# Algorithmique et Programmation
## Projet : Algorithme de Bron-Kerbosch pour Maximum-Clique

Ecole normale supérieure
Département d'informatique
algoL3@di.ens.fr

2013-2014

## 1 Contexte

On s'intéresse au problème des gardiens de musée. Un musée est schématiquement formé de couloirs, où des peintures sont accrochées. Les couloirs doivent être surveillés par des gardiens. Quand un gardien est positionné au croisement de plusieurs couloirs, il surveille tous les couloirs attenants. Les gardiens du musée en question, tout comme la direction, ont à cœur que la surveillance ait effectivement lieu. La direction a utilisé un algorithme glouton pour assigner les gardiens à des croisements.

Cependant, il semble aux gardiens qu'ils sont légèrement en sur-effectif et ils cherchent donc à déterminer le nombre minimal de gardiens nécessaires à la surveillance du musée.

Formellement, les couloirs du musée sont les arêtes d'un graphe, et le problème consiste à placer des gardiens sur les sommets de ce graphe de telle sorte que chaque arête soit attenante à un gardien (au moins). On cherche à placer le moins de gardiens possibles. On ne connaît pas d'algorithme polynomial pour résoudre ce problème (qui est en fait le Minimum Vertex Cover). Le problème est au passage NP-complet.

## 2 Terminologie

Dans un graphe arbitraire, une Clique est un ensemble de sommets qui sont tous reliés deux-à-deux. Très clairement, dans un graphe, la taille de la plus grande clique du graphe est bien définie. Le problème Maximum-clique consiste à trouver une clique de taille maximale dans un graphe.

De même, un Independent Set est un ensemble de sommets qui ne sont *pas* reliés entre eux (aucune paire n'est reliée). C'est donc le contraire d'une clique, et il y a une correspondance entre les cliques d'un graphe et les independent sets du graphe complémentaire (où $\overline{E} = V^2 - E$).

Enfin, un Vertex Cover est un ensemble de sommets qui touche toutes les arêtes du graphe. Il n'est pas difficile de voir que le complémentaire d'un independent set est un vertex cover. Par conséquent, un Minimum Vertex Cover peut être obtenu à partir d'un Maximum Independent Set, et réciproquement.

Une clique peut être contenue dans une autre clique plus grande. Quand ce n'est pas le cas, elle est dite maximale (attention, une clique maximale n'est pas forcément une clique maximum ! Maximale signifie qu'elle ne peut pas être agrandie. Maximum signifie qu'il n'en existe pas de plus grande).

On note $\Gamma(x)$ les sommets adjacents à $x$ dans un graphe donné. Étant donné un sous-ensemble $W$ de $V$, le sous-graphe induit par $W$ est le graphe formé des sommets $W$ et des arêtes de $E$ qui relient des sommets de $W$.

## 3 Algorithme de Bron-Kerbosch

L'algorithme de Bron-Kerbosch a été décrit en 1973 par deux chercheurs hollandais, Joep Kerbosch et Coenraad Bron. Leur algorithme consiste à énumérer toutes les cliques maximales d'un graphe, sachant que les cliques maximum font partie du lot. Il est connu qu'un graphe à $n$ sommets possède au maximum $3^{n/3}$ cliques maximales, et on connaît une famille de graphe qui atteint cette borne. Ceci donne donc une borne inférieure en temps de $\mathcal{O}(1.44^n)$ dans le pire des cas.

**Version de base.** L'idée est que si on a trouvé une clique $K = \{k_1, \ldots, k_m\}$ dans un graphe $G = (V, E)$, alors il suffit de regarder $C = \Gamma(k_1) \cap \cdots \cap \Gamma(k_m)$. Si $C = \emptyset$, alors la clique $K$ est maximale. Pour en trouver d'autre, il faut donc revenir sur le dernier sommet ajouté à $K$ et explorer les autres possibilités. Sinon, $K$ peut être agrandie en une clique plus grande par l'adjonction d'un sommet de $C$, qui est

l'ensemble des candidats à l'agrandissement. Simplement, une fois qu'on a réussi à trouver une clique $K$, on peut se contenter de travailler sur le sous-graphe induit par $C$.

1: **function** MaxClique($G, K, C$)
2: **if** $C = \emptyset$ **then** noter la clique maximale $K$
3: **else for each** $x \in C$ **do** MaxClique($G, K \cup \{x\}, C \cap \Gamma(x)$)
4: **end function**

L'inconvénient de cette manière de faire est que la même clique maximale peut être signalée plusieurs fois. Par exemple, s'il existe une arête $1 \leftrightarrow 2$ dans un graphe, alors la clique $\{1, 2\}$ sera trouvée à partir de $K = \{1\}$ et $K = \{2\}$, de même que toutes les cliques qui contiennent $\{1, 2\}$.

**Candidats autorisés.** On peut empêcher ceci en disant qu'une fois qu'on a énuméré toutes les cliques qui contiennent le sommet $x$, alors on pourra s'interdire de considérer $x$ dans toute la suite. On maintient donc trois ensembles : la clique courante $K$, un ensemble de candidats $C$, et un sous-ensemble $A$ de $C$ formé des sommets qu'on s'autorise à ajouter à $K$. On aboutit alors à un algorithme :

1: **function** MaxClique($G, K, C, A$)
2: **if** $C = \emptyset$ **then** noter la clique maximale $K$ **else**
3: **while** $A \neq \emptyset$ **do**
4: **choose** $x \in A$
5: MaxClique($G, K \cup \{x\}, C \cap \Gamma(x), A \cap \Gamma(x)$)
6: $A \leftarrow A - \{x\}$
7: **end while**
8: **end function**

**Utilisation de pivots.** Cette nouvelle version ne signale chaque clique maximale qu'une seule fois, ce qui est un progrès. Cependant, elle effectue un appel récursif par clique, maximale ou non. Pour améliorer ça, une idée consiste à choisir un sommet $u \in C$ (le "pivot"), et à supposer qu'on a énuméré toutes les cliques maximales contenant $K \cup \{u\}$. Alors, nous affirmons que chaque nouvelle clique contenant $K$ mais pas $u$ doit contenir un nouveau sommet qui n'est pas adjacent à $u$. Justification (par l'absurde) : on étend $K$ en une clique maximale $K \cup S$, où tous les éléments de $S$ sont adjacents à $u$. Alors, comme $u$ est adjacent à tous les éléments de $K$, on trouve que $K \cup S \cup \{u\}$ est une clique plus grosse, ce qui contredit la maximalité de la précédente. Il en découle immédiatement un nouvel algorithme :

1: **function** MaxClique($G, K, C, A$)
2: **if** $C = \emptyset$ **then** noter la clique maximale $K$ **else**
3: **let** $u \in C$ tel que $|C \cap \Gamma(u)|$ est maximal **in**
4: **while** $A - \Gamma(u) \neq \emptyset$ **do**
5: **choose** $x \in A - \Gamma(u)$
6: MaxClique($G, K \cup \{x\}, C \cap \Gamma(x), A \cap \Gamma(x)$)
7: $A \leftarrow A - \{x\}$
8: **end while**
9: **end function**

Tomita, Tanaka et Takahashi ont démontré qu'il est possible d'implémenter l'algorithme de telle sorte qu'il exécute $\mathcal{O}\left(3^{n/3}\right)$ opérations dans le pire des cas (c'est-à-dire un nombre *constant* d'opérations par clique trouvée). Un des avantages de l'algorithme de Bron-Kerbosch est que, mis à part dans des cas très dégénérés, il trouve rapidement de grosses cliques, voire même la plus grosse clique, et passen ensuite le reste de son temps à vérifier que c'est bien la meilleure. Seulement, la réponse est déjà disponible. Si une approximation de la taille de la plus grosse clique est suffisante, on peut faire tourner l'algorithme quelques minutes et s'en tenir là. On peut aussi noter que l'algorithme permet facilement de trouver une clique maximum particulière (comme une clique maximum de poids minimum).

**Optimisation pour une clique maximum.** Si on peut se contenter d'une clique maximum, au lieu de l'énumération de toutes les cliques maximales, alors on peut arrêter la récursion dès que $|K|+|C| \leq |K_{\max}|$ (où $K_{\max}$ désigne la plus grande clique trouvée à présent) car alors on est certain de ne pas trouver de meilleure clique.

**Heuristique de coloriage.** Un coloriage d'un graphe est une fonction qui associe une couleur à chaque sommet, de telle sorte que deux sommets adjacents sont toujours de couleur différente. Le nombre chromatique d'un graphe est le plus petit nombre de couleur nécessaire pour colorier le graphe en question. Il est clair que le nombre de couleurs de n'importe quel coloriage *majore* la taille de la plus grande clique

d'un graphe. Déterminer le nombre chromatique est au moins aussi dur que Maximum-Clique, mais on peut utiliser un algorithme glouton pour trouver un coloriage approximatif.

On peut donc, à un moment donné de l'algorithme, tenter de trouver un coloriage du sous-graphe induit par $C$, pour voir si par hasard il n'utiliserait pas moins de couleurs que $|K_{\max}|$. Le cas échéant, on a aucune chance de trouver une meilleure clique en explorant les candidats.

# 4 Travail demandé

Vu le caractère exponentiel des calculs à réaliser, et la (relative) simplicité de l'algorithme, on accordera une certaine importance à la performance de l'implémentation. Cependant, elle doit être correcte avant tout ! Vous pouvez tester plusieurs structures de données pour représenter les ensembles de sommets, et choisir la plus performante. Vous pouvez aussi (c'est optionnel) tester l'heuristique de coloriage.

Vous testerez votre algorithme sur des graphes aléatoires, mais aussi sur les graphes du DIMACS Challenge for Maximum Clique.