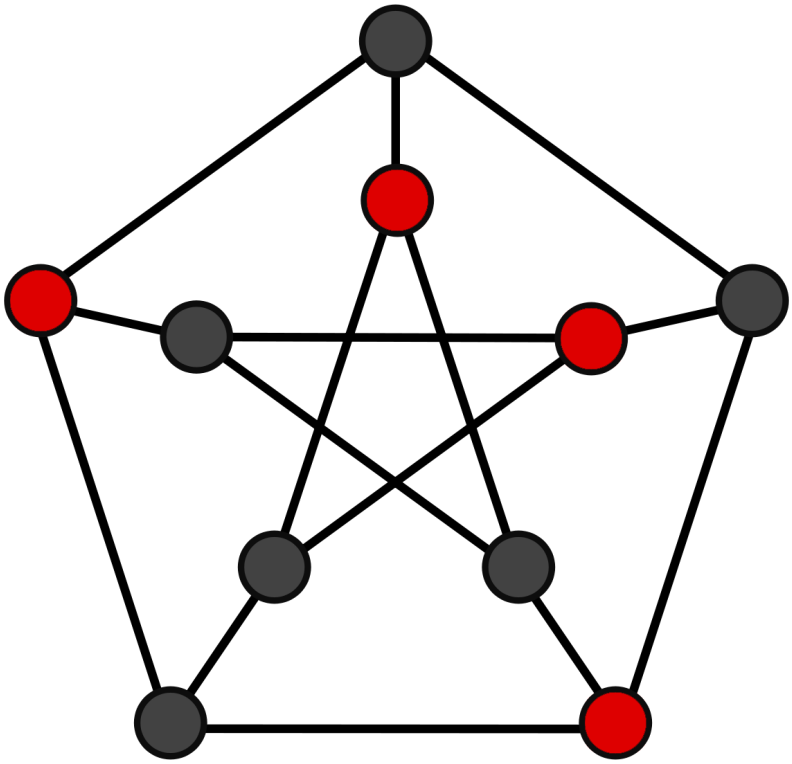
Problème du Stable Maximum

Projet final, mai 2017



Guillaume Cocatre-Zilgien
UFR Sciences Exactes et Expérimentales,
Université de Perpignan Via Domitia
Licence Informatique, Semestre 4
Professeur: Michel Ventou

Sommaire

Introduction

Ce projet nous a été assigné par notre professeur d'algorithmique dans le cadre d'une nouvelle matière de 2ème année, sobrement nommée "Projet" et dénuée de cours magistraux : l'intégralité des heures d'enseignement a été consacrée à des séances de Travaux Pratiques, durant lesquelles M. Ventou s'est tenu à notre disposition pour répondre à nos questions.

Nous avons joui d'une totale liberté de travail, et une saine ambiance d'émulation et de collaboration s'est développée peu à peu parmi nous. Ce rapport, et surtout le code source du programme qui l'accompagne, sont le fruit de plusieurs mois de travail, dont j'ai passé la majeure partie à modifier et améliorer l'implémentation des algorithmes glanés sur le Web.

Objectif

Trouver un algorithme exact et une heuristique pour la recherche de stable maximum, et les implémenter dans un programme capable de fournir un rendu graphique de la solution calculée.

Consignes

- le programme doit pouvoir lire en entrée un fichier représentant un graphe non orienté, avec un format spécifique;
- l'utilisateur doit avoir le choix de l'algorithme à exécuter (exact ou heuristique);
- le graphe doit être représenté graphiquement avec les sommets du stable maximum en rouge, les autres en noir;
- le programme doit retourner le cardinal du stable trouvé, et sauvegarder sa composition dans un fichier de sortie spécifié par l'utilisateur.

UN PROBLÈME NP-COMPLET

Le sujet qui nous intéresse est étudié à la fois en mathématiques dans la *théorie des graphes*, et en informatique dans la *théorie de la complexité*. C'est un problème facile à énoncer et à comprendre, pour lequel il n'existe pourtant pas de procédé permettant une résolution *rapide* produisant une solution *exacte*.

Définitions

Un graphe *non orienté* est un ensemble d'objets appelés **sommets**, connectés entre eux par des **arêtes**. Deux sommets liés par une arête forment un sous-ensemble non-ordonné, c'est à dire que les *paires* {v,w} et {w,v} sont équivalentes.

Un **stable** est un sous-ensemble de sommets d'un graphe qui ne possède aucune arête. Il est dit ***maximal*** s'il ne fait pas partie d'un stable de plus grande taille; il est dit ***maximum*** s'il n'existe aucun autre stable de plus grande taille dans le graphe. La confusion est aisée, et il semblerait qu'elle se soit glissée dans le sujet qui nous a été donné, qui parle de stables *maximaux* et non *maximums* !

Note: il peut y avoir plusieurs stables maximums, donc de même cardinalité, dans un graphe donné ! Notre objectif est de trouver n'importe quel stable, du moment que sa cardinalité est maximum.

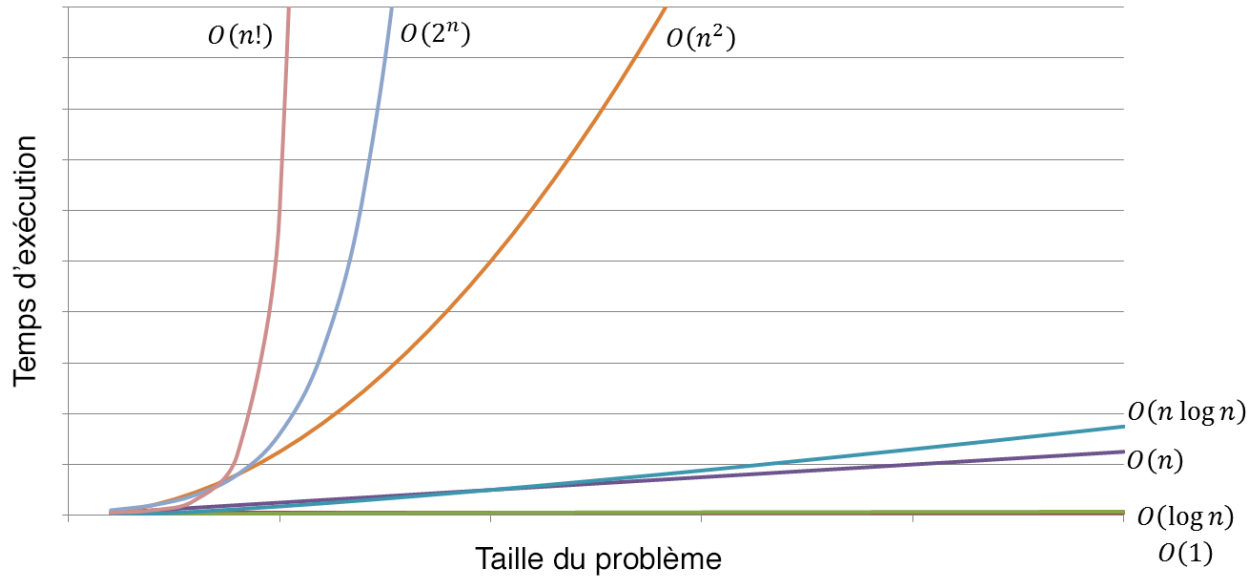
Complexité

Le problème du stable maximum est l'un des *21 problèmes NP-complets* énoncés en 1972 par le chercheur américain **Richard Karp**, professeur d'informatique et de mathématiques à Berkeley, Université de Californie.

Pour faire simple (et parce que je me sens très incompétent sur le sujet), il s'agit d'un ensemble de problèmes pour lesquels il existe des algorithmes capables de produire une solution exacte, mais dont la *complexité* (au sens informatique) est dite *exponentielle*.

Concrètement, dans le cadre d'un graphe non-orienté et de la recherche d'un stable maximum, cela signifie qu'à chaque fois qu'on ajoute un sommet à un graphe (sa taille / son cardinal augmente de 1), le temps d'exécution d'un algorithme *naïf* sera multiplié par 2. Un algorithme plus intelligent sera en mesure de calculer une solution un peu plus vite, mais sans parvenir à une complexité *polynomiale*, caractérisée par un temps d'exécution qui croît de manière beaucoup plus raisonnable en fonction de la taille du problème.

C'est donc l'intérêt des *heuristiques* : des algorithmes de complexité polynomiale qui calculent une solution approchée, correcte mais pas forcément *optimale*.



$O(2^n)$ est l'expression d'une complexité exponentielle. $O(n^2)$ et les autres graphes sur la droite sont l'expression d'une complexité polynomiale, qui montrent bien que le temps d'exécution croît beaucoup moins vite en fonction de la taille du problème.

LES ALGORITHMES

J'ai retenu pour mon programme 3 heuristiques et un algorithme exact. La 1ère heuristique est la plus simple, mais aussi, on le verra, une des plus efficaces. Les deux autres sont VSA (Vertex Support Algorithm) et MVSA (Modified Vertex Support Algorithm). Pour ce qui est de l'algorithme exact, mon implémentation reste relativement simple et compte beaucoup moins de cas particuliers qu'on ne peut en trouver dans la recherche dans ce domaine depuis les années 70.

Pour les algorithmes décrits dans ce rapport, on considère un graphe G(V,E) constitué d'un ensemble de sommets V connectés par un ensemble d'arêtes E. On utilisera la procédure retirerSommet(G,v) suivante:

```
Problème : retirer un sommet d'un graphe et éliminer ses arêtes
Entrée : un graphe G(V,E) de sommets V et d'arêtes E, un sommet v
Sortie : le graphe G(V,E) moins le sommet v et ses arêtes
Procédure retirerSommet(G, v)
      V ensemble des sommets de G
      E ensemble des arêtes de G
Début
      V ← V - {v}
      E ← E - Ev
Fin Procédure
```

…ainsi que la procédure retirerSommets(G, U) permettant de traiter plusieurs sommets à la fois:

```
Problème : retirer des sommets d'un graphe et éliminer leurs arêtes
Entrée : un graphe G(V,E) de sommets V et d'arêtes E, un ensemble de sommets U
Sortie : le graphe G(V,E) moins les sommets de U et leurs arêtes
Procédure retirerSommets(G, U)
      i entier
Début
      Pour i de 1 à |U| faire
            retirerSommet(G, Ui)
      Fin Pour
Fin Procédure
```

On admettra que ces procédures "mettent à jour" les cardinaux |V| et |E| en conséquence.

Heuristiques

Le principe des heuristiques choisies pour la recherche de stable maximum est toujours le même : on attribue un score aux sommets du graphe, on sélectionne un sommet en fonction de son score, on l'élimine, et on recommence jusqu'à ce qu'il ne reste plus aucune arête dans le graphe. Le score étant modifié à chaque itération, il faut le recalculer à chaque fois. L'ensemble des sommets restants forme alors un stable.

Max Degree (MDG)

C'est l'heuristique la plus simple du lot, et je ne saurais lui attribuer un auteur. Le score attribué aux sommets est le *degré*, qui est le nombre de sommets voisins auxquels il est connecté par une arête. Par exemple, un sommet connecté à 3 autres sommets a pour degré 3.

Dans cet algorithme, le graphe initial G est donné en entrée, et à la fin du traitement sur ce graphe, il devient le stable recherché, dont le cardinal |G| sera inférieur ou égal à son cardinal de départ :

```
Problème : recherche d'un stable dans un graphe donné
Entrée : un graphe G(V,E) de sommets V et d'arêtes E
Sortie : le graphe G(V,E) réduit, |E| = 0, c'est un stable
Procédure MDG(G)
      V ensemble des sommets de G
      E ensemble des arêtes de G
      v un sommet quelconque
Début
      Tant Que |E| > 0 faire
            v ← degréMax(V)
            retirerSommet(G, v)
      Fin Tant Que
Fin Procédure
```

Vertex Support Algorithm (VSA)

Cet algorithme a été publié en janvier 2010 (*A Simple Algorithm to Optimize Maximum Independent Set*) par S. Balaji, K. Kannan et Swaminathan Venkatasubramanian, des chercheurs indiens. Le score attribué à un sommet est ici le ***support***, c'est à dire la somme des degrés de ses voisins directs. L'algorithme est

quasiment le même que le précédent, à ceci près que l'on sélectionne le sommet dont la valeur de support est la plus grande:

```
Problème : recherche d'un stable dans un graphe donné
Entrée : un graphe G(V,E) de sommets V et d'arêtes E
Sortie : le graphe G(V,E) réduit, |E| = 0, c'est un stable
Procédure VSA(G)
      V ensemble des sommets de G
      E ensemble des arêtes de G
      v un sommet quelconque
Début
      Tant Que |E| > 0 faire
            v ← supportMax(V)
            retirerSommet(G, v)
      Fin Tant Que
Fin Procédure
```

Modified Vertex Support Algorithm (MVSA)

Cet algorithme a été publié en novembre 2013 (*Modified Vertex Support Algorithm: A New Approach for Approximation of Minimum Vertex Cover*) par Khan Imran et Khan Hasham, des chercheurs pakistanais. Le score attribué à un sommet est ici également le support, sauf que l'on cherche un sommet de **support minimum** et non maximum, et la sélection se fait en deux temps:

1. on sélectionne les voisins (W) des sommets de support minimum (U);

2. on sélectionne parmi W le sommet de support minimum.

```
Problème : recherche d'un stable dans un graphe donné
Entrée : un graphe G(V,E) de sommets V et d'arêtes E
Sortie : le graphe G(V,E) réduit, |E| = 0, c'est un stable
Procédure MVSA(G)
      V ensemble des sommets de G
      E ensemble des arêtes de G
      U, W ensembles de sommets quelconques
      v un sommet quelconque
Début
      Tant Que |E| > 0 faire
            U ← ensembleSupportMin(V)
            W ← voisinage(U)
            v ← supportMin(W)
            retirerSommet(G, v)
      Fin Tant Que
Fin Procédure
```

Comparaison

Voici une comparaison de la taille moyenne du cardinal trouvé sur 1000 graphes de 100 sommets générés aléatoirement avec un pourcentage d'arêtes variable :

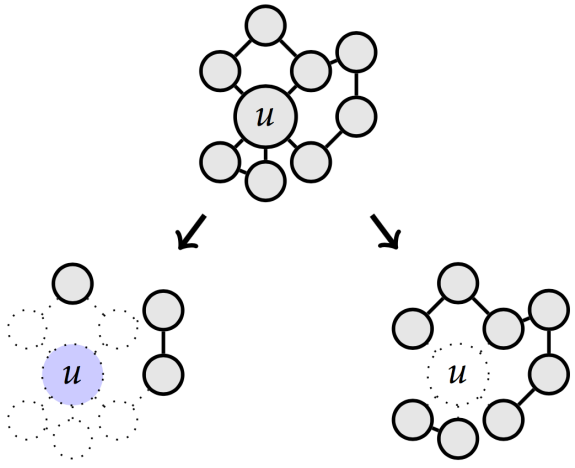
	MDG	VSA	MVSA	Exact
10% d'arêtes	27,19	23,04	27,3	30,56
20% d'arêtes	16,76	14,1	17,18	19,81

	MDG	VSA	MVSA	Exact
30% d'arêtes	12,11	9,92	12,54	14,67
40% d'arêtes	9,27	7,93	9,81	11,32
50% d'arêtes	7,63	6,35	7,95	9,24
60% d'arêtes	6,13	5,28	6,5	7,48
70% d'arêtes	5,05	4,29	5,24	6,16
80% d'arêtes	4,14	3,52	4,23	5,07
90% d'arêtes	3,25	2,66	3,31	4

MDG, de part sa simplicité, est très rapide à l'exécution. Il est aussi étonnamment performant. MDG est systématiquement 2ème en terme de cardinalité (la taille du stable trouvé). Le 1er est MVSA dans tous les cas, et VSA est toujours à la dernière place.

Algorithme exact

J'ai utilisé pour l'algorithme exact un principe de base bien connu: quelque soit le sommet considéré, il existe 2 possibilités: soit il fait partie d'un stable maximum, soit il n'en fait pas partie. Il convient alors d'explorer les 2 branches d'un arbre binaire de possibilités, de manière récursive, puis de comparer les cardinaux des 2 stables trouvés et de conserver le stable le plus grand.



En l'état, c'est un algorithme naïf dont la complexité est la plus fortement exponentielle. Il existe un grand nombre de techniques de décisions sur le choix des sommets (*branching*) qui permettent d'accélérer cet algorithme, mais elles sont trop nombreuses et trop complexes pour ce projet. Je ne suis même pas sûr qu'il soit possible de les implémenter toutes sans que cela fasse baisser les performances de façon contre-productive.

J'ai cependant conservé 3 optimisations sur des sommets "évidents" (cf. *Finding a Maximum Independent Set* (1976), par Robert Endre Tarjan et Anthony E. Trojanowski):

- un sommet de degré 1 fait forcément partie d'un stable maximum;
- un sommet de degré 2, dont les deux voisins sont également connectés entre eux (ils forment un cycle), fait forcément partie d'un stable maximum;
- si l'on peut trouver deux sommets de degré 2 qui ont les mêmes 2 voisins, ils font forcément partie d'un stable maximum.

J'ai également appliqué une dernière optimisation, partant du constat que les heuristiques MDG et MVSA sont très performantes en terme de cardinalité et en terme de rapidité. Avant d'exécuter l'algorithme exact, j'exécute MDG et MVSA afin de trouver un stable de grande taille très rapidement; puis, l'algorithme exact fait son travail mais ignore systématiquement tout sous-graphe dont le cardinal est inférieur ou égal au stable trouvé par les heuristiques. Dès qu'une branche trouve un stable plus grand, c'est ce nouveau stable qui devient la référence et c'est son cardinal qui est utilisé pour la comparaison.

Pour la description de cet algorithme, on peut reprendre les conventions employées pour les heuristiques. Soit G(V,E) un graphe donné et S un stable quelconque.

```
Problème : recherche d'un stable maximum dans un graphe donné
Entrée : un graphe G(V,E) de sommets V et d'arêtes E
Sortie : un stable maximum S
Fonction analyserGraphe(G)
      G1, G2 des copies du graphe G pour traitement
Début
      G1 ← G, G2 ← G
      MDG(G1), MVSA(G2)
      S ← max(G1, G2)
      S ← analyserSousGraphe(G, S)
      Retourner S
Fin Fonction

Problème : recherche d'un stable maximum dans un sous-graphe donné
Entrée : un sous-graphe G(V,E) de sommets V et d'arêtes E, et le plus grand
stable trouvé S
Sortie : un stable S
Fonction analyserSousGraphe(G, S)
      G1, G2 des copies de G pour traitement
      V, V1, V2 ensembles des sommets de G, G1, G2
      E ensemble des arêtes de G
      U un ensemble de sommets quelconques
      v un sommet quelconque
      S1, S2 des stables candidats
Début
      Si |E| == 0 alors
            retourner max(G, S)
      Fin Si

      U ← selectionSommetsEvidents(V)
      Si U != Ø alors
            retirerSommets(G, voisinage(U))
            retourner analyserSousGraphe(G, S)
      Fin Si

      G1 ← G, G2 ← G
      v ← degréMax(V)
      retirerSommet(G1, v)
      retirerSommets(G2, voisinage(v))
```

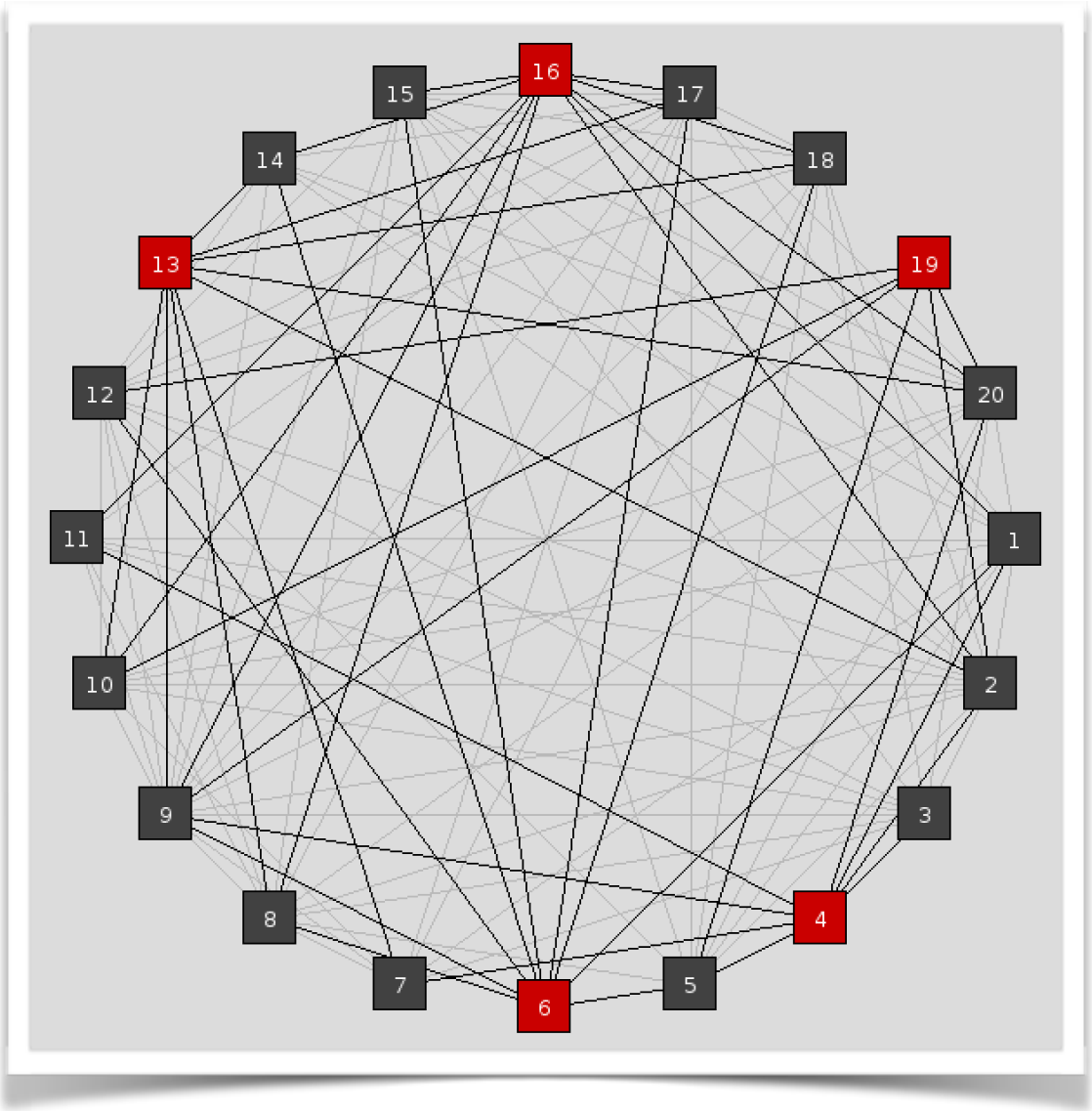
```
        Si |V₁| > |S| alors
              S₁ ← analyserSousGraphe(G₁, S)
        Fin Si

        Si |V₂| > max(|S|, |S₁|) alors
              S₂ ← analyserSousGraphe(G₂, max(S, S₁))
        Fin Si

        Retourner max(S, S₁, S₂)
Fin Fonction
```

IMPLÉMENTATION

Le programme joint à ce rapport a été réalisé en C++ (standard de 2011) sous Xcode, avec une représentation graphique utilisant la librairie SDL2. Je l'ai testé sous macOS 10.12.5 et sous Debian Jessie. Il est à noter que les fichiers DIMACS sont supportés à la lecture.



Optimisation

Il s'agit de la 5ème version depuis le début de ce projet. Une grande partie du temps a été employée à améliorer les performances, avec succès. Le profiler d'Xcode (*Instruments*) a identifié la cause majeure de lenteur dans l'algorithme exact, la copie de graphe, dont j'ai limité le nombre d'occurences à son strict minimum. Enfin, l'usage du paramètre d'optimisation du compilateur (-O3) s'est montré très avantageux.

Parallélisme

Du *multi-threading* rudimentaire a été employé pour exécuter en même temps les 3 heuristiques afin de conserver le plus grand stable ainsi trouvé. Sur une machine multi-cœurs banale, le temps d'exécution est pratiquement celui de l'algorithme le plus lent (en l'occurence, MVSA). Ce parallélisme élémentaire est aisé et l'avantage est évident.

J'ai également tenté d'employer une *pool* de 4 threads pour accélérer MVSA, en partageant le calcul à chaque itération des valeurs de support des sommets. Les 4 threads étaient lancé une seule fois, au début de l'algorithme, et commandés via des mutex (un par thread) pour démarrer et stopper leur calcul à volonté.

Malheureusement, je n'ai réussi, au mieux, qu'à égaler la performance de la version single-thread (qui est celle que j'ai fini par retenir pour ce projet), même sur de très gros graphes. Je pense que les calculs à effectuer ici sont trop légers et qu'il est difficile de faire mieux que les optimisations automatiques du compilateur.

Quant à l'algorithme exact, il est par nature très difficilement parallélisable. On se rend compte en le déroulant, qu'une des deux branches (celle qui élimine un sommet) est beaucoup plus longue que l'autre (celle qui élimine les voisins du sommet). Je n'ai pas trouvé de moyen d'optimiser cet algorithme via le parallélisme.

NVIDIA CUDA

Un algorithme exact encore plus naïf que celui retenu, consiste à tester tous les sous-ensembles possibles d'un graphe pour voir s'il s'agit d'un stable ou non. Pour un graphe de taille n, le nombre de sous-ensembles est de 2^n, ce qui est énorme et témoigne explicitement de l'exponentialité du problème ! Le seul avantage de cette méthode est qu'elle est très facilement parallélisable.

Dans l'une des premières version de mon programme, j'ai observé que je pouvais traiter un graphe d'environ 32 sommets de cette manière, avec un seul thread, dans un temps raisonnable. En lançant plusieurs threads en même temps, on pourrait réduire le temps d'exécution de cette méthode; ou plutôt, cela permettrait d'augmenter la taille des graphes que l'on saurait tester en un temps raisonnable.

Sachant que la toute dernière carte graphique de NVIDIA est capable de faire tourner 262144 threads en même temps, c'est à dire 2^{18}, un PC doté d'une seule de ces

cartes permettrait de rajouter… à peine 18 sommets à la taille maximale de graphes traitables:

$$2^{32} \times 2^{18} = 2^{50} \qquad \text{d'où:} \qquad n = 50 \text{ sommets}$$

Enfin, rajouter une autre de ces (très onéreuses) cartes dans un système n'aurait pour effet que de rajouter 1 à la puissance de 2, c'est à dire un seul sommet de plus à la taille maximale traitable:

$$2^{50} \times 2 = 2^{51} \qquad \text{d'où:} \qquad n = 51 \text{ sommets}$$

Vu que l'algorithme exact retenu (single-thread) est capable de traiter des graphes bien plus grands (de 100 à 400 sommets selon leur configuration en nombre d'arêtes), j'ai rapidement abandonné cette idée.

Conclusion

C'est le projet de programmation le plus formateur que j'aie eu depuis le début de ma Licence. C'était pour moi le meilleur prétexte pour apprendre le C++ (malgré le fait qu'un autre projet, le Tetris, était dédié à cet apprentissage) et pour commencer à réaliser un interface graphique (avec la SDL2, librairie requise pour le Tetris, ce qui a dicté mon choix).

J'ai également beaucoup apprécié cette ambiance collaborative qui s'est développée entre camarades de promotion, dont tous les acteurs ont tiré profit. La qualité de nos projets respectifs a été tirée vers le haut, et je me rends compte que cet esprit de partage des connaissances est un privilège dont les élèves de filières différentes ne jouissent pas forcément (loin de là).

Enfin, ce projet a été une excuse pour aborder les problèmes de parallélisation d'algorithmes, et c'est certainement un domaine que j'aurai à cœur d'explorer en détail plus tard dans ma formation, peut-être en Master.

Ce rapport a été remis par e-mail le 28 mai 2017 au format PDF, à :

Michel Ventou <ventou@univ-perp.fr>

Patrick Vilamajo <vilamajo@univ-perp.fr>

Christophe Negre <christophe.negre@univ-perp.fr>