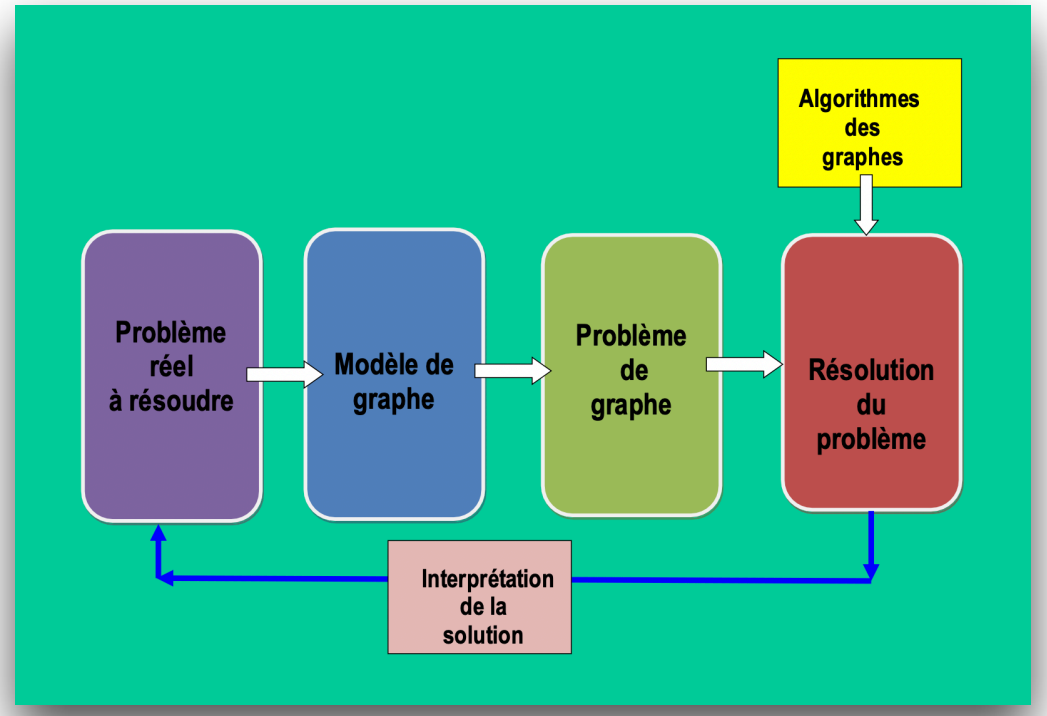
TD N° 3: Allocation de niveaux de vol à l'approche d'un porte-avion

Cholley Mathilde – Caussé Clara

I-POSITION DU PROBLEME

Le problème abordé ici est l'allocation de niveaux de vol à l'approche d'un porte-avion. En effet, à l'approche du pont d'un porte-avion, les aéronefs qui manœuvrent (décollage/atterrissage) doivent impérativement observer une distance de sécurité. Cette distance est censée prévenir tout risque de télescopage.
Une solution pour éviter les conflits aériens sans pour autant réduire le flux, consiste à allouer un niveau de vol (position en altitude) à chaque aéronef de sorte que deux aéronefs se trouvant à une distance inférieure à la distance de sécurité ne manœuvrent pas au même niveau de vol.
Mais l'espace aérien autour du porte avion étant restreint, le nombre de niveaux de vol est très limité. Le problème consiste donc à minimiser à tout instant le nombre de niveaux de vol attribué.



L'objectif du projet est donc de fournir une application pour calculer le nombre minimum de niveaux de vol à attribuer pour prévenir tout risque de télescopage entre les aéronefs.

Pour cela nous allons étudier ce cas réel et réaliser un modèle de graphe afin de le nombre minimum de niveaux de vol. Premièrement, nous allons proposer une méthode de construction du modèle de graphe pour représenter le système d'allocation de niveaux de vol puis appliquer cette méthode de modélisation à l'étude de cas présentée. Troisièmement, nous irons montrer que le problème d'attribution du nombre minimum de niveaux de vol se ramène, en général, à un problème de coloration de graphe. Enfin, nous formulerons précisément le problème ainsi posé et proposerons une solution particulière à l'étude de cas.

II-REALISATION

1) Présenter le modèle de graphe du sytème étudié

Formellement, on appelle graphe G :
1- un ensemble **S** d'**objets** appelés **sommets**,
2- un ensemble **A** de **relations** entre ces sommets.

Deux hypothèses se présentent:
1- les relations sont **symétriques**: on parle alors de **graphe non orienté**,
2- les relations ne sont **pas symétriques**: on parle alors de graphe orienté.

Ici, le moyen de résoudre ce problème est de commencer par modéliser le cas à l'aide d'un **graphe non orienté**, soit :

G=(S,A,) où S: ensemble fini de sommets, A: un ensemble fini d'arêtes.
Ils sont définis tel que :
- chaque sommet Si (i=1,...,8) représente un aéronef Ai;
- chaque arête (Si, Sj) peut montrer le risque de collision entre Ai et Aj (Ai bouge à distance critique de Aj)

2) Montrer comment on ramène le problème réel à un problème relevant de théorie des graphes:

Dans le sujet du TP, on nous donne ce tableau ci-dessous :

L'aéronef	A1	A2	A3	A4	A5	A6	A7	A8
...risque d'entrer en collision avec	A3 A6 A8	A4 A6 A7	A1 A5 A7 A8	A2 A6 A8	A3 A7 A8	A1 A2 A4	A2 A3 A5	A1 A3 A4 A5

On considère la période durant laquelle 8 aéronefs, notés **A1, A2, ... A8** manœuvrent à l'approche du porte-avion dans la situation décrite par le tableau ci-dessous. Pour chaque aéronef, on indique la liste des aéronefs évoluant à une distance **inférieure** à la distance de sécurité.

CAS N°1 : SOMMETS NON ADJACENTS

Chaque aéronef Ai1, Ai2, Ai3... qui ne sont pas à distance critique peuvent être représentés par des sommets (S1, S2, S3...) non adjacents.
Dans un graphe coloré, les sommets non adjacents ont alors la possibilité d'avoir la même couleur. En effet, la couleur utilisée pour les sommets correspond aux niveaux de vols dans lequel ces aéronefs manoeuvrent.

On parle alors de graphes colorés.

CAS N°2 : SOMMETS ADJACENTS

A l'inverse, nous avons alors des aéronefs Ai'1, Ai'2, Ai'3... qui sont cette fois ci à distance critique et qui sont représentés par des sommets adjacents.
Ainsi, ici dans ce graphe coloré les sommets adjacents ont alors une couleur différente et chaque couleur pourrait correspondre à un niveau de vol.

Le problème posé se ramène donc bien à un problème relevant de la théorie des graphes qui ici s'expose par la coloration de graphe ainsi qu'avec le calcul du nombre chromatique...

---

3) Exposer la solution retenue mettant en oeuvre une algorithmique de la théorie des graphes :

En effet, comme expliqué ci-dessus, le problème réel étudié relève de la théorie des graphes. Ici il s'expose par la coloration de graphe pour trouver le nombre minimum de vol affecté aux aéronefs.
Pour trouver ce nombre minimum k « pas trop grand », on utilise alors le nombre chromatique $\gamma(G)$ d'un graphe coloré.
Soit : $k=\gamma(G)$.

En revanche, le problème de calcul de ce nombre chromatique relève d'un problème de NP-Complet.
Pour cela, il existe un algorithme polynomial qui peut donner une solution approchée : l'algorithme de Welsh-Powell.

Nous allons donc calculer le nombre chromatique en utilisant l'algorithme de welsh-Powell.

La mise en oeuvre de cet algorithme implique 3 étapes :

ETAPE N°1 :

1- Classer les sommets du graphe dans l'ordre **décroissant** de leur degré.
2- Attribuer à chacun des sommets son numéro d'ordre dans la liste triée.

ETAPE N°2 :

En parcourant la liste des sommets **dans l'ordre** de tri:

1- attribuer une couleur **non encore utilisée** au premier sommet non encore coloré

2- attribuer cette **même couleur** à chaque sommet **non encore coloré** et **non adjacent** à un sommet de **cette couleur**.

ETAPE N°3 :

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

On a alors créé un algorithme nommé, tp3.cpp
Comme défini en haut on a :
G=(S,A) avec

- S = {A1, A2, A3, A4, A5, A6, A7, A8}
- A = {(A1,A3), (A1,A6), (A1,A8), (A2,A4), (A2,A6), (A2,A7), (A3,A5), (A3,A7), (A3,A8), (A4,A6), (A4,A8), (A5,A7), (A5,A8)}

Voici quelques capture de notre algorithme traduisant notre cas réel :

```cpp
// 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
```

```cpp
// 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 = 0; 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 = 0; 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 = 0; u < V; u++){
        cout << "Vertex " << u << " --->  Color "
             << result[u] << endl;
    }
}

// Driver program to test above function
int main()
{
    Graph g1(8);
    g1.addEdge(0, 2);
    g1.addEdge(0, 5);
    g1.addEdge(0, 7);

    g1.addEdge(1, 3);
    g1.addEdge(1, 5);
    g1.addEdge(1, 6);

    g1.addEdge(2, 4);
    g1.addEdge(2, 6);
    g1.addEdge(2, 7);

    g1.addEdge(3, 5);
    g1.addEdge(3, 7);

    g1.addEdge(4, 6);
    g1.addEdge(4, 7);

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



    return 0;
}
```

Dans cet algorithme, on peut voir un constructeur et un destructeur dans une classe Graph. Et également une procédure pour la coloration du graphe (d'après Wels-Powell).

---

4) En s'appuyant sur le ou les exemples proposés, donner une interprétation des résultats dans le cadre du problème réel.

Lors de l'exécution de notre algorithme on obtient alors sur le terminale cette capture suivante :

```
Coloring of graph
Vertex 0 --->  Color 0
Vertex 1 --->  Color 0
Vertex 2 --->  Color 1
Vertex 3 --->  Color 1
Vertex 4 --->  Color 0
Vertex 5 --->  Color 2
Vertex 6 --->  Color 2
Vertex 7 --->  Color 2
```

On voit alors que l'on a color0, 1 ou 2 qui apparaissent qui est l'équivalent de la couleur n°1, la couleur n°2 et la couleur n°3.
Soit nous avons donc $\chi(G) \approx 3$ (en effet car c'est une valeur approchée).

Vérifions avec la vérification d'admissibilité :
$\chi(G) \leq r + 1 = d°(3) + 1 = 4 + 1 = 5$

$\chi(G) \leq n + 1 - \alpha(G) = 8 + 1 = 6$

$\chi(G) \geq \omega(G) = 3$

Regardons maintenant ces résultats dans le cas de notre problème réel relié au TPN°3.

On a donc le nombre chromatique qui est égale plus ou moins au nombre 3. Cela traduit donc que le nombre minimum de niveaux de vol à donner pour éviter tous risques de collisions entre des avions qui manoeuvrent est de 3.

III-CONCLUSION

Dans ce TP N°3, nous avons pu apprendre à partir d'un cas réel à en déduire une théorie des graphes sur la coloration de graphe.
Nous avons pu également découvrir comment calculer un nombre chromatique qui ici traduit le nombre minimum de vol affecter aux aéronefs. Ce nombre étant traduit par l'algorithme de Welsh-Powell.

Ainsi, nous pouvons réaliser l'importance des graphes pour des cas réels comme ici sur la sécurité.

Ce TP nous a également donné un exemple concret de la coloration des graphes qui pourra nous servir plus tard pour une autre utilisation qui s'y rapprochera.