Rapport de Projet de complexité (TP 2)

LOIGNON Lucas

CAUET Christopher

DEUTSCH Rémi

VINCENT Pierre

I. Introduction

Ceci est le rapport de projet 2 de complexité qui présente le rapport effectué par l'équipe composé de LOIGNON Lucas, CAUET Christopher, DEUTSCH Rémi et VINCENT Pierre.

Dans ce rapport nous nous efforcerons de prouver la réduction polynomiale du problème STABLE vers le problème SAT et inversement. Dans une première partie nous présenterons un court rappel sur les définitions des problèmes SAT et STABLE. Puis nous verrons l'algorithme de réduction polynomiale du problème SAT vers le problème STABLE ensuite nous présenterons l'algorithme de réduction polynomiale du problème STABLE vers le problème SAT et enfin dans une troisième partie nous étudierons le comportement de ces algorithmes de façon expérimentale sur divers jeux de données pertinents.

II. Généralités

Le problème SAT est défini comme étant :

INSTANCE : une formule sous forme normale conjonctive.

QUESTION : existe-t-il un modèle à cette formule ?

Le problème STABLE est défini comme étant :

INSTANCE : un graphe non orienté G (S, A) et un entier k.

QUESTION : existe-t-il un stable de taille k ou plus dans le graphe G ?

III. Réduction polynomiale de SAT vers STABLE

Dans cette partie nous allons détailler la réduction de SAT vers STABLE.

1. Preuve

$Soit\ F\ une\ formule\ CNF, F \in V_{SAT}$

$=>\ \exists M\ un\ modèle\ de\ F.$

$=>\ \exists V\ valuation\ de\ F, tel\ que\ V(F) = Vrai.$

$Soit\ \varphi\ l'ensemble\ des\ formules\ propositionelles, Soit\ h(l)\ le\ nombre\ d'occurence\ de\ l\ dans\ F.$

$Soit\ f: L \times \varphi \rightarrow S^{h(L)}\ tel\ que\ f(l, F) = \{l1, l2, \dots, ln \mid n\ est\ le\ nombre\ d'occurence\ de\ l\ dans\ F\}.$

$Soit\ g: L \times \varphi \rightarrow A^{h(L)}\ tel\ que\ g(l, F) = \{ \cup (x, y), tel\ que\ x \in f(l, F)\ et\ y \in f(\neg l, F)\ OU\ U(x, y), tel\ que\ x\ et\ y\ appartiennent\ à\ la\ même\ clause\ \}.$

$Soit\ G(S, A)\ tel\ que\ S = f(L, F) et\ A = g(L, F)\ .$

Pour chaque clause C de F on va construire un ensemble de sommets ST tel que |C| = |ST| en nommant ces sommets de façon unique et en les reliant tous entre eux (on va créer une clique). Ensuite pour chaque couple de littéraux dans F de la forme (l,-l) c'est-à-dire un littéral et sa négation, nous allons ajouter une arête entre tous les sommets que nous avons créés pour le littéral l à l'étape précédente et tous les sommets créés pour le littéral -l.

On définit k comme étant le nombre de clause de F.

De cette manière lorsque l'on va rechercher un stable dans G de taille k nous ne pourrons pas mettre des sommets issus de l et des sommets issu de -l dans le stable (ce qui est logique car il est impossible de satisfaire l et -l en même temps).

Nous ne pourrons pas non plus mettre deux sommets issus de la même clause dans le stable (car chaque clause de F forme une clique dans le graphe G) ce qui garantit que si le graphe G possède un stable de taille k cela signifie que toutes les clauses de F peuvent être satisfaites.

Cela s'explique par le fait que dans le graphe G nous avons créé k cliques donc s'il existe un stable de taille k dans G cela signifie que pour chaque clique il y a un sommet appartenant à cette clique qui se trouve dans le stable.

Or mettre un sommet appartenant à une clique de G dans le stable revient à satisfaire la clause de F qui est à l'origine de la création de cette clique.

$$=> (G(S, A), k) \in V_{STABLE}\ .$$

2. Algorithme

L'algorithme utilisé pour passer d'une instance de problème SAT à une instance de problème STABLE est le suivant :

```
Soit f la fonction de transformation d'une variable propositionnelle en sommet.
ENTREE : une formule propositionnelle F sous forme CNF dans le format Minisat.

VARIABLES : S ensemble de sommets, A ensemble d'arêtes, littéraux tableau de littéral, i nombre.

DEBUT
        i = 0 ;
        Pour chaque clause c de F faire :
                Pour chaque variable v de c faire :
                        Ajouter f(c) à S ;
                        Littéraux[i] = c ;
                        i = i+1 ;
                        Pour chaque variable n de c faire :
                                Ajouter {f(c), f(n)} à A ;

        Pour j allant de 0 à i faire :
                Pour k allant de j à i faire :
                        Si littéraux[j] == -littéraux[k] faire :
                                Ajouter {f(j), f(k)} à A ;
FIN
```

Nous pouvons aisément voire ici que cet algorithme est de complexité polynomiale, en effet c'est un simple parcours de toutes les variables propositionnelles deux fois pour la création des cliques qui correspondent à une clause de F, et c'est un double parcours de tableau pour la création des arêtes.

Cet algorithme de transformation est donc de complexité $O(n^2)$ avec n étant le nombre de variables propositionnelles de la formule SAT. Donc la transformation d'une instance de SAT en instance de STABLE se fait en temps polynomial.

IV. Réduction polynomiale de STABLE vers SAT

Dans cette partie nous allons détailler la réduction polynomiale du problème STABLE vers le problème SAT.

1. Preuve

$(G,k) \in V_{STABLE} => (F) \in V_{SAT}, avec\ G = (S,A)\ et\ F\ une\ formule\ normale\ conjonctive.$

$(G,k) \in V_{STABLE} => (\exists\ X \subseteq S\ tel\ que\ |X| \geq k,\ \forall\ x,y \in X, (x,y) \notin A)$
$=> (\exists\ X \subseteq S\ tel\ que\ |X| \geq k, \quad \forall x \in S, \quad x \in X => \forall y\ tel\ que\ (x,y) \in A,$
$y \notin X)$
$=> (\exists\ X \subseteq S, x_1, x_2, \dots, x_k \in S, tel\ que\ \forall\ x_i, i\ \in \mathbb{N},\ 1 \leq \mathrm{i} \leq \mathrm{k},\ x_i \in X, et\ \forall\ y \in S, (x_i, y) \in A, y$
$\notin X)$
$=> (\exists\ X \subseteq S, x_1, x_2, \dots, x_k \in X, tel\ que\ \forall x, x' \in S,$
$(\exists j \in \mathbb{N},\ 1 \leq \mathrm{j} \leq \mathrm{k}, tel\ que\ x = x_j \neq x', et\ (x,x') => x' \notin X)\ OU\ (x \notin X)$

On a donc transformé les 2 propriétés sur G intrinsèques aux instances positives de STABLE, à savoir,

- Il doit exister un sous-ensemble de sommets de S, X, pour lequel tous ses sommets ne sont pas adjacents ;
- X doit contenir au moins k sommets de S ;

En autres 2 propriétés sur G :

- Chaque sommet de S est soit un des k sommets de X, qui n'est aucun autre sommet de S, soit n'est pas dans X ;
- Si un sommet est dans X, aucun de ses adjacents ne l'est.

Ces propriétés peuvent être plus facilement traduite dans un formalisme propositionnel :

On associe à chaque sommet x une variable propositionnelle px, qui indiquera s'il est ou non dans un stable solution.

- De cette façon, si y est un adjacent de x, alors (px => -py && py => -px), ce qui se peut se traduire par une clause, que l'on appellera *clause d'adjacence*,

  (-px ∨ -py).

- L'attribution des k sommets de X à des sommets de S peut se voir comme l'attribution d'une position dans le stable : ainsi, si x = x_i, x sera le i-ème sommet de X, et aucun autre sommet de S ne sera à la i-ème position dans X.
  On peut ainsi traduire cela selon la logique propositionnelle, en un ensemble de conjonction de clauses, que l'on appellera *conjonction de positions* :

Soient x0 la variable propositionnelle indiquant que x n'est pas dans X, et x1,…,xk les variables propositionnelles indiquant que x est à la i-ème position, et z n'importe quel autre sommet du graphe :

$(-x1 \vee px) \wedge (-x2 \vee px) \wedge \dots \wedge (-xk \vee px)$ //Si position dans le stable, alors //sommet dans le stable

$\wedge (-x0 \vee -x1) \wedge (-x0 \vee -x2) \wedge \dots \wedge (-x0 \vee -xk)$ //Une seule position par sommet

$\wedge (-x0 \vee -z0) \wedge (-x1 \vee -z1) \wedge \dots \wedge (-xk \vee -zk)$ //Un seul sommet par position

$\wedge (x0 \vee z0) \wedge (x1 \vee z1) \wedge \dots \wedge (xk \vee zk)$ //Toutes les positions doivent être //attribuées

En combinant ainsi ces deux types de clauses, on peut donc former une formule normale conjonctive que l'on pourra passer au problème SAT qui formalisera bien le problème STABLE, qui sera de la forme :

F = *clauses d'adjacences* ∧ *conjonction de positions*

$$=> (F)\ admet\ un\ modèle$$
$$=> (F) \in V_{SAT}$$

On doit maintenant prouver que cette transformation peut se faire en un temps polynomial sur modèle de calcul déterministe, que l'on adaptera ainsi en langage de programmation C.

Pour rappel, notre formule normale conjonctive, F, se décompose en deux parties :

F = *clauses d'adjacences* ∧ *conjonction de positions.*

- Les clauses d'adjacences sont les clauses qui représentent l'adjacence des sommets entre eux, donc les arcs du graphe. Ainsi, pour chaque arc (x,y), on sait que l'on aura une clause d'adjacence :

$$(-px \vee -py).$$

On parcourt ainsi l'ensemble du graphe afin de trouver tous les arcs, ce qui se fait en $\theta(n+m)$ ou $\theta(n^2)$ selon l'implémentation des graphes choisies, où n est le nombre de sommets et m le nombre d'arcs du graphe, et l'on crée une clause par arc, ce qui se fait donc en temps polynomial.

- La conjonction de positions représente les conditions nécessaires à l'attribution des positions aux sommets, et est donc une conjonction de toutes celles-ci :

➔ Si l'on attribue une position i à un sommet x, alors celui-ci est dans le stable. Ainsi, pour chacune des positions dans le stable i ≠ 0, et ce pour chaque sommet du graphe, on construit une clause :

$$(-xi \vee px).$$

L'attribution se faisant pour chacun des sommets et chacune des positions, il nous faudra n*k clauses, ce qui se fait donc en temps polynomial.

➔ Si l'on attribue une position i à un sommet x, alors celui-ci ne peut pas avoir d'autres positions j. Ainsi, on construit des clauses :

$$(-xi \vee -xj),$$

Pour chaque couple (i,j), i≠j, de positions, et ce pour chaque sommet x du graphe, ce qui se fera en $\theta(k^2*n)$, donc en temps polynomial.

➔ Si l'on attribue une position i à un sommet x, alors aucun autre sommet x2 ne peut avoir cette position. On construit alors des clauses :

$$(-xi \vee -x2i),$$

Pour chaque couple (x,x2), x≠x2, de sommets, et ce pour chaque position i, ce qui se fera en $\theta(n^2*k)$, donc en temps polynomial.

➔ Il faut impérativement que les positions soient attribuées à un sommet. Ainsi, on construit des clauses, pour chaque position i et l'ensemble S = {x1,x2,…,xn} de sommets de G :

$$(x1i \vee x2i \vee \ldots \vee xni).$$

Puisque l'on doit affecter k positions, on crée k clauses, ce qui se fait en temps polynomial.

La conjonction de positions étant constitués de la conjonction de ces 4 conjonctions de clauses, chacune construite en un temps polynomial, la conjonction de position est construite en temps polynomial, ici $\theta(n^2*k)$.

Puisque F est constituée de la conjonction de deux formules construites en temps polynomial, celle-ci est donc construite en temps polynomial, ici $\theta(n^2*k)$.

2. Algorithme

Afin d'effectuer la transformation, il suffit, étant donné un graphe G et un entier k en entrée, de créer l'ensemble de clauses de la formule F : on suivra donc les étapes ci-dessus.

```
StableToSat (Graphe G = (S, A), int k) {

        //Clauses d'adjacences
        Pour chaque sommet x1 ∈ S
                Pour chaque sommet x2 ∈ S
                        Si (x1, x2) ∈ A, construire (-px1 ∨ -px2)

        //Attribution des positions
Pour chaque sommet x ∈ S {
        Pour chaque position i, 0 ≤ i < k+1 {
                Si i ≠ 0, construire (-xi ∨ px).
                Pour chaque position j, 0 ≤ j < k+1
                        Si j ≠ 0, construire (-xi ∨ -xj).
                Pour chaque sommet x2 ∈ S
                        Si x ≠ x2, construire (-xi ∨ -x2i).

                }
         }

        Pour chaque position i, 0 < i < k+1
                Construire (V_{x ∈ S} xi).

}
```

La transformation ayant été prouvé, et puisque l'on respecte scrupuleusement sa logique pour construire l'ensemble de clauses, ce pseudo est valide.

Sa complexité est bien polynomiale, ici en $\theta(n^2*k)$.

V. Résultats expérimentaux

1. Evaluation du formalisme SAT

Dans cette partie nous allons tenter d'évaluer expérimentalement la puissance du formalisme SAT et surtout d'un solveur : Minisat.

Pour cette expérience nous nous donnons un jeu de données de deux instances SAT au format Minisat :

Instance 1	Instance 2
p cnf 3 9	P cnf 5 9

On peut constater ici que ces deux instances peuvent être considéré comme étant petite mais lorsque nous transformons ces instances en instance de problème STABLE voici ce qu'il se passe :

Instance 1	Instance 2

Nous pouvons dès à présent constater une différence nette entre les deux instances en regardant le nombre de sommets de chacune.

Maintenant nous allons comparer les temps d'exécution de ces deux instances avec Minisat puis avec l'algorithme de recherche d'un STABLE.

	Minisat	Stable
Instance 1	0.001 sec	0.135 sec
Instance 2	0.001 sec	Ne termine pas sur la machine de test

En effet dans le formalisme SAT la taille de la donnée est nettement plus petite que dans le formalisme Graphe. Les algorithmes de résolution des problèmes SAT et STABLE étant exponentiels, il est normal que le nombre de récursion nécessaire à la résolution d'un problème STABLE sur un graphe de 40 sommets explose.

2. Evaluation du formalisme Graphe pour le problème STABLE

Dans cette partie nous allons étudier la résolution du problème STABLE avec deux formalismes différents : Graphe et SAT.

Pour cette expérience nous nous donnons un jeu de données de trois instances du problèmes STABLE sous formalisme Graphe.

Les trois instances sont respectivement les instances myciel3, myciel4 et 1-FullIns_3 issus du benchmark de graphe fourni sur le site de Mr TERRIOUX.

Voici quelques statistiques sur ces graphes :

	Myciel3	Myciel4	1-FullIns_3
Nombre de sommets	10	23	30
Nombre d'arêtes		71	100

Voici maintenant des statistiques sur les instances SAT créé à partir de ces graphes :

	Myciel3	Myciel4	1-FullIns_3
Nombre de clauses	1000	9500	20000
Nombre de littéraux			

Nous allons maintenant comparer les temps d'exécution des deux formalismes :

	Minisat	Stable
Myciel3		
Myciel4		
1-Fullin_3		

I apparait donc ici que le formalisme SAT peut en effet produire de grandes instances mais que néanmoins la méthode de résolution implémenté avec Minisat est très performante.

En effet les solveur SAT fonctionnent souvent sur le modèle d'un algorithme DPLL qui permet l'implémentation de diverses heuristiques afin de gagner en performance.

VI. Conclusion

D'après le théorème de Cook le problème SAT appartient à la classe de problème NP-Complet ce qui veut dire que tous les problèmes appartenant à la classe de problème NP sont polynomialement réductible à SAT.

Les progrès fait dans l'implémentation et la recherche sur la résolution du problèmes SAT sont donc très important pour la résolution d'autres problème NP car à ce jour les SAT-Solveurs sont une des seules solutions efficaces pour la résolution exacte de problème NP.

Parfois Il est en effet plus efficace de réduire un problème NP à un autre problème NP-C dont on a déjà une solution efficace comme c'est le cas pour SAT.