors assigned to each antenna.
  - `graph_model.assign_colors`: Assigns colors to antennas in the AntennaGraph instance based on the obtained color list.
  - `graph_model.print_graph`: Visualizes the final colored graph using Matplotlib and NetworkX.

This algorithm leverages the MIS approach to iteratively find optimal colorings for antennas, ensuring minimal interference while efficiently utilizing the available frequencies. The resulting visual representation provides a clear depiction of the frequency assignment strategy for the given set of antennas.

3.3 Step 3: Implementation of MIS Solver in Pulser

Python code implements a Maximum Independent Set (MIS) solver using Pulser, a quantum programming library. This solver is designed to find an optimal configuration of qubits (representing antennas) in a register such that neighboring qubits do not interact. The process involves iteratively updating the set of active qubits based on the results of the MIS Pulser simulations.

3.3.1 Key Components of the MIS Solver

- **Pulse Sequence Setup**:
  - `qubit_register`: Defines a register for the active qubits based on their positions.
  - `coupling_strength`: Calculates the coupling strength between qubits based on the specified separation.
- **Pulse Sequence Configuration**:
  - `pulse_sequence`: Sets up a pulse sequence for the qubit register, utilizing the `rydberg_global` channel on a mock device.
  - `coupling_wf` and `detuning_wf`: Define pulse waveforms for coupling and detuning.
- **Simulation and Analysis**:
  - `simulation_results`: Runs a simulation using the QuTip backend to determine the most probable final state.
  - `probs`: Extracts the probabilities of different final states.
  - `most_probable_state`: Identifies the most probable state based on the simulation results.
  - `binary_representation`: Converts the state to a binary representation.
- **Mismatched Qubits Identification**:
  - `mismatched_qubits`: Identifies qubits that are active in the current MIS solution.
- **Updating Active Qubits**:
  - `update_active_qubits`: Updates the set of active qubits by excluding those identified in the `mismatched_qubits` list.
- **Example Usage**:
  - `solution_1`: Calls the `MIS_pulser` function for the initial set of qubits and prints the active qubits.
  - `solution_2`: Calls the function again with the updated set of active qubits from the previous solution.

Notes:

- The `QutipBackend` class is assumed to be part of an external module (`pulser_simulation`).
- The `MockDevice.rabi_from_blockade` method is used to calculate the coupling strength based on the specified qubit separation.

This implementation demonstrates how Pulser can be utilized to find a Maximum Independent Set for the given qubit positions, ensuring that qubits with shared interference radii are not active simultaneously. The iterative updating of active qubits allows for the refinement of the MIS solution.

3.4 Step 4: Implementation of Graph Coloring Algorithm in Pulser

The provided Python code integrates the graph coloring algorithm developed in Step 2 with the Maximum Independent Set (MIS) solver from Step 3, adapting it for use in Pulser. The quantum graph coloring algorithm assigns colors to antennas while minimizing interference using the Pulser library.

3.4.1 Key Components of the Quantum Coloring Algorithm

- **Function for Quantum Coloring**:
  - `quantum_coloring`: Iteratively applies the MIS solver to find color assignments for antennas, updating the set of active qubits after each iteration.
- **Example Usage**:
  - `antenna_coordinates`: Utilizes the coordinates of antennas from Step 1.
  - `interference_radius`: Specifies the interference radius for determining adjacency.
  - `adj_list, num_antennas`: Calls the `model_antenna_frequency` function to calculate the adjacency list and the number of antennas.
  - `graph_pulser`: Creates an instance of the AntennaGraph class and visualizes the graph.
  - `color_list`: Calls the `quantum_coloring` function to obtain color assignments for antennas.
  - `graph_pulser.assign_colors`: Assigns colors to antennas based on the obtained color list.
  - `graph_pulser.print_graph`:Visualizes the final colored graph using Matplotlib and NetworkX.

This implementation showcases the application of quantum algorithms, specifically using Pulser, to address the graph coloring problem for antenna frequency assignment. By leveraging the MIS solver, the algorithm ensures that neighboring antennas do not share the same color, contributing to the effective management of radio frequencies.

As quantum computing capabilities progress, such algorithms may provide advantages in solving combinatorial optimization problems with real-world applications.