Le solveur Minisat

Auteurs : PELLETIER Sébastien
Boudermine Antoine

Au cour de ce TP nous allons expérimenter le solveur minisat au travers de deux problèmes de décisions : Le principe des tiroirs et la trois coloriabilité d'un graphe.
Pour ce faire, les différents problèmes seront réduit à SAT et rentré au format DIMACS dans le solveur minisat.

Initiation avec Minisat

Le début de ce premier exercice permet de s'initier avec le format DIMACS et les formes normales conjonctives.

1.a,b
Format DIMACS : p cnf 4 3
1 2 -3 4 0
-2 3 0
-1 -4 0
Cette formule est satisfaisable.

1.c.i
Mise en forme normale conjonctive de :
$\varphi = (\neg t \rightarrow \neg s) \rightarrow (((b \cup t) \rightarrow s) \cap ((r \cap m) \rightarrow (b \cup a) \cap \neg r)$

$(\neg t \rightarrow \neg s) \equiv t \cup \neg s$,
$(b \cup t) \rightarrow s \equiv \neg(b \cup t) \cup s$
$(r \cap m) \rightarrow (b \cup a) \equiv \neg(r \cap m) \cup (b \cup a) \equiv (\neg r \cup \neg m) \cup b \cup a$

d'où :
$\varphi \equiv t \cup \neg s \rightarrow ((\neg(b \cup t) \cup s) \cap ((\neg r \cup \neg m) \cup b \cup a) \cap \neg r))$
$\varphi \equiv \neg(t \cup \neg s) \cup ((\neg(b \cup t) \cup s) \cap ((\neg r \cup \neg m) \cup b \cup a) \cap \neg r))$
$\varphi \equiv (\neg t \cap s) \cup ((\neg(b \cup t) \cup s) \cap ((\neg r \cup \neg m) \cup b \cup a) \cap \neg r))$
$\varphi \equiv (\neg t \cap s) \cup ((\neg b \cup s) \cap (\neg b \cup t) \cap (\neg r \cup \neg m \cup b \cup a) \cap \neg r))$
$\varphi \equiv (\neg t \cup s) \cap (\neg b \cup s) \cap (\neg t \cup \neg r) \cap (\neg r \cup s)$

1.c.iii

Cette formule est satisfaisable par :

b = 0, s = 0 ,t = 0, r = 0

b = 0, s = 1, t = 0, r = 0

1.d

Pour décider si une formule est une tautologie, il est possible d'utiliser cette procédure :

Algorithme Check_tautologie
ENTREE
phi une formule propositionnelle.
DEBUT

pour chaque combinaison possible de n, n des variables booléennes
Si phi(n) == Faux alors retourner Faux
fin pour
retourner Vrai

FIN

La complexité de cette procédure est en $O(2^n)$ avec n le nombre de variable car il faut tester toutes les combinaisons possibles.

Le principe des tiroirs

Le principe des tiroirs dit qu'il est impossible de ranger n+1 chaussettes dans n tiroirs en imposant qu'il y ai une seule chaussette par tiroirs.

Ce principe peux ce réduire au problème SAT via cette formule propositionnel :

$$\bigcap_{1\leqslant i\leqslant n+1}\left[\bigcup_{1\leqslant j\leqslant n}(Ci,j)\wedge\bigcap_{i\leqslant j'\leqslant n|j'\neq j\ 1\leqslant j\leqslant n}(\overline{Ci,j}\vee\overline{Ci,j'})\right]\wedge\bigcap_{1\leqslant j\leqslant n\ 1\leqslant i\leqslant n+1\ 1\leqslant i'\leqslant n+1\ i'\neq i}(\overline{Ci,j}\vee\overline{Ci',j})$$

Ici on vérifie que : - Chaque chaussette est rangé dans au moins un tiroir.
- Chaque chaussette est rangé dans un seul tiroir.
- Un tiroir contient une seule chaussette.

Ci,j = 1 SSI la chaussette i est rangé dans le tiroir j.

Nous avons créé un programme en c++ qui avec un nombre de tiroirs donné en paramètre génère un fichier dimacs correspondant à cette formule
(./gen_chausette.bash nb_tiroir pour l'utilisation)

Cette algorithme est utilisable pour des valeurs inférieur à 10, à partir de cette valeur, le temps d'exécution devient irraisonnable (1 minutes pour 10) sur les machines des salles de TP.

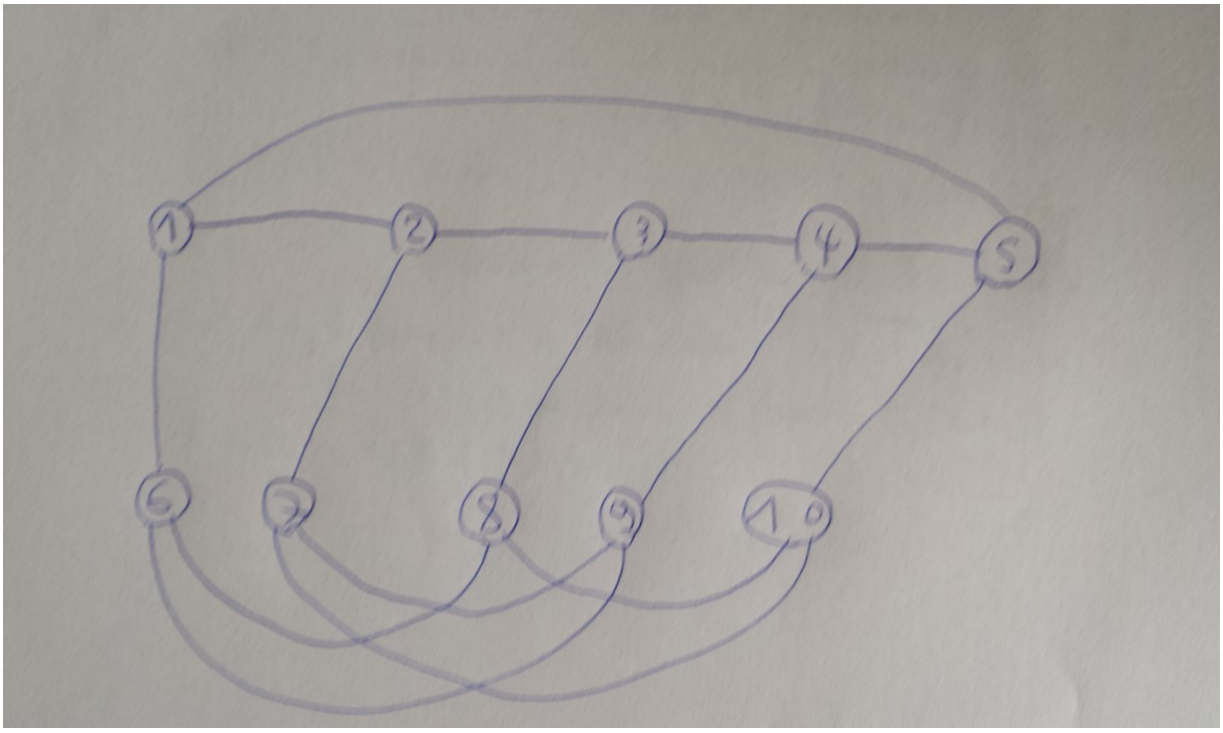
Trois coloriabilité d'un graphe

Nous allons maintenant intéresser au problème de trois coloriabilité d'un graphe. Pour ce faire nous allons représenter nos graphe ce cette manière :

3 3 1 2 2 3 3 1

Les deux premier nombre signifient respectivement le nombre de sommets puis le nombre d'arêtes. Chaque couple de nombres suivant représentent une arête.

Exemple : 10 15 1 2 3 3 4 4 5 5 1 1 6 2 7 3 8 4 9 5 10 6 8 7 9 8 10 9 6 10 7



Nous allons proposer une réduction polynomial de ce problème à SAT selon cette formule :

$$A = \bigcap_{n}^{i=1} \left(\left(C_{i,1} \wedge \overline{C_{i,2}} \wedge \overline{C_{i,3}} \right) \vee \left(\overline{C_{i,1}} \wedge C_{i,2} \wedge \overline{C_{i,3}} \right) \vee \left(\overline{C_{i,1}} \wedge \overline{C_{i,2}} \wedge C_{i,3} \right) \right)$$

$$A \equiv \bigcap_{n}^{i=1} \left(\left(\overline{C_{i,2}} \vee \overline{C_{i,1}} \right) \wedge \left(\overline{C_{i,3}} \vee \overline{C_{i,1}} \right) \wedge \left(\overline{C_{i,3}} \vee \overline{C_{i,2}} \right) \wedge \left(C_{1,2} \vee C_{i,2} \vee C_{i,3} \right) \right)$$

$$\varphi = A \wedge \bigcap_{(i,i' \in A)} \left(\left(\overline{C_{i,1}} \vee \overline{C_{i',1}} \right) \wedge \left(\overline{C_{i,2}} \vee \overline{C_{i',2}} \right) \wedge \left(\overline{C_{i,3}} \vee \overline{C_{i',3}} \right) \right)$$

Ci,1 = 1 SSI le sommet i est de couleur 1, il y a trois couleur 1, 2 et 3.
Ici on transcrit : - Chaque sommet et colorier d'un seule couleur
- Chaque sommet relié par une arête sont de couleur différentes.

Le programme gen_fnc_graph.exe génère la formule propositionnelle du graphe donné en paramètre (ex : ./gen_fnc_graph.exe 3 3 1 2 2 3 3 1).

Le scripte test_graph.bash teste un graphe et donne sont coloriage si il est 3_col. Il possède une option -r pour préciser si on désire générer un graphe aléatoire. Ce scripte affiche le graphe sous la forme vu précédemment.

Ex : ./test_graph.bash ''3 3 1 2 2 3 3 1'' ou ./test_graph.bash -r n
(n est le nombre de sommet maximal possible du graphe généré aléatoirement)

La complexité du solveur dépend du nombre de sommet et d'arête car il s'arrête au premier modèle trouve. Moins il y a d'arête, plus les modèles sont nombreux. Le solveur n'a pas besoin de parcourir toutes les possibilités pour affirmer la satisfaisabilité d'une formule propositionnel. Dans le pire des cas, comme il y a 3*nb_sommet variable dans la formule, la complexité est O(8^(nb_sommet)).