1 Algorithme de base

```
Structure globale :
T, l'ensemble des ensembles stables maximum
taille, la taille d'un ensemble stable maximum
Entrée:
Un graphe non orienté G= [S,A].
Un ensemble stable en cours de calcul Ss
Sortie: Rien
1: function EnuStableMax(G,Ss)
2: Si (S=Æ) Alors
3: Si (|Ss|>taille) Alors
4: Vider T et taille ← 0
5: T ← {Ss} et taille ← |Ss|
6: Sinon Si (|Ss|=taille) Alors
7: T ← {Ss}
8: FinSi
9: FinSi
10 : Sinon
11: Pour i=1 à |S| Faire
12: s ← S[i] // s est le sommet en position i dans la liste des sommets restants S
13: Ss ← Ss+ {s}
14: S ← S - {s}
15: Retirer de S tous les sommets ayant une arrête commune avec s
16: EnuStableMax(G,Ss)
17: Remettre dans S les sommets supprimés à l'étape 14 et 15
18: Fin Pour
19: Fin Si
```

Cette fonction parcours l'ensemble des sommets du graphe, puis prélève un premier sommet du graphe l'ajoute à la solution puis supprime les sommets liés à ce sommet prélevé et recommence l'opération jusqu'à ce qu'il n'y ai plus de sommet à parcourir. On calcule ainsi l'ensemble des stables possible du graphe et on ne garde que ceux qui ont la taille la plus grande.
Cet algorithme est facilement réalisable, mais il n'est pas très efficace.

Voici plusieurs point qu'il semble intéressant de chercher à améliorer :

- On s'aperçoit dans un premier temps que chaque stable calculé au cours de l'algorithme est recalculé autant de fois qu'il y a de sommet dans ce stable, l'algorithme ne détecte pas si il a déjà trouvé un stable et considère deux stables différents si ils ont un ordre de sommet différent.
  Il serait donc intéressant de pouvoir limité ces calculs inutiles.

- Si on s'intéresse au pire des cas pour notre algorithme, on s'aperçoit que moins le graphe ne comporte d'arête, moins les sommets sont liés et plus l'algorithme sera long. Dans notre algorithme le pire des cas et un graphe sans arête où tous les sommets sont déjà indépendants (on obtient un temps maximum de $n!$ avec n le nombre de sommets).

- Lorsque notre algorithme tourne, les sommets sont pris " au hasard " dans l'ordre dans lequel ils ont été numéroté. Si on choisie de prendre les sommets arbitrairement selon une certaine règle on peut peut-être amélioré l'algorithme.

Voici les améliorations qui pourraient être faites :

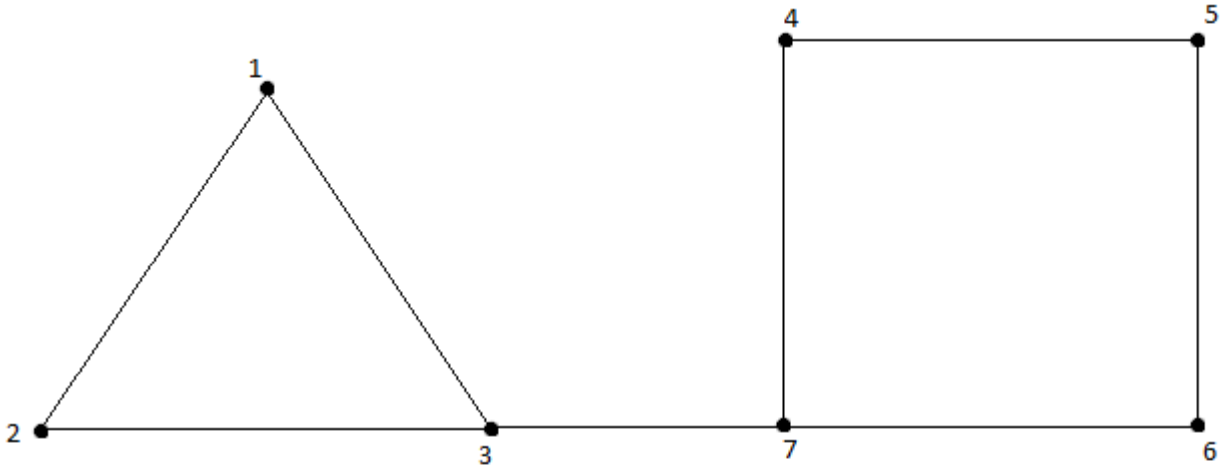
- Dans le cas des solutions recalculées, il ne semble pas évident de pouvoir vérifier qu'une solution ai été déjà calculé, nous verrons plus tard si une amélioration peut-être réalisable.

- Par contre, en ce qui concerne le pire des cas, on sait déjà que l'on peut créé une règle permettant d'ajouter automatiquement tout les sommets du graphes qui n'ont aucune arêtes dans la solution (car ils y seront nécessairement). Mais une fois cette règle mise en place, le pire des cas revient à un graphe qui aurait n paires de sommet reliés par une arête, en imaginant que l'on fasse une autre règle spécifique pour ce cas précis, on ne ferait que décaler le problème à un graphe qui aurait n ensemble de sommet reliés entre eux. Ce qui nous amène à considérer ce problème de façon plus large, en réalité ce qui se cache derrière tout ceci est de savoir si notre graphe est connexe ou non. Dans le cas ou le graphe n'est pas connexe, on peut savoir que si n est le nombre de parties connexe qui forment le graphe, le nombre minimum de sommets dans un stable est n. Si on arrive à diviser le graphe non connexe en plusieurs graphe connexe il ne reste alors qu'à faire tourner l'algorithme sur chacun de ces sous graphe et on aurait la solution optimale du stable maximum.
  Si l'on s'intéresse à la connexité d'un graphe, on peut même aller plus loin. Si l'on considère un graphe connexe, on peut le diviser en plusieurs ensembles connexes, puis chercher le stable maximum pour chacun de ses sous graphes et rassembler les résultats pour obtenir notre solution. Cette amélioration diminue grandement la complexité de notre algorithme en limitant le pire des cas (il reste à déterminer un moyen de faire cela). Si l'on prend un graphe qui se divise en deux sous graphes connexes reliés entre eux par une arête (figure 1) , il parait évident que l'on peut diviser ce graphe en deux sous graphes puis recalculer les solutions en enlevant l'un ou l'autre des sommets qui servent à la liaison des deux graphes (si ils apparaissent dans les sous solutions).

- Concernant le choix des sommets lors du parcours, on peut déjà avant même de travailler sur le graphe ordonner ses sommets par ordre croissant de nombre d'arêtes avec un tri rapide. Cela permet de parcourir en priorité les sommets qui ont un faible nombre d'arête, ce qui normalement permet d'obtenir une solution optimale plus rapidement. Cela revient à choisir le sommet ayant le nombre d'arête minimum à chaque fois dans l'ensemble des sommets restant. On pourrait aussi rajouter une règle faisant arrêter l'algorithme dès lors que l'on part d'un sommet ayant un nombre d'arêtes trop grand pour que l'on puisse tomber sur une solution qui soit optimale.

Lorsque l'on regarde l'ensemble des stables maximums pour un graphe donné, on s'aperçoit qu'un ensemble de sommets apparait à chaque fois dans ces solutions seuls certains sommets changent. Y a-il un moyen permettant de connaitre cette ensemble afin de trouver les solutions plus rapidement ?

Ils nous reste à déterminer un algorithme afin de diviser un graphe en sous graphe connexe, nous avons commencé à regarder l'algorithme de tarjan (même si il ne s'applique que à des graphes orienté), nous nous intéressons aux différents types de décomposition d'un graphe (modulaire, cographe).



Solution : {1, 4, 6} {1, 7, 5 } {2, 4, 6} {2, 7, 5} {3, 4, 6}
Si on séparre les deux graphes :

Graphe Triangle : {1} {2} {3}
Graphe Carré : {4, 6} {5, 7}
Les solutions du graphe total sont donc des combinaisons des solutions du triangle et du carré {1} + {4, 6} = {1, 4, 6}
Bien entendu la solution {3} + {5, 7} n'est pas possible les combinaisons dans lesquelles il y a les deux sommets qui relient les deux graphes (3 et 7) ne sont pas réalisables.