Incidence des réductions polynomiales pour la résolution d'instances SAT difficiles

VARET Adrien - ENTAKLI Romain - D'ARRIGO Valentin

20 avril 2018

Résumé

La réduction polynomiale est un outil particulièrement important en théorie de la complexité, au regard notamment de l'analyse de problèmes en vue de déterminer la classe de complexité à laquelle ils appartiennent. Cet outil a particulièrement été étudié au niveau théorique, mais il semblerait qu'au niveau pratique, ce type d'étude ait beaucoup moins été développé. L'objet de ce projet porte sur cette seconde question. Depuis une dizaine d'années, des progrès spectaculaires ont été réalisés pour la résolution pratique du problème de la satisfiabilité (SAT), à tel point que de nombreuses instances industrielles, issues de problèmes réels donc, sont exprimées dans les termes d'instances SAT, et sont résolues très efficacement par les solveurs de l'état de l'art. Il n'en demeure pas moins vrai que certaines instances sont toujours extrêmement difficiles à résoudre. Nous focaliserons d'ailleurs cette étude sur ce type d'instances. Il s'agit ici d'étudier certaines transformations polynomiales et d'analyser leur incidence sur la résolution pratique de ces instances difficiles. La question étant de savoir si des transformations permettent de réduire la difficulté et dans quelles proportions. Nous étudierons notamment les réductions, qui, partant d'une instance SAT quelconque, permettent d'exprimer celles-ci sous forme 3-SAT, sous forme d'instances de la 3-COLORATION (de graphes), de CSP (CSP correspond aux "Constraint Satisfaction Problems" et constitue une généralisation de SAT), voire des retraductions vers le formalisme SAT.

Depuis le début de l'informatique, on cherche sans cesse à essayer d ?optimiser les algorithmes le plus possible afin que ces derniers soient le plus rapide possible. Mais à l'heure d'aujourd ?hui, on pourrait être tenté de penser que cette recherche frénétique d ?optimisation soit devenu obsolète. En effet, de nos jours, même les machines dites 'bas de gamme' sont considérées comme puissantes en comparaison avec les machines d'il y a dix ans. Cette hausse exponentielle des capacités des machines a également pour conséquence la résolution de certains problèmes qui ne pouvaient être traités auparavant ainsi que le traitement d'instances beaucoup plus grandes.

Néanmoins, cette recherche d'optimisation n'est toujours pas obsolète. En effet afin de répondre à certaines problématiques dans le monde de la recherche, ou encore à des problèmes concrets, on cherche à résoudre des problèmes dits 'difficiles' avec des instances de très grande taille qui sont très lourds à résoudre, même pour les machines actuelles.

Notre sujet de TER portera donc sur une partie de cette recherche d'optimisation et sera fortement associé à l'UE sur la théorie de la complexité que nous avons étudié lors du précédent semestre. Dans ce compte-rendu, nous effectuerons une approche pratique et théorique sur notre sujet, puis nous détaillerons et commenterons les résultats obtenus.

Ce projet fait partie de l'UE "TER" du Master 1 Informatique de l'Université d'Aix-Marseille (AMU). Il est encadré par les enseignants chercheurs suivants :

JEGOU Phillipe et OSTROWSKI Richard

Table des matières

1 Recherche bibliographique

A travers cette section, nous allons étudier les réductions polynomiales lors de compétitions SAT. Ces compétitions s'organisent sous forme de concours, dans lesquels plusieurs équipes s'affrontent. Le challenge est de pouvoir fournir plusieurs solveurs, capables de résoudre des fichiers contenant des clauses et instances, qui se comptent en milliers, voire en millions.

La recherche bibliographique s'est orientée grâce à la **compétition organisée en 2017** [1]. Ainsi, tout les solveurs et résultats qui seront présentés dans cette partie concerneront l'issue de cette compétition, sauf indication contraire.

1.1 L'environnement d'une compétition

Les compétitions SAT durent environ sur une période de un mois (autour de Mai). Les candidatures ouvrent et les équipes peuvent inscrire leurs membres. Ils ont ensuite un mois pour produire ce qui est requis par les règles de la compétition. Les résultats sont alors annoncés annuellement lors de la SAT Conference, organisée généralement fin Août. En voici les principales règles [2] :

Le code source de tout solveur SAT doit être mis à disposition des jurys (déposé sous licence, permettant l'utilisation de ces derniers dans le domaine de la recherche), à l'exception des solveurs de type NoLimits.

Une à deux page(s) de description du solveur doit(vent) être fourni(s) afin d'en faire une bonne utilisation. Les auteurs et co-auteurs d'un solveur, dans une description doivent correspondre aux auteurs inscrits sur le site d'admission. Les solveurs SAT doivent être conformes à la norme DIMACS (voir la compétition 2009 pour plus de détails).

La visualisation graphique des modèles (en cas d'instances satisfaisables) doit être obligatoire (sauf pour les solveurs No-Limits). En parallèle, toute formule dite UNSAT se doit d'être amenée d'une preuve.

Chaque équipe participante doit soumettre 20 nouvelles instances benchmarks. 10 d'entre-elles se doivent d'être "intéressantes" : ni trop faciles (Solvables par MiniSat en moins d'une minute), ni trop difficiles (non solvables par un des solveurs des participants en plus d'une heure).

1. https ://baldur.iti.kit.edu/sat-competition-2017/
2. https ://baldur.iti.kit.edu/sat-competition-2017/index.php ?cat=rules

1.2 Les différents types de solveurs

Lors de compétitions, plusieurs types de solveurs doivent/peuvent être réalisés par les équipes concurrentes. En voici la liste principale :

- Agile Solvers
- Incremental Solvers
- Main Solvers
- NoLimits Solvers
- Parralel Solvers
- Random Solvers

1.3 Zoom sur types de solveurs SAT

1.3.1 – Le solveur parallèle Glucose [3]

Le solveur Glucose est un solveur basé sur un schéma de scores (présenté pour la première fois en 2009). il résout les clauses en déterminant si elles sont satisfaisables ou non. Depuis 2014, Glucose se caractérise comme étant un solveur travaillant en parallèle. Lors de compétitions, Glucose à remporté les prix suivants (les plus récents) :

1. Certified UNSAT, Applications - 2nde Place, 2015
2. Parallel SAT+UNSAT Applications - 4ème place, 2015

Au classement général, le solveur Glucose développé par Gilles Audemard (CRIL [4]) et Laurent Simon (LaBRI [5]) a été classé meilleur solveur parallèle de la SAT Race 2015 (et troisième dans la catégorie 'résolution incrémentale'). Il est bon à noter que Glucose 3.0 se base en grande partie sur le Solveur MiniSat [6].

Aujourd'hui, Glucose est l'un des solveurs les plus reconnus en France (étant produit essentiellement par des francophones). Il utilise essentiellement les méthodes de résolutions d'instances (DPLL, propagation unitaire et valuation partielle notamment).

1.3.2 – Le solveur CaDiCal [7]

Le but du développement de CaDiCal était en premier lieu d'obtenir un solveur CDCL (Conflict-Driven Clause Learning), facile d'utilisation et de compréhension. Le but était de simplifier radicalement le design et les structures internes des données, mais actuellement, ce but n'est atteint que partiellement. Cependant, ce solveur reste performant quant à ses résultats lors de compétitions :

1. Certified SAT+UNSAT, Applications - 1ère Place, 2017
2. Parallel SAT+UNSAT Applications - 3ème place, 2017

3. http ://www.labri.fr/perso/lsimon/glucose/
4. http ://www.cril.univ-artois.fr/
5. http ://www.labri.fr/
6. http ://minisat.se/
7. http ://fmv.jku.at/cadical/

2 Approche théorique et définitions utiles

2.1 La logique propositionnelle et le problème SAT

Soit V un ensemble de variables propositionnelles. On définit l'ensemble des formules propositionnelles sur V de la façon suivante :

1. toute variable $x \in V$ est une formule.
2. si Φ et Ψ et sont des formules, alors $\Phi \vee \Psi$, $\Phi \wedge \Psi$, $\Phi \Rightarrow \Psi$ et $\neg\Phi$ sont des formules.

Ces formules peuvent être évaluées (à vrai ou à faux) en associant une valeur de vérité à chaque variable. Ainsi étant donné une formule Φ et une interprétation $I : V \rightarrow 0, 1$, on définit $\Phi(I)$ comme étant la valeur de vérité obtenue en remplaçant dans Φ chaque variable par son interprétation et en appliquant les règles usuelles de la logique booléenne. Si $\Phi(I) = 1$ on dit que I satisfait Φ ou que I est un modèle de Φ . On note $mod(\Phi)$ l'ensemble des modèles de Φ.

Le problème SAT est un problème de décision qui prend en entrée une formule propositionnelle et détermine si cette formule est ou non satisfaisable, on peut le définir de la façon suivante :

entrée : une formule propositionnelle
question : est-elle satisfaisable (oui ou non) ?

2.2 La réduction polynomiale : définition

Soient π_1 et π_2 deux problèmes de décision. Une réduction polynomiale de π_1 à π_2 est une application $r : I(\pi_1) \rightarrow I(\pi_2)$ telle que :
$\forall x \in I(\pi_1) : x \in I^+(\pi_1) \Leftrightarrow r(x) \in I^+(\pi_2)$
r est calculable en temps polynomial.

3 Approche pratique : les réductions polynomiales

Cette partie explique en détail l'intégralité des travaux que nous avons effectué. Par rapport aux outils utilisés, tous les programmes implémentés ont été codés en Java. Les formules propositionnelles [8] et les graphes [9] sont modélisés sous le format DIMACS les instances des problèmes CSP sont modélisées sous le format XCSP [10]

3.1 SAT vers 3-SAT

Pour transformer une instance de SAT en une instance de 3-SAT, il faut parcourir l'ensemble des clauses de l'instance initiale et appliquer une des modifications suivantes en fonction de la taille de la clause :

- Si la clause initiale est de longueur 1, l'unique littéral présent dans la clause initiale sera triplé dans la clause finale, par exemple, la clause x deviendra $x \vee x \vee x$.
- Si la clause initiale est de longueur 2, la nouvelle clause contiendra les deux littéraux de la clause initiale, puis on lui ajoutera un des deux littéraux. Par exemple, la clause $x \vee y$ peut devenir $x \vee y \vee x$.
- Si la clause initiale est de longueur 3, elle correspond déjà au spécificités de 3-SAT et n'est pas modifiée.
- Si la clause est de longueur supérieure à 3, alors on va tout d'abord créer une première clause contenant les deux premiers littéraux de la clause initiale suivi d'un littéral intermédiaire (noté a) instancié à vrai. On va ensuite créer une seconde clause contenant a ainsi que le troisième littéral de la clause initiale, suivi d'un autre littéral intermédiaire. On va répéter cette opération jusqu'à ce qu'il reste un ou deux littéraux non traités, la dernière clause sera composée d'un littéral intermédiaire instancié à faux ainsi que du ou des deux derniers littéraux de la clause initiale. Par exemple la clause $v \vee w \vee x \vee y \vee z$ donnera l'ensemble de clauses suivant :

3.2 3-SAT vers 3-COLORATION

Pour effectuer cette réduction polynomiale, on doit transformer une instance de 3-SAT en un graphe. Soit ϕ une formule propositionnelle instance de 3-SAT, on va tout d'abord créer deux sommets pour chaque variable x de ϕ , un pour son affectation à vrai et un autre pour son affectation à faux, une arête va relier chaque nouveau couple de sommet créé de cette manière. Par la suite, on notera V_1 cet ensemble de sommets.

8. Format DIMACS CNF : http ://people.sc.fsu.edu/ jburkardt/data/cnf/cnf.html
9. Format DIMACS pour graphe : http ://people.sc.fsu.edu/ jburkardt/data/cnf/cnf.html
10. Site officiel de XCSP : http ://www.xcsp.org/).

Ensuite, il faut créer trois sommets particuliers appelés Vrai, Faux et Neutre (par la suite, on les notera respectivement T, F, N). On va ensuite relier le sommet N à tous les sommets de V_1.

Maintenant, il faut encore modéliser les contraintes pour chaque clause, pour cela, on va définir un gadget OR qui est de la forme suivante :

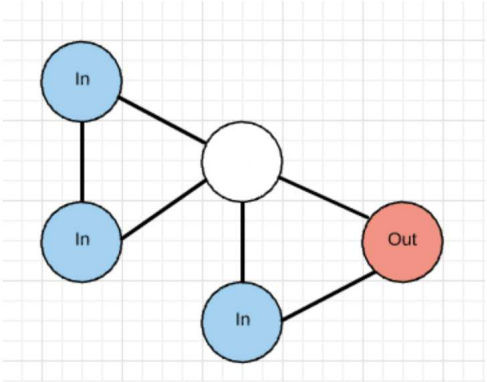

FIGURE 1 – Schéma d'un gadget OR

Il faut maintenant parcourir l'ensemble des clauses de ϕ puis pour chaque clause, créer un gadget OR et connecter aux trois sommets d'entrée (coloriés en bleu) aux sommets deV_1 associés aux littéraux présents dans la clause, puis connecter le sommets de sortie (colorié en rouge) aux sommets F et N décrits précédemment. Par exemple, la formule *phi* ayant pour ensemble de clause $\{a \vee b \vee \neg c; \neg b \vee c\}$ deviendra le graphe suivant :

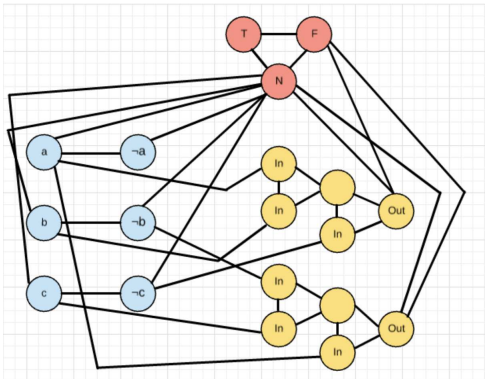

FIGURE 2 – Transformation de ϕ en instance 3- COL

3.3 3-COL vers CSP

La transformation 3-COL vers CSP est relativement triviale. Prenons un graphe. A chaque sommet du graphe on fait correspondre une variable dans l'instance de CSP. Chaque domaine des variables valent $\{0, 1, 2\}$, chaque nombre correspond à une couleur. Puis pour chaque arête du graphe G on ajoute la même contrainte entre les deux sommets. Cette contrainte vaut

$$\{(0,1),(0,2),(1,0),(1,2),(2,0),(2,1)\}.$$

On peut aussi définir cette contrainte par les valeurs que le couple ne doit pas prendre, ce qui rend la contrainte plus concise. On a donc $\{(0,0),(1,1),(2,2)\}$

— Exemple :
Soit G une clique de trois sommets.

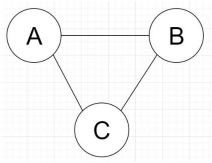

FIGURE 3 – Graphe G à 3 sommets

L'instance CSP correspondante est la suivante :
$X : < a, b, c >$
$D : < D_a = D_b = D_c = \{0, 1, 2\} >$
$C : < C_{ab}, C_{ac}, C_{bc} >$
$R : < RC_{ab} = RC_{bc} = RC_{ac} = \{(0,1),(0,2),(1,0),(1,2),(2,0),(2,1)\} >$

3.4 3-SAT vers VERTEX-COVER

Soit ϕ l'instance initiale de 3-SAT. On va tout d'abord récupérer l'ensemble des variables de ϕ et pour chacune d'elles, on va créer deux sommets, un pour l'affectation positive de la variable, un autre pour l'affectation négative. Ces deux sommets seront ensuite reliés par une arête. On note V_1 l'ensemble des sommets créés de cette manière.

Ensuite, il faut parcourir l'ensemble des clauses de ϕ puis créer un sommet pour chaque littéral, chacun reliés entre eux (comme ϕ est une instance de 3-SAT, on obtient donc un triangle par clause). Pour chaque sommet créé de cette manière, on va également créer une arête le reliant à son équivalent de V_1. Pour ce problème de couverture on fixe k au nombre de clauses de ϕ.

Par exemple, pour la formule ϕ ayant l'ensemble de clauses suivant : $\{a \vee b \vee \neg c, \neg a \vee \neg b \vee c, a \vee \neg b \vee \neg c\}$, on obtiendrait le graphe suivant :

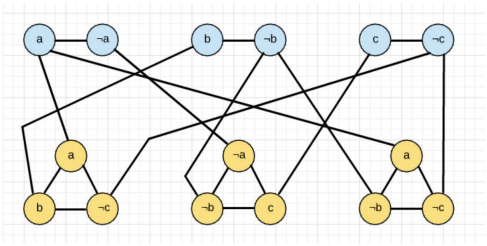

FIGURE 4 – Transformation de ϕ en une instance de $VERTEX\text{-}COVER(k = 3)$

3.5 SAT vers CLIQUE

Soit ϕ l'instance initiale de SAT. On va parcourir l'ensemble des clauses de ϕ et pour chaque littéral, on va créer un nouveau sommet. Ensuite, on va connecter chaque sommet créé de cette manière à tous les autres sommets des autres clauses n'étant pas associés à sa négation. Comme pour la transformation précédente, on va fixer k au nombre de clause de ϕ .

Par exemple, soit ϕ l'instance de SAT possédant l'ensemble de clauses suivant : $\{a \vee b, \neg a \vee b, \neg a \vee \neg b\}$, si l'on applique la transformation décrite ci-dessus à ϕ , on obtient le graphe suivant :

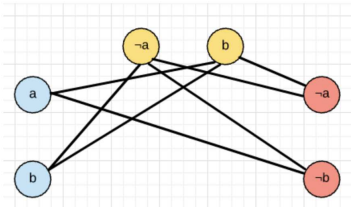

FIGURE 5 – Transformation de ϕ en une instance de $CLIQUE(k = 3)$

3.6 CSP binaire vers SAT

Les transformations de CSP vers SAT sont particulières car il en existe plusieurs, nous avons décidé d'en implémenter deux : le direct encoding et le support encoding.

La première étape est commune aux deux transformations, pour chaque variable CSP x, on va créer n variables propositionnelles (n étant la taille du domaine de x) qui vont modéliser toutes les valeurs que peut prendre x. On doit maintenant modéliser les deux faits suivants :

— Une variable doit prendre au moins une valeur
— Une variable doit prendre au plus une valeur

Le premier fait peut être modélisé par la simple clause $x_1 \vee ...x_n$. Pour le second fait, il faut créer $\frac{n-(n-1)}{2}$ clauses de deux négations entre deux variables différentes.
Pour la seconde étape, cela va dépendre de la transformation utilisée. Le direct encoding va encoder les couples interdits, alors que le support encoding va encoder les couples autorisés. Pour illustrer, prenons le CSP suivant :

$X : < a, b >$
$D : < D_a = D_b = \{0, 1, 2\} >$
$C : < C_{ab} >$
$R : < RC_{ab} = \{(0, 0), (1, 1), (2, 2)\} >$

Les couples présents dans la relation modélisent les couples interdits.

Si on applique le direct encoding à ce CSP, nous obtenons la formule suivante

$$
\begin{array}{l|l}
a_0 \vee a_1 \vee a_2 & \neg b_0 \vee \neg b_2 \\
\neg a_0 \vee \neg a_1 & \neg b_1 \vee \neg b_2 \\
\neg a_0 \vee \neg a_2 & \neg a_0 \vee \neg b_0 \\
\neg a_1 \vee \neg a_2 & \neg a_1 \vee \neg b_1 \\
b_0 \vee b_1 \vee b_2 & \neg a_2 \vee \neg b_2 \\
\neg b_0 \vee \neg b_1 &
\end{array}
$$

En revanche, si l'on applique le support encoding à ce CSP, nous obtenons la formule suivante :

$$
\begin{array}{l|l}
a_0 \vee a_1 \vee a_2 & \neg a_0 \vee b_1 \vee b_2 \\
\neg a_0 \vee \neg a_1 & \neg a_1 \vee b_0 \vee b_2 \\
\neg a_0 \vee \neg a_2 & \neg a_2 \vee b_0 \vee b_1 \\
\neg a_1 \vee \neg a_2 & \neg b_0 \vee a_1 \vee a_2 \\
b_0 \vee b_1 \vee b_2 & \neg b_1 \vee a_0 \vee a_2 \\
\neg b_0 \vee \neg b_1 & \neg b_2 \vee a_0 \vee a_1 \\
\neg b_0 \vee \neg b_2 & \\
\neg b_1 \vee \neg b_2 &
\end{array}
$$

3.7 SAT vers CSP

A rajouter

4 Phases de tests de benchmarks et résultats

En conclusion, LaTeX est particulièrement bien adapté pour rédiger de longs documents.

une case centrée	deux encore centrée	trois à gauche	quatre à droite
six	sept	huit	neuf

5 Conclusion