Transformation polynomiale de STABLE en SAT :

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 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 ˅ px) ˄ (-x2 ˅ px) ˄ … ˄ (-xk ˅ px) //Si position dans le stable, alors //sommet dans le stable

˄ (-x0 ˅ -x1) ˄ (-x0 ˅ -x2) ˄ … ˄ (-x0 ˅ -xk) //Une seule position par sommet

˄ (-x0 ˅ -z0) ˄ (-x1 ˅ -z1) ˄ … ˄ (-xk ˅ -zk) //Un seul sommet par position ˄ (x0 ˅ z0) ˄ (x1 ˅ z1) ˄ … ˄ (xk ˅ 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*

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 ˅ -py).

On parcourt ainsi l’ensemble du graphe afin de trouver tous les arcs, ce qui se fait en θ(n+m) ou θ(n²) 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 attribut 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 ˅ 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 ˅ -xj),

pour chaque couple (i,j), i≠j, de positions, et ce pour chaque sommet x du graphe, ce qui se fera en θ(k²\*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 ˅ -x2i),

pour chaque couple (x,x2), x≠x2, de sommets, et ce pour chaque position i, ce qui se fera en θ(n²\*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 ˅ x2i ˅… ˅ 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 θ(n²\*k).

Puisque F est constituée de la conjonction de deux formules construites en temps polynomial, celle-ci est donc construite en temps polynomial, ici θ(n²\*k).

Pseudo-code :

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 les 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 ).

}

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 θ(n²\*k).