**Discussing Frequency Allocation**

Table of Contents

[**Original Frequency Allocation Script Evaluation by AI** 2](#_Toc195986472)

[**A.** **Frequency Reuse Distance** 2](#_Toc195986473)

[**B.** **Greedy Graph Coloring** 2](#_Toc195986474)

[**C.** **Putting it all together** 4](#_Toc195986475)

[**Why this matters?** 4](#_Toc195986476)

[**PyQGIS Integration** 4](#_Toc195986477)

[**One-Sentence Description** 4](#_Toc195986478)

[**Frequency Allocation Re-coding for Testing (1) by AI** 5](#_Toc195986479)

[**Re-coded Frequency Allocation** 5](#_Toc195986480)

# **Original Frequency Allocation Script Evaluation by AI**

The core of this script is a **frequency-planning engine** built on two pillars:

1. Frequency-Reuse Distance
2. Greedy Graph Coloring

Below, I’ll walk through how each is implemented, and then show how they’re married together to produce an assignment of “channels” (frequencies) to each cell site.

## **Frequency Reuse Distance**

In cellular systems, you want to ensure that two cells using the same frequency are far enough apart that their signal don’t interfere. Here:

* Each Node has a coverage radius (3 km for 3G. 1.5 km for 4G )
* We treat two sites as “potential interferers” if their coverage circles overlap.
* Overlap test in **build\_graph()**

|  |
| --- |
| Python/PyQGIS Code |
| r1 = node1.coverage\_radius / 2  r2 = node2.coverage\_radius / 2  if distance(node1, node2) < (r1 + r2):  graph[node1].append(node2)  graph[node2].append(node1) |

Here, distance(…) is the geodetic separation (in meters) computed via PyQGIS’s QgsDistanceArea

**Interpretation**: If the distance between two towers is less than the sum of their half-radii, their coverage “bubbles” intersect. Those two towers **must not share** same frequency (they become adjacent in the interference graph).

## **Greedy Graph Coloring**

Once we have an **interference graph** (nodes = cell sites, edges = “overlap” relationships), we want to color it (assign one of a small pallete of frequencies) so that no adjacent nodes share a color.

The standard greedy approach is:

1. Iterate over each line (in some order).
2. Look at the colors already used by its neighbors.
3. Choose the “lowest-indexed” color that isn’t in that neighbor set.
4. Assign it.

In this code, however, we go beyond “lowest-indexed” and use distance as a tie breaker:

|  |
| --- |
| **Python/PyQGIS** |
| for node in manager.nodes:  used = {neighbor.frequency for neighbor in graph[node] if neighbor.frequency}    #Build list of all frequencies not used by neighbors  available = [f for f in frequencies[node.node\_type] if f not in used]  #For every candidate f, gather all other nodes already using f:  cochannel\_nodes[f] = [n for n in manager.nodes if n.frequency == f]  #Find, for this node, the \*\*closest\*\* co-channel neighbor for each available frequencies  Lowest\_nodes = find\_lowest\_distance\_nodes(node, cochannel\_nodes)  # → yields { f: (closest\_node, distance\_to\_it) }    #Pick the f whose \*minimum\* distance is the \*largest\*:  chosen = argmax\_f(lowest\_nodes[f].distance)  node.frequency = chosen |

**Why?** Even if two nodes aren’t direct “neighbors” in the overlap graph, you still want them as far apart as possible if they end up sharing a channel. This heuristic picks the frequency whose *nearest* co‑channel tower is *farthest* away, maximizing your reuse distance margin.

## **Putting it all together**

1. **Build the interference graph** using actual geodetic distance (PyQGIS) and coverage radii.
2. **Greedy-color** that graph, but rather than just picking any available color, **score each candidate** by how close the nearest existing co‑channel tower is.
3. **Maximize** that minimum distance → ensures not just “no direct overlap,” but also that same‑channel cells maintain as large a guard distance as possible.

## **Why this matters?**

* **Frequency reuse** is what gives cellular systems their enormous capacity: the same spectrum can be used over and over across the map, provided reuse distance constrainsts are met.
* A purely graph‑theoretic coloring only cares about immediate neighbors; by injecting real distances into the tie‑break, you push co‑channel cells even farther apart, reducing co‑channel interference and boosting SNR at the cell edge.

## **PyQGIS Integration**

 **QgsDistanceArea:** computes real‐world distances on the globe.

 **QgsPointXY** & **QgsProject**: make sure all points and layers sit in EPSG:4326 (WGS84) so distances are meaningful.

 **Map canvas items** (Node, Edge) are kept in sync with their geographic positions, so users can drag sites and see frequencies recalculated in real time.

## **One-Sentence Description**

*This script builds an overlap-based interference graph of your cell sites and then runs a distance-aware greedy graph coloring algorithm: when choosing among available frequencies it always picks the one whose nearest co-channel neighbor is farthest away, thus maximizing frequency-reuse distance.*

# **Frequency Allocation Re-coding for Testing (1) by AI**

**Query**: Now, re-code this combination of frequency reuse distance and greedy graph coloring that is tested for 3G and 4G frequencies with different distances from each other. There should be 30 nodes (15 for each)

## **Re-coded Frequency Allocation**

|  |
| --- |
| **Python/PyQGIS Code** |
| import math  import random  # -----------------------------  # Frequency definitions and reuse distances  # -----------------------------  frequencies = {  '3G': [700, 715, 730, 755, 780, 805, 830],  '4G': [1800, 1815, 1850, 1870, 1900, 1930]  # Minimum separation (meters) for co-channel reuse by tech  # This distance is derived from the frequency reuse distance R = D \* sqrt(N),  # where N is the reuse factor based on acceptable SIR (Signal to Interference Ratio).  # Values are approximated for practical urban macrocell planning.  distance\_threshold = {  '3G': 5000.0, # Larger reuse distance due to lower frequency band propagation  '4G': 3000.0 # Shorter reuse distance due to higher frequency band and smaller cell size  }  class Node:  def \_\_init\_\_(self, node\_id, x, y, node\_type):  self.id = node\_id  self.x = x  self.y = y  self.node\_type = node\_type  self.frequency = None  def distance\_to(self, other):  return math.hypot(self.x, other.x, self.y, other.y)  def \_\_repr\_\_(self):  return f"Node({self.id}, {self.node\_type}, x={self.x:.1f}, y={self.y:.1f}, freq={self.frequency})"  **#--------------------------------------------------------------------------------------**  **Build interference graph based only on technology and threshold**  **#--------------------------------------------------------------------------------------**  **def build\_interference\_graph(nodes):**  **graph = {node: [] for node in nodes}**  **for i, node1 in enumerate(nodes):**  **for node2 in nodes[i+1:]:**  **if node1.node\_type == node2.node\_type:**  **graph[node1].append(node2)**  **graph[node2].append(node1)**  **return graph**  **#-------------------------------------------------------------------------------------**  **Greedy coloring + record candidate distances**  **#-------------------------------------------------------------------------------------**  **def greedy\_graph\_coloring(nodes, graph):**  **assignment\_info = {}**  **for node in nodes:**  **used = {nbr.frequency for nbr in graph[node] if nbr.frequency is not None}**  **candidates = [f for f in frequencies[node.node\_type] if f not in used]**  **candidate\_min\_d = {}**  **for freq in candidates:**  **co\_nodes = [n for n in nodes if n.node\_type == node.node\_type and n.frequency == freq]**  **if co\_nodes:**  **min\_d = min(node.distance\_to(n) for n in co\_nodes)**  **else:**  **min\_d = float(‘inf’)**  **# choose frequency maximizing min\_d**  **if candidate\_min\_d:**  **best\_freq = max(candidate\_min\_d, key=lambda f: candidate\_min\_d[f])**  **node.frequency = best\_freq**  **else:**  **node.frequency = None**  **best\_freq = None**  **assignment\_info[node] = {**  **'candidates': candidate\_min\_d,**  **'assigned': best\_freq**  **}**  **return assignment\_info**  **# -----------------------------**  **# Simulation with explanations**  **# -----------------------------**  **if \_\_name\_\_ == '\_\_main\_\_':**  **random.seed(42)**  **nodes = []**  **# Create 15 3G and 15 4G nodes randomly in 20km×20km area**  **for i in range(15):**  **x, y = random.uniform(0, 20000), random.uniform(0, 20000)**  **nodes.append(Node(f'3G-{i+1}', x, y, '3G'))**  **for i in range(15):**  **x, y = random.uniform(0, 20000), random.uniform(0, 20000)**  **nodes.append(Node(f'4G-{i+1}', x, y, '4G'))**  **graph = build\_interference\_graph(nodes)**  **assignment\_info = greedy\_graph\_coloring(nodes, graph)**  **# Print detailed results**  **print("Frequency assignment with interference details:\n")**  **for node in nodes:**  **info = assignment\_info[node]**  **print(f"Node {node.id} ({node.node\_type}): Assigned {info['assigned']} MHz")**  **# Distances to all same-technology nodes**  **print("  Distances to same-tech nodes:")**  **for other in nodes:**  **if other is not node and other.node\_type == node.node\_type:**  **print(f"    {other.id}: {node.distance\_to(other):.1f} m")**  **# Distances to co-channel nodes**  **print("  Distances to co-channel nodes:")**  **co\_channel = [n for n in nodes if n is not node and n.frequency == node.frequency]**  **if co\_channel:**  **for other in co\_channel:**  **print(f"    {other.id}: {node.distance\_to(other):.1f} m")**  **else:**  **print("    None")**  **# Candidate frequency distances**  **cands = info['candidates']**  **print("  Candidate freq -> min dist to existing co-channel:")**  **for f, d in cands.items():**  **label = "(chosen)" if f == info['assigned'] else ""**  **dist\_str = f"{d:.1f} m" if d != float('inf') else "∞"**  **print(f"    {f} MHz: {dist\_str} {label}")**  **# Explanation**  **chosen = info['assigned']**  **if chosen:**  **max\_d = cands[chosen]**  **print(f"  Explanation: chose {chosen} MHz because its minimum co-channel distance {max\_d if max\_d!=float('inf') else '∞'} m is the largest among candidates, minimizing interference.\n")**  **else:**  **print("  Explanation: no available frequencies due to neighbor usage.\n")** |

This script does the following:

* **Defines 30 nodes**—15 for 3G and 15 for 4G—randomly placed within a 20 km×20 km area.
* Defined distance\_threshold = {'3G': 5000, '4G': 3000} meters.
* **Builds an interference graph** based cell sites with the same technology.
* **Runs a distance‐aware greedy graph‐coloring** to assign frequencies so that co‑channel cells are as far apart as possible.
* Prints distances to all same‑technology neighbors
* **Lists distances to its co‑channel peers** (i.e. other nodes sharing its assigned frequency)
* **Shows, for each candidate frequency**, the minimum distance to existing co‑channel nodes (using “∞” if none exist)
* **Explains the choice**—highlighting that the selected frequency maximizes that minimum distance, thus yielding the lowest co‑channel interference.