Coca : Réduction SAT

FAYSAL AHMED Khaled, GALLARDO Arnaud

25 novembre 2016

https://github.com/ArnaudGallardo/CoCa

Table des matières

1 Problématique

Le problème qu'on essaie de résoudre tout au long de ce travail est celui de la coloration des graphes. On souhaite savoir s'il est possible d'attribuer une couleur parmi k,à chacun des sommets du graphe de telle sorte que les extremités de chaque arète aient des couleurs différentes. Pour celà, nous allons utiliser GLUCOSE, un puissant solveur, pour résoudre des formules SAT que nous aurons créées à partir de graphes passés en entrée.
Pour nous aider dans cette tâche, nous utiliserons une interface contenant trois fonctions liées aux graphes :
int orderG()// renvoyant le nombre de sommets.
int sizeG()// renvoyant le nombre d'arêtes
int are_adjacent // renvoyant 1 si deux sommets sont adjacents, 0 sinon.
Les formules SAT créées respectent le format DIMACS.

2 La réduction SAT

Pour obtenir les formules SAT, nous avons énoncé trois prédicats : Tout d'abord nous devions nous assurer que tous les sommets étaient colorés. Pour se faire, nous parcourons tous le graphe en vérifiant le respect de cette règle. Durant ce parcours, nous écrivons dans le fichier GtoD (pour "graph to DIMACS") la formule SAT. C'est ce fichier que nous allons envoyer au solveur Glucose.

```
for(int i = 1; i<= order*k; i+=k){
  for(int j = 0; j< k; j++){
    fprintf(fp, "%d ", i+j);
  }
  cpt++;
  fprintf(fp, "0\n");
}
```

FIGURE 1 – Première boucle de création des clauses

La variable *cpt* nous sert de compteur afin de vérifier le nombre de sommets

parcourus.

Le premier for parcourt de 1 aux nombre de sommet*nombre de couleur car chaque sommet peut avoir k couleurs.

Ensuite nous devions nous assurer qu'un sommet est associé à une seule couleur (pas plus).

```
for(int i=0; i < order; i++){
  for(int j = 1; j <= k; j++){
    for(int l = j+1; l <= k; l++){
      fprintf(fp, "-%d -%d 0 \n", k*i+j, k*i+l);
      cpt++;
    }
  }
}
```

FIGURE 2 – Deuxième boucle de création des clauses

Dans cette boucle on compare les couleurs d'un sommet deux à deux, en évitant les doublons. Par exemple, soit A un sommet, et k = 3 : on vérifie bien ((non A1)ou(non A2)) et ((non A1)ou(non A3)) et ((non A2)ou(non A3))
Enfin, nous avons du nous assurer que les sommets voisins, n'étaient pas colorés de manière identique. D'où notre dernière boucle.

```
for (int i=0; i< order; i++){
  for(int j= i+1; j<order; j++){
    if(are_adjacent(i,j)){
      for(int l =1; l<=k; l++){
        fprintf(fp, "-%d -%d 0 \n", k*i+l, k*j+l);
        cpt++;
      }
    }
  }
}
```

FIGURE 3 – Dernière boucle de création des clauses

Ici nous parcourons chaque sommet i , nous vérifions si i et i+1 sont voisins. Si oui, alors on souhaite s'assurer qu'ils n'ont pas la même couleur (représentée ci-dessus par l'indice "l").

3 Les tests

Parmi le jeu de tests qui nous a été proposé par Mr DORBEC sur sa page, seules deux tests sont réellement sortis du lot. Effectivement, les graphes qui nous ont été donnés, accompagnés de leur k-colorabilité, ont été les plus rapides à vérifier car nous connaissions les réponses. Cependant pour **m47** et **m95** ce fut plus long. En effet, pour le premier nous avons commencé à voir s'il était 10 coloriables, puis 9, puis 8... jusqu'à 6. Quand nous avons testé k = 5, nous avons attendu environ **10 secondes** pour avoir une réponse négative. En effet m47 n'est pas 5-coloriable.

Pour m95, nous avons procédé de la même manière dans un premier temps : mais une fois arrivé à k = 6, malgré un temps de calcul de plus de 3 jours (voir résultat pls bas), impossible de trouver une solution. Ainsi, nous sommes surs que ce graphe est 7-coloriable, et qu'il n'est pas 5-coloriable.

```
c
c |-------------------------------------------------------------------------------------------------|
c | id | starts | decisions  |  confls    |  Init T  |  learnts | exported | imported | promoted |   %   |
c |-------------------------------------------------------------------------------------------------|
c |  0 | 4048698 |  406471735 |  245520525 |  2387859 |  2455634 | 13623554 | 41539693 |  4737613 |  0.541 |
c |  1 | 3994746 |  396173844 |  242664836 |  2172324 |  1402214 | 14086588 | 41076662 |  5787365 |  0.541 |
c |  2 | 4040340 |  392654332 |  244433102 |  2100757 |   769366 | 14780680 | 40382568 |  5944224 |  0.539 |
c |  3 | 3754389 |  388264551 |  231454133 |  2398515 |  1601307 | 12672428 | 42490820 |  4324491 |  0.527 |
c
c synthesis   964072596 conflicts 35995873827 propagations      716 conflicts/sec    26730 propagations/sec
c Total Memory so far : 3531.45Mb
^C
*** INTERRUPTED ***
c
c
c
c |---------------------------------------- FINAL STATS ----------------------------------------------|
c
c |---------------|------------|------------|------------|------------|----------------|
c | Threads       |          0 |          1 |          2 |          3 |      Total     |
c |---------------|------------|------------|------------|------------|----------------|
c | Conflicts     |  245520832 |  242665335 |  244433261 |  231454436 |       964073864 |
c | Exported      |   13623555 |   14086593 |   14780692 |   12672429 |        55163269 |
c | Imported      |   41539714 |   41076674 |   40382573 |   42490839 |       165489800 |
c | Good          |         78 |         76 |         66 |         83 |             303 |
c | Purge         |   41539636 |   41076598 |   40382507 |   42490756 |       165489497 |
c | Promoted      |    4737615 |    5787366 |    5944232 |    4324497 |        20793710 |
c | Remove imp    |   38315459 |   36759814 |   32260890 |   39382896 |       146719059 |
c | Blocked Reuse |          0 |          0 |          0 |          0 |               0 |
c | Orig seen     |    2387859 |    2172324 |    2100757 |    2398515 |                 |
c | Unaries       |          1 |          1 |          1 |          0 |                 |
c | Binaries      |         23 |         25 |         35 |         18 |                 |
c | Glues         |         29 |         26 |         37 |         24 |                 |
c |---------------|------------|------------|------------|------------|----------------|

*** INTERRUPTED ***
```

FIGURE 4 – Interruption après plusieurs jours d'exécution

4 Affichage

Pour pousser le projet un peu plus loin, nous avons décidé de mettre en place un affichage "dynamique" de notre coloriage.

La méthode utilisée est la génération automatique d'un fichier JSON représentant le graphe et son coloriage, fichier qui est ensuite transmis à un code Javascript l'affichant sur notre page web.

La bibliothèque Javascript utilisée pour l'affichage est *arbor.js*. Elle utilise des *web workers*, le fichier javascript ne peut donc pas être envoyé par email, c'est pourquoi nous avons mis en place un dépôt Git.

Si vous observez avec attention le fichier **start.sh** vous remarquerez l'utilisation de la ligne *python -m SimpleHTTPServer 3001* permettant la création d'un serveur web local autorisant le code javascript à lire notre fichier de données (autrement bloqué en lecture par les navigateurs).

Après avoir répondu **y** à la question *Display a visual representation of the graph ? (y/n)*, le graphe est visible à l'adresse suivante : http://localhost:3001

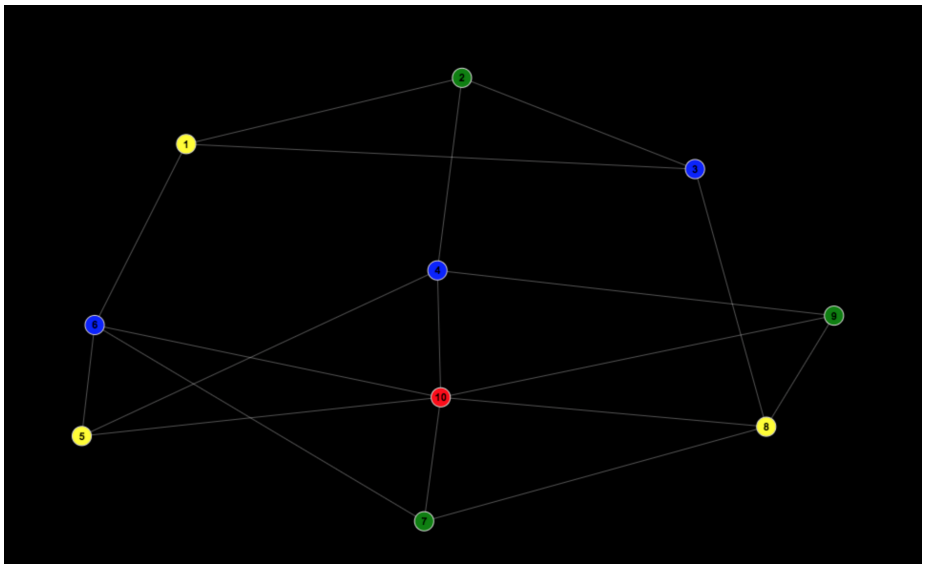

FIGURE 5 – Graphe de Golomb, 4-coloriable

Cependant, lorsque le nombre de sommets est trop important, le graphe en devient illisible.

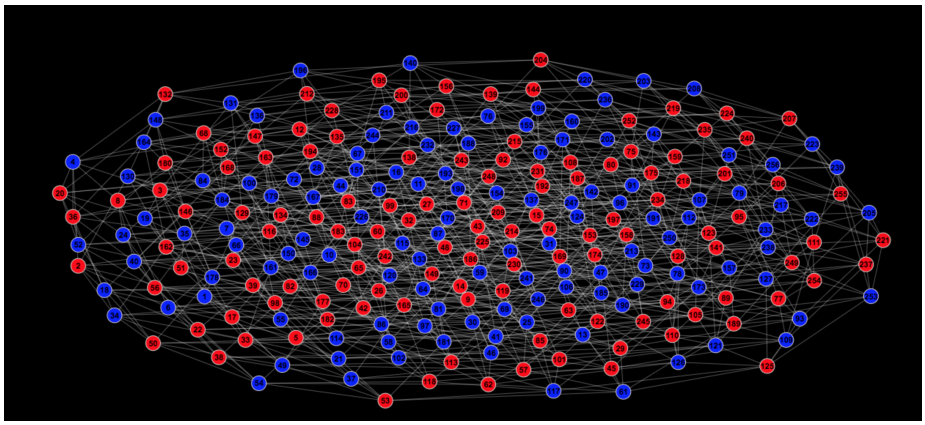

FIGURE 6 – Hypercube de dimension 8, 256 sommets, 8 régulier, biparti, 2-coloriable

5 Bibliographie

-

La page de Glucose
Le cours de CoCa de Mme Anca Muscholl
Le format DIMACS
La bibliothèque javascript "Arborjs"