GNEBEHI BAGRE JEAN-PHILIPPE

L3 INFORMATIQUE

2017-2018

ALLOCATION DE FREQUENCE D'UN RESEAU MOBILE

Khadir Ouriachi

Enseignant Chercheur à Université de Pau

I- POSITION DU PROBLEME

L'**allocation des fréquences** est l'ensemble des mécanismes qui permettent de définir comment sont réparties les fréquences entre les différents acteurs.
L'enjeu principal est d'éviter les interférences entre les émetteurs.

Dans le cadre de notre formation à l'Université de Pau, nous nous confrontons à un problème d'allocation de fréquences d'un réseau mobile.

Le problème consiste à minimiser le nombre de fréquences à allouer tout en garantissant leur compatibilité vue la disposition géographique des transmetteurs.

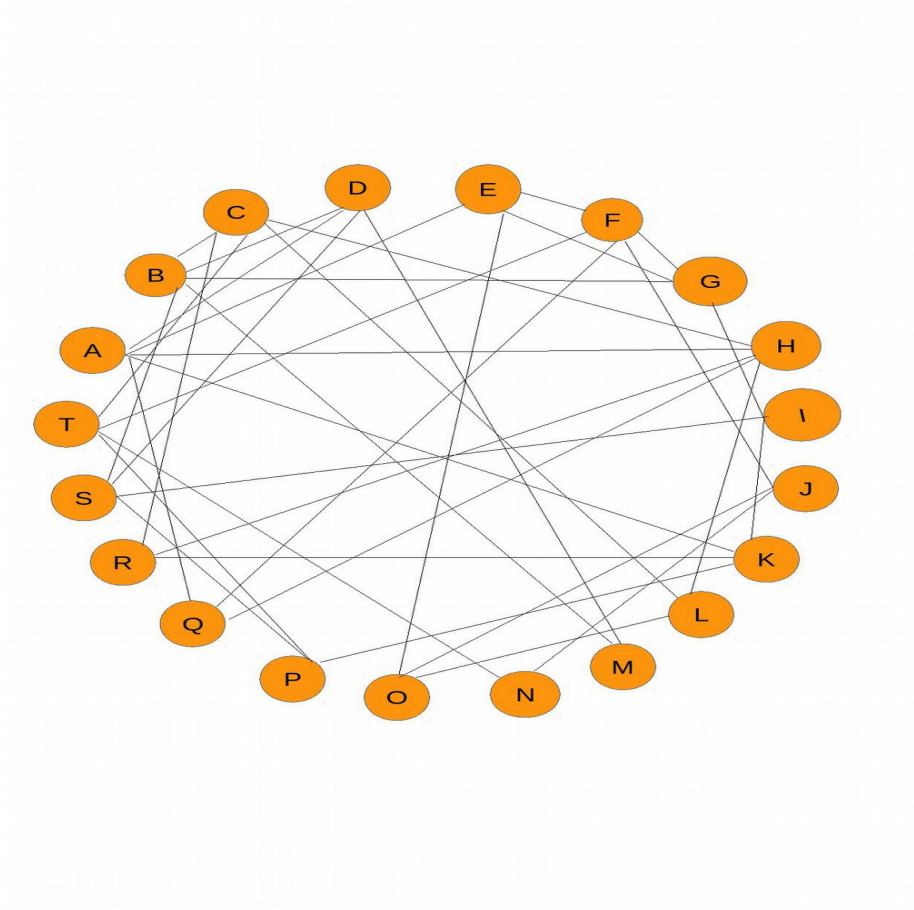
II- REALISATION

II-1 GRAPHE

Pour formaliser le problème nous allons modéliser l'incompatibilité dans l'allocation des fréquences à l'aide d'un graphe non orienté G= (S,A) définit de la manière suivante :

- S représentant l'ensemble des fréquences des différents transmetteurs
- A l'ensemble des relation d'incompatibilité entre les fréquences

Ci-dessous notre graphe d'étude :



II-2 PROBLEME DE COLORATION

La recherche du nombre minimum de fréquences compatibles peut être formulée en termes d'un problème de coloration de graphe.
Cela implique qu'il faut colorer les nœuds représentant les fréquences utilisées.
Les nœuds incompatibles doivent alors avoir des couleurs différentes.
Par conséquent la recherche du nombre minimum de fréquences compatibles se ramène donc au calcul du nombre chromatique du graphe.

II-3 SOLUTION AU PROBLEME

II-3.1 ALGORITHME DE WESH POWELL

L'algorithme de Welsh-Powell est un algorithme qui fournit une solution dite approchée.
Il permet d'obtenir une coloration du graphe en utilisant un nombre k (pas trop grand) de couleurs.
Le résultat retourné n'est cependant pas minimum.
L'algorithme proposé exige les étapes suivantes :

ETAPE 1 :

- Classer les nœuds du graphe dans l'ordre décroissant de leur degré.
- Attribuer à chacun des nœuds un ordre dans une liste triée.

ETAPE 2 :

En parcourant la liste des nœuds dans l'ordre de tri:
- Attribuer une couleur qui n'a pas encore été utilisée au premier nœud non encore coloré
- Attribuer cette même couleur à chaque nœud non encore coloré et non adjacent à un nœud de cette couleur.

ETAPE 3 :

- S'il reste des nœuds non colorés revenir à l'étape 2
- Sinon, la coloration des nœuds est terminée.

II-3.2 IMPLEMENTATION EN C++

```
// A C++ program to implement greedy algorithm for graph coloring
#include <iostream>
#include <list>
using namespace std;

// A class that represents an undirected graph
class Graph
{
        int V; // No. of vertices
        list<int> *adj; // A dynamic array of adjacency lists
public:
```

```
	// Constructor and destructor
	Graph(int V) { this->V = V; adj = new list<int>[V]; }
	~Graph()		{ delete [] adj; }

	// function to add an edge to graph
	void addEdge(int v, int w);

	// Prints greedy coloring of the vertices
	void greedyColoring();
};

void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w);
	adj[w].push_back(v); // Note: the graph is undirected
}

// Assigns colors (starting from 0) to all vertices and prints
// the assignment of colors
void Graph::greedyColoring()
{
	int result[V];

	// Assign the first color to first vertex
	result[0] = 0;

	// Initialize remaining V-1 vertices as unassigned
	for (int u = 1; u < V; u++)
		result[u] = -1; // no color is assigned to u

	// A temporary array to store the available colors. True
	// value of available[cr] would mean that the color cr is
	// assigned to one of its adjacent vertices
	bool available[V];
	for (int cr = 1; cr < V; cr++)
		available[cr] = false;

	// Assign colors to remaining V-1 vertices
	for (int u = 1; u < V; u++)
	{
		// Process all adjacent vertices and flag their colors
		// as unavailable
		list<int>::iterator i;
		for (i = adj[u].begin(); i != adj[u].end(); ++i)
			if (result[*i] != -1)
				available[result[*i]] = true;

		// Find the first available color
		int cr;
		for (cr = 1; cr < V; cr++)
			if (available[cr] == false)
				break;

		result[u] = cr; // Assign the found color

		// Reset the values back to false for the next iteration
		for (i = adj[u].begin(); i != adj[u].end(); ++i)
			if (result[*i] != -1)
```

```
                                        available[result[*i]] = false;
        }

        // print the result
        for (int u = 1; u < V; u++)
                cout << "Vertex " << u << " ---> Color "
                        << result[u] << endl;
}

// Driver program to test above function
int main()
{
   //1=A , 2=B .... 20=T
   Graph g1(21);
   g1.addEdge(5, 6);
        g1.addEdge(5, 7);
        g1.addEdge(5, 15);

        g1.addEdge(2, 3);
        g1.addEdge(2, 4);
        g1.addEdge(2, 7);
        g1.addEdge(2, 13);
        g1.addEdge(2, 19);

        g1.addEdge(3, 8);
        g1.addEdge(3, 12);
        g1.addEdge(3, 18);
        g1.addEdge(3, 20);

        g1.addEdge(7, 9);


        g1.addEdge(1, 4);
        g1.addEdge(1, 5);
        g1.addEdge(1, 8);
        g1.addEdge(1, 11);
        g1.addEdge(1, 17);

        g1.addEdge(4, 13);
        g1.addEdge(4, 19);

        g1.addEdge(6, 7);
        g1.addEdge(6, 10);
        g1.addEdge(6, 17);
        g1.addEdge(6, 20);

        g1.addEdge(9, 11);
        g1.addEdge(9, 19);

        g1.addEdge(10, 14);
        g1.addEdge(10, 15);


        g1.addEdge(11, 10);

        g1.addEdge(8, 12);
        g1.addEdge(8, 17);
        g1.addEdge(8, 18);
```

```
        g1.addEdge(12, 16);
        g1.addEdge(12, 18);

        g1.addEdge(12, 15);

        g1.addEdge(14, 20);

        g1.addEdge(16, 19);
        g1.addEdge(16, 20);

        cout << "Coloring of graph  \n";
        g1.greedyColoring();


        return 0;
}
```

II-3.2 RESULTAT ET TRACE D'EXECUTION

A l'exécution de l'algorithme de que nous avons implémenté, nous le résultat suivant :

```
bgnebehi@scinfe172:/import/etud/32/bgnebehi/Bureau/stockage/Graphes$ ./coloration
Coloring of graph
Vertex 1 ---> Color 1
Vertex 2 ---> Color 1
Vertex 3 ---> Color 2
Vertex 4 ---> Color 2
Vertex 5 ---> Color 2
Vertex 6 ---> Color 1
Vertex 7 ---> Color 3
Vertex 8 ---> Color 3
Vertex 9  ---> Color 1
Vertex 10 ---> Color 2
Vertex 11 ---> Color 3
Vertex 12 ---> Color 1
Vertex 13 ---> Color 3
Vertex 14 ---> Color 1
Vertex 15 ---> Color 3
Vertex 16 ---> Color 2
Vertex 17 ---> Color 2
Vertex 18 ---> Color 4
Vertex 19 ---> Color 3
Vertex 20 ---> Color 3
```

Le sommet 1 représente A, le sommet 2 représente B ainsi de suite jusqu'au sommet 20 qui représente T.

Il appert alors qu'il faut 4 couleurs différentes pour la coloration du graphe

ETAPE 1 : Classer les nœuds du graphe dans l'ordre décroissant de leur degré

1- d°(A) =5
2- d°(B) =5
3- d°(C) =5

4- d°(F) =5
5- d°(H) =5
6- d°(D) =4
7- d°(E) =4
8- d°(G) =4
9- d°(K) =4
10-d°(S) =4
11-d°(T) =4
12-d°(I) =3
13-d°(J) =3
14-d°(L) =3
15-d°(O) =3
16-d°(P) =3
17-d°(Q) =3
18-d°(R) =3
19-d°(M) =3
20-d°(N) =3

ETAPE 2 : Donner une couleur à un nœud et attribuer cette même couleur à un nœud non coloré et non adjacent

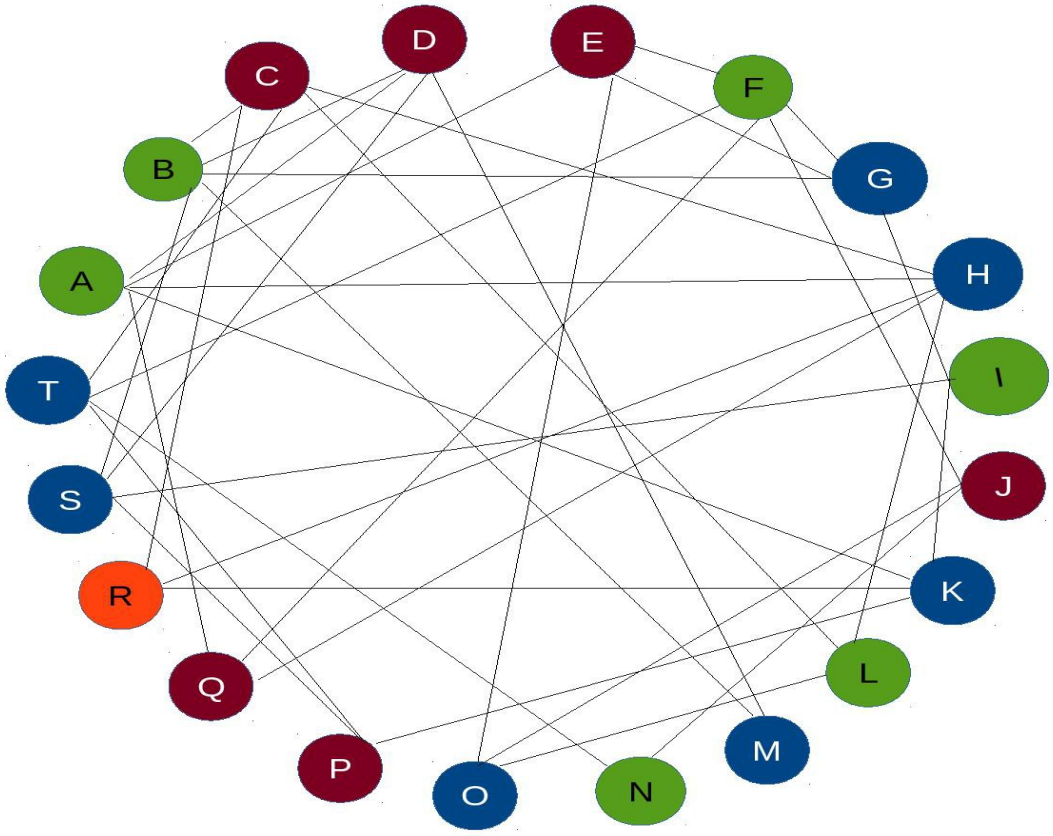
1- d°(**A**) =5
2- d°(**B**) =5
3- d°(C) =5
4- d°(**F**) =5
5- d°(H) =5
6- d°(D) =4
7- d°(E) =4
8- d°(G) =4
9- d°(K) =4
10-d°(S) =4
11-d°(T) =4
12-d°(**I**) =3
13-d°(J) =3
14-d°(L) =3
15-d°(O) =3
16-d°(P) =3
17-d°(Q) =3
18-d°(R) =3
19-d°(M) =3
*20-d°(**N**) =3*

ETAPE 3 : *S'il reste des nœuds non colorés revenir à l'étape 2*

1- d°(**A**) =5
2- d°(**B**) =5
3- d°(**C**) =5
4- d°(**F**) =5

5- d°(**H**) =5
6- d°(**D**) =4
7- d°(**E**) =4
8- d°(**G**) =4
9- d°(**K**) =4
10-d°(**S**) =4
11-d°(**T**) =4
12-d°(**I**) =3
13-d°(**J**) =3
14-d°(**L**) =3
15-d°(**O**) =3
16-d°(**P**) =3
17-d°(**Q**) =3
18-d°(**R**) =3
19-d°(**M**) =3
*20-d°(**N**) =3*

Les nœuds étant tous coloriés on obtient le graphe suivant :



II-4 INTERPRETATION

Le résultat de l'algorithme de Welsh-Powell est k=4.
Cela revient donc à dire que $\gamma(G) \approx 4$

Vérification d'admissibilité:

$\gamma(G) \leq r+1 = d°(A) +1 = 5+1= 6$
$\gamma(G) \leq n+1 - \alpha(G) = 20+1 - 4= 17$
$\gamma(G) \geqslant \omega (G) = 4$

$$4 \leq \gamma(G) \leq 17$$

III- BILAN/CONCLUSION

Au terme de notre étude, nous retenons que l'algorithme de **Welsh-Powell** nous permet de résoudre le problèmes d'allocation de fréquence.

Cette affirmation est déduite de l'application de l'algorithme de **Welsh-Powell** que **nous avons implémenté sur notre graphe d'étude**.

On en déduit que cet algorithme résout notre problématique de minimiser le nombre de fréquences.

Les recherches que nous avons effectuée tout au long de notre travail nous ont permis d'apprendre que la coloration de graphe est utile pour résoudre des problèmes de télécommunication comme celui qui nous à été soumis .