GNEBEHI BAGRE JEAN-PHILIPPE

L3 INFORMATIQUE

2017-2018

OPTIMISATION DU CYCLE D'ASSEMBLAGE D'UN SMARTPHONE

Khadir Ouriachi

Enseignant Chercheur à Université de Pau

I- POSITION DU PROBLEME

De nos jours la production des appareils comme le smartphone est en pleine apogée.
Il est donc important d'améliorer la qualité de production et augmenter la fiabilité de ces smartphones.
Dans le cadre de notre formation à l'Université de Pau, nous nous confrontons à un problème d'**optimisation du cycle d'assemblage d'un smartphone**.

Le problème soumis à notre sagacité consiste donc à disposer les tâches de sorte à maximiser la **parallélisation** des tâches.
Cela revient à mettre en évidence dans le cycle de montage **le nombre minimum de tâches incompatibles**.

En d'autres termes nous sommes amenés à mettre en exergue dans le cycle de montage le nombre minimum de tâches.

II- REALISATION

II-1 GRAPHE

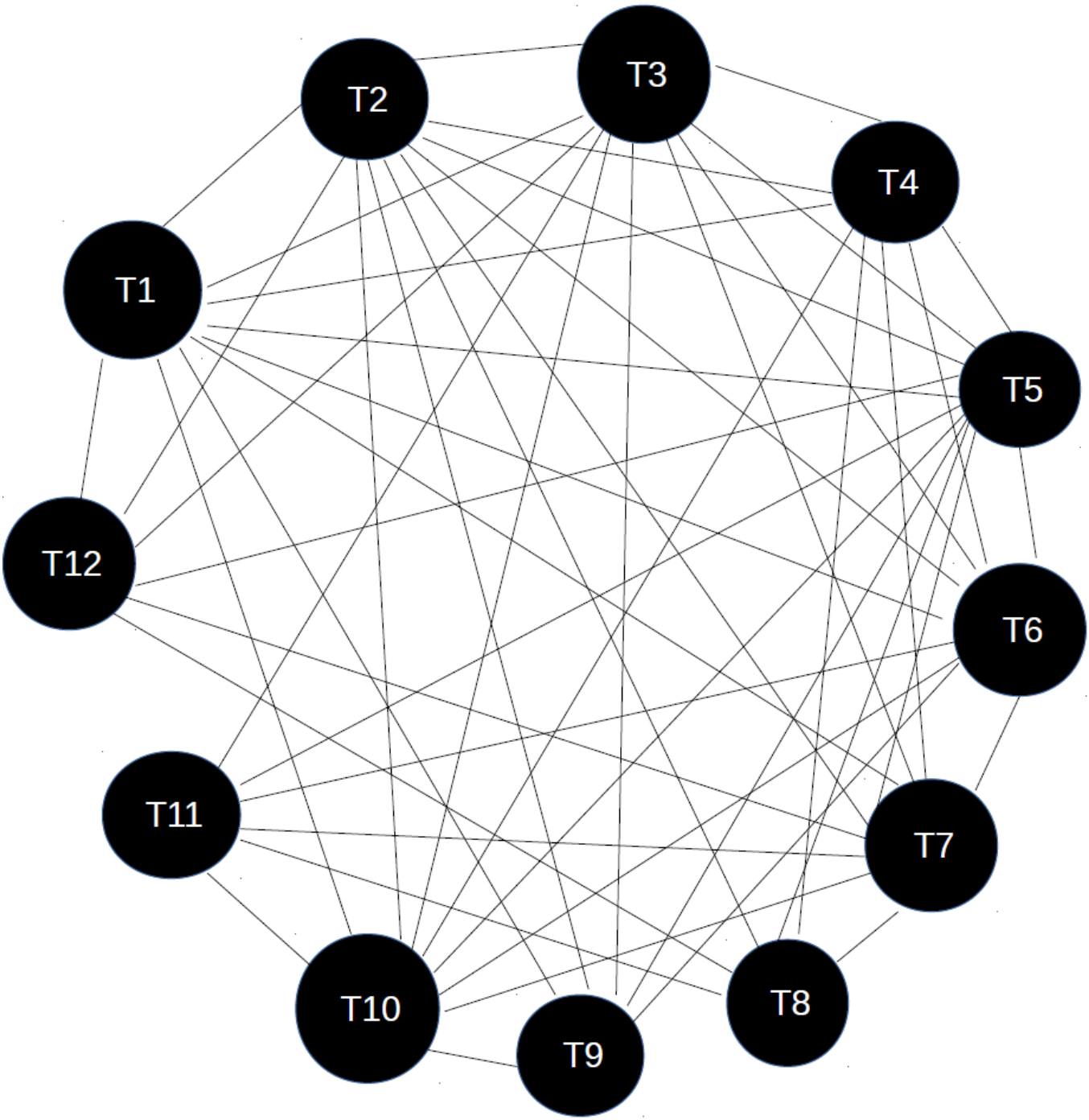
Pour formaliser le problème nous allons modéliser un graphe qui met en évidence la préemption des tâches à l'aide d'un graphe non orienté G=(S,A), défini comme suit :

-l'ensemble S des nœuds représente l'ensemble des tâches
-l'ensemble A des arêtes qui formalisent la relation d'incompatibilité entre les tâches

Après avoir analyser et étudier le tableau soumis à notre compréhension nous déduisons donc le tableau ci-dessous .
Ce tableau montre les différentes **relations d'incompatibilité entre les différentes tâches .**

Tâches	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Incompatibe avec	T2 T3 T4 T5 T6 T7 T9 T10 T12	T1 T3 T4 T5 T6 T7 T8 T9 T10 T12	T1 T2 T4 T5 T6 T7 T9 T10 T11 T12	T1 T2 T3 T5 T6 T7 T8 T10 T11	T1 T2 T3 T4 T6 T7 T8 T9 T10 T11 T12	T1 T2 T3 T4 T5 T7 T9 T10 T11	T1 T2 T3 T4 T5 T6 T8 T10 T11 T12	T2 T4 T5 T7 T11 T12	T1 T2 T3 T5 T6 T10	T1 T2 T3 T4 T5 T6 T7 T9 T11	T3 T4 T5 T6 T7 T8 T10	T1 T2 T3 T5 T7 T8

A partir du tableau ci-dessus nous sommes désormais à mesure de construire notre graphe d'étude .
Il apparaît comme suit:



II-2 PROBLEME DE COLORATION DE GRAPHE

En appliquant une coloration au modèle de graphe proposé plus haut, les nœuds qui représentent les tâches incompatibles sont adjacents .

Le problème peut être ramené à un problème classique de recherche du nombre minimum pour la coloration du graphe.

Il faut ainsi calculer le **nombre chromatique** du modèle de graphe.

Sur le modèle de graphe, les nœuds portant une même couleur représentent des tâches compatibles d'autant qu'elles ne priorise pas les même robots.

II-3 SOLUTION AU PROBLEME

II-2.1 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-2.2 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 = 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 = 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=T1 2=T2 3=T3 4=T4 5=T5 6=T6 7=T7 8=T8 9=T9 10=T10 11=T11 12=T12
   Graph g1(13);
   g1.addEdge(5, 6);
        g1.addEdge(5, 7);
        g1.addEdge(5, 8);
        g1.addEdge(5, 9);
        g1.addEdge(5, 10);
        g1.addEdge(5, 11);
```

```
        g1.addEdge(5, 12);

        g1.addEdge(2, 3);
        g1.addEdge(2, 4);
        g1.addEdge(2, 5);
        g1.addEdge(2, 6);
        g1.addEdge(2, 7);
        g1.addEdge(2, 8);
        g1.addEdge(2, 9);
        g1.addEdge(2, 10);
        g1.addEdge(2, 12);

        g1.addEdge(3, 4);
        g1.addEdge(3, 5);
        g1.addEdge(3, 6);
        g1.addEdge(3, 7);
        g1.addEdge(3, 9);
        g1.addEdge(3, 10);
        g1.addEdge(3, 11);
        g1.addEdge(3, 12);

        g1.addEdge(7, 8);
        g1.addEdge(7, 10);
        g1.addEdge(7, 11);
        g1.addEdge(7, 12);

        g1.addEdge(1, 2);
        g1.addEdge(1, 3);
        g1.addEdge(1, 4);
        g1.addEdge(1, 5);
        g1.addEdge(1, 6);
        g1.addEdge(1, 7);
        g1.addEdge(1, 9);
        g1.addEdge(1, 10);
        g1.addEdge(1, 12);

        g1.addEdge(4, 5);
        g1.addEdge(4, 6);
        g1.addEdge(4, 7);
        g1.addEdge(4, 8);
        g1.addEdge(4, 10);
        g1.addEdge(4, 11);
        g1.addEdge(4, 12);

        g1.addEdge(6, 7);
        g1.addEdge(6, 9);
        g1.addEdge(6, 10);
        g1.addEdge(6, 11);
```

```
        g1.addEdge(9, 10);

        g1.addEdge(11, 10);

        g1.addEdge(8, 11);

        g1.addEdge(8, 12);

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

        return 0;
}
```

II-2.3 RESULTAT ET TRACE D'EXECUTION

```
Coloring of graph
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 3
Vertex 4 ---> Color 4
Vertex 5 ---> Color 5
Vertex 6 ---> Color 6
Vertex 7 ---> Color 7
Vertex 8 ---> Color 1
Vertex 9 ---> Color 4
Vertex 10 ---> Color 8
Vertex 11 ---> Color 2
Vertex 12 ---> Color 6
```

Le sommet 1 représente T1 , le sommet 2 représente T2
ainsi de suite jusqu'au sommet 12 .
Il appert alors qu'il faut 8 couleurs pour la coloration du graphe .

Au départ :

Dans le graphe non orienté G ,il n'existe aucun nœud coloré

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

d°(T5) = 11
d°(T2) = 10
d°(T3) = 10
d°(T7) = 10
d°(T1) = 9
d°(T4) = 9
d°(T6) = 9
d°(T10) = 9
d°(T11) = 7
d°(T8) = 6
d°(T9) = 6
d°(T12) = 6

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

d°(**T5**) = 11
d°(T2) = 10
d°(T3) = 10
d°(T7) = 10
d°(T1) = 9
d°(T4) = 9
d°(T6) = 9
d°(T10) = 9
d°(T11) = 7
d°(T8) = 6
d°(T9) = 6
d°(T12) = 6

Il n'existe aucun nœud non adjacents à T5
ETAPE 3 : S'il reste des nœuds non colorés revenir à l'étape 2

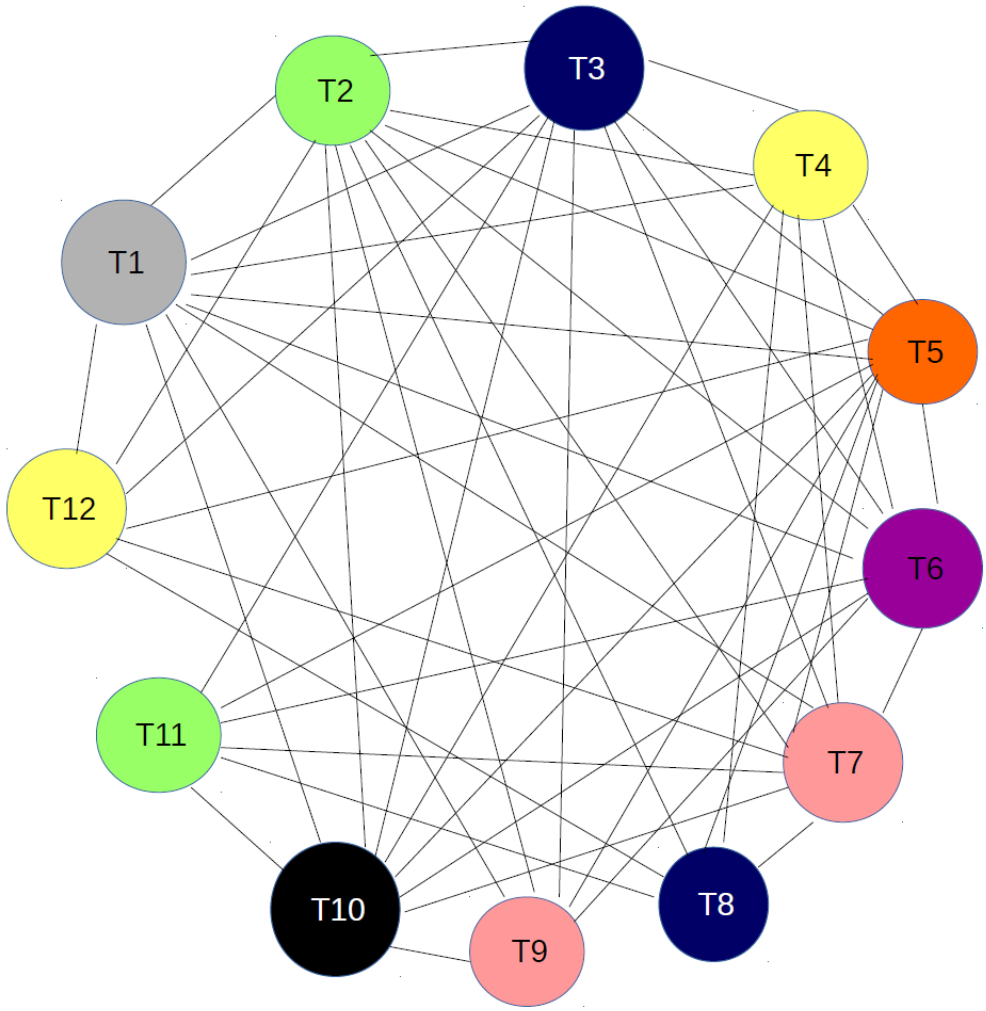
T2 et T11 sont non adjacent alors ils héritent de la même couleur

d°(**T5**) = 11
d°(**T2**) = 10
d°(T3) = 10
d°(T7) = 10
d°(T1) = 9
d°(T4) = 9
d°(T6) = 9
d°(T10) = 9
d°(**T11**) = 7
d°(T8) = 6
d°(T9) = 6
d°(T12) = 6

Les nœuds n'étant encore pas tous colorés on revient à l'étape 2

d°(**T5**) = 11
d°(**T2**) = 10
d°(**T3**) = 10
d°(**T7**) = 10
d°(**T1**) = 9
d°(**T4**) = 9
d°(**T6**) = 9
d°(**T10**) = 9
d°(**T11**) = 7
d°(**T8**) = 6
d°(**T9**) = 6
d°(**T12**) = 6

Ainsi en appliquant l'algorithme de Wesh-Powell il appert que tous les nœuds non adjacents héritent de la même couleur.
Les nœuds étant tous coloriés alors la coloration du graphe G non orienté est terminée.On obtient donc le graphe coloré suivant :



II-4 INTERPRETATION

Le résultat de l'algorithme de Welsh-Powell est k=8 .
Cela revient donc à dire que $\gamma(G) = 8$

Vérification d'admissibilité:

$\gamma(G) \leq r+1 = d°(T5) +1 = 11+1= 12$
$\gamma(G) \leq n+1 - \alpha(G) = 12+1 - 2= 11$
$\gamma(G) \geqslant \omega(G) = 8$

$8 \leq \gamma(G) \leq 11$

III – CONCLUSION

Au terme de notre étude, nous retenons que l'algorithme de Welsh-Powell nous permet de procéder à la coloration des graphes en théorie des graphes.

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 problème classique de recherche du nombre minimum pour la coloration du graphe .

La perspicacité que nous avons mis dans nos recherches nous a permis de comprendre que la coloration de graphe est efficace pour optimiser l'utilisation des machines de travail c'est à dire comment mettre en parallèle des fabrications utilisant plusieurs machines.

Ou encore comment faire cohabiter des personnes ou des animaux en tenant compte de leur incompatibilité .