Université USTHB – Bab-Ezzouar Bab-Ezzouar, 11 Décembre 2017

Faculté de l’Electronique et de l’Informatique, Département de l’Informatique Année universitaire 2017/2018

1ère année Master Informatique, Semestre 1 Semestre 1

Module : Conception et Complexité des Algorithmes

-------------------------------------------------------------------------------------------------------------------------------------------------------

**Série de Travaux Pratiques n° 6**

**Algorithmes de complexité temporelle exponentielle O(an) (a>1)**

**Le problème SAT et les classes de problèmes P, NP et NP-complets**

L’objet de ce TP est l’étude expérimentale des algorithmes de vérification et de résolution du problème de satisfiabilité des formues booléennes connu sous l’abréviation ″SAT (boolean **SAT**isfiability problem)″. On s’intéresse ici à l’algorithme de résolution brut (ou naïf) . C’est un problème devenu classique et fondamental en informatique. Cela permet d’introduire les classes de problèmes P, NP et NP-complets ; et d’illustrer ce qu’est un algorithme non-déterministe polynomial.

**Le problème SAT (boolean SATisfiability problem** ou p**ropositional SATisfiability problem) :**

**Description :**

- Une proposition atomique est une variable booléenne, c’est-à-dire prenant ses valeurs dans l’ensemble BOOL={VRAI, FAUX}.

- Un littéral est une proposition atomique ou la négation d’une proposition atomique.

- Une proposition atomique est aussi appelée littéral positif ; et la négation d’une proposition atomique est appelé littéral négatif.

- Une clause est une disjonction (somme logique ou (or)) de littéraux.

Etant données m propositions atomiques p1, …, pm, une instanciation du m-uplet (p1, …,pm) est un élément de {VRAI,FAUX}m. Une instanciation (e1,…,em) de (p1,…,pm) satisfait une clause c (notée (e1, …,em) c) si et seulement si l’une des conditions suivantes est satisfaite :

1- il existe i Є {1,…,m} tel que (ei = VRAI) et (pi occurre ou existe dans c) ;

2- il existe i Є {1,…,m} tel que (ei = FAUX) et (┐pi occurre ou existe dans c).

Le problème SAT est maintenant défini comme suit :

* **Entrée :** une conjonction C de n clauses construites à l’aide de m propositions

atomiques p1, …,pm ;

* **Sortie :** la conjonction C est-elle satisfiable ?

Une instanciation satisfait une conjonction de clauses si et seulement si elle satisfait chacune de ses clauses. Une conjonction (ou produit logique) de clauses est satisfiable si et seulement si il existe une instanciation la satisfaisant. Une instanciation satisfaisant une conjonction est dite solution ou modèle de la conjonction.

1. Proposer une structure de données permettant de représenter une instance du problème SAT.
2. Développer l’algorithme de vérification ou validation, noté validation\_sat, sous forme d’une fonction booléenne dont il est important d’expliquer les paramètres.
3. Calculer le nombre d’opérations élémentaires de l’algorithme dans le pire cas.
4. Montrer que la complexité de l’algorithme de vérification est polynômiale. Quelle est la classe du problème SAT ?
5. Développer l’algorithme de résolution du problème SAT qui cherche une solution pour ce problème.
6. Calculer la complexité de cet algorithme. Quelle est la classe du problème SAT ?
7. Développer les programmes correspondants et mesurer les temps d’exécutions pour un échantillon de valeurs de n et m.

**Le Corrigé du TP n°6**

**Version 1 :** SAT comme problème de décision (oui/non)

**début**

Cond=faux

i=0

**Tant que** (non Cond) **et** (i<=2m-1) {

**Si** (i solution de C) **alors** {

Imprimer « OUI »

Cond=vrai

}

i++

}

**Si** (non Cond) **alors** imprimer « NON »

**Fin**

**Version 2 :** impression d’une solution si satisfiabilité

**début**

Cond=faux

i=0

**Tant que** (non Cond) **et** (i<=2m-1){

**Si** (i solution de C) **alors**{

Imprimer i

Cond=vrai

}

i++

}

**Si** (non Cond) **alors** imprimer « AUCUNE SOLUTION »

**fin**

**Version 3 :** impression de toutes les solutions

**début**

Cond=faux

i=0

**Tant que** (i<=2m-1){

**Si** (i solution de C) **alors**{

Imprimer i

Cond=vrai

}

i++

}

**Si** (non Cond) **alors** imprimer « AUCUNE SOLUTION »

**Fin**

**Programme C pour les versions 1 et 2**

main(){

int C[100][100],inst[100],

n,m,i,j,k,dividende,c\_satisfaite,i\_solution,max ;

printf(« saisie du nombre de clauses : ») ;

scanf(« %d\n »,&n) ;

printf(« Saisie du nombre de propositions atomiques : ») ;

scanf(« %d\n »,&m) ;

for(i=0 ;i<n ;i++){

printf(“Saisie de la clause %d : »,i) ;

for(j=0 ;j<m;j++){

printf(« C[%d][%d]= ? ») ;

scanf(« %d\ n»,&C[i][j]) ;

}

}

i=0;

cond=0 ;

max=2^m;

while( !cond && i<max){

//Mise à zéro du tableau inst//

for(j=0 ;j<m ;j++)inst[j]=0 ;

dividende=i ;

j=m-1 ;

while(dividende !=0){

inst[j]=dividende%2 ;

dividende=dividende/2 ;

j-- ;

}

i\_solution=1 ;

k=0 ;

while(i\_solution && k<n){

c\_satisfaite=0;

j=0;

while(!c\_satisfaite && j<m)

if( (inst[j]==0 && C[k][j]==0) ||

(inst[j]==1 && C[k][j]==1) )c\_satisfaite=1

else j++ ;

if( !c\_satisfaite)i\_solution=0 ;

k++ ;

}

cond=i\_solution ;

i++;

}

**//VERSION 1**

if(cond)printf(“oui”);

else printf(“non“);

**//VERSION 2**

if(cond){

printf(“L’instance du problème SAT est satisfiable par l’instanciation “);

for(j=0;j<m;j++)

if(inst[j]==0)prinf(“F”);

else printf(“V”);

printf(“qui en est donc solution”);

}

else printf(“L’instance du problème SAT n’est pas satisfiable“);

}

**Un algorithme pour la version 3 s’en déduit facilement …**

**Discussion**

Le programme, dans ses versions 1 et 2, procède par énumération des différentes instanciations possibles (e1,…,em) du m-uplet (p1,…,pm) de propositions atomiques. Quand il y a satisfiabilité, l’énumération s’arrête à la toute première rencontre d’une instanciation satisfaisant la conjonction de clauses : il y a alors sortie de la boucle la plus externe et impression du résultat. S’il n’y a pas satisfiabilité, l’énumération des 2m instanciations possibles et vérification, pour chacune d’entre elles, qu’elle ne satisfait pas toutes les clauses de la conjonction, sont faites avant de se rendre compte qu’il n’y a pas satisfiabilité. Pour une instanciation donnée, représentée par le tableau *inst*, le programme parcourt, dans le pire des cas, chacune des lignes de la matrice C pour tester si la clause qu’elle représente est satisfaite par l’instanciation : le coût d’un tel test est en O(n\*m). Le programme est donc non-déterministe polynomial, de complexité O(n\*m). Cela implique deux choses :

* si on est suffisamment chanceux pour « tomber » sur la bonne instanciation dès le départ, on répond (positivement) à la question d’existence d’une solution, et on imprime une telle solution, au bout d’un temps polynomial (en la taille de l’instance en entrée), en O(n\*m) plus précisément
* si, par contre, l’instance n’a pas de solution, ou en a une seule consistant en la toute dernière dans l’énumération, le programme entrera dans la boucle la plus externe 2m fois dont le coût de chacune est en O(n\*m) : le coût du programme sur de telles instances est exponentiel