Une autre conversion de SAT vers CSP

Olivier Roussel

Une autre conversion de SAT vers CSP

Olivier ROUSSEL*

CRIL - CNRS FRE 2499
rue de l'Université, SP 16, 62 307 Lens Cedex, France
olivier.roussel@cril.univ-artois.fr

Résumé

Cet article présente une transformation d'une formule propositionnelle du problème SAT en une autre formule propositionnelle qui, à son tour, peut être directement transformée en un problème CSP binaire. Cette transformation est basée sur une procédure de recherche de modèles locaux qui est contrôlée pour obtenir une transformation qui soit globalement polynomiale. Une méthode pour réduire la taille de la formule finale est proposée. Nous montrons également que le maintien de la k-consistance sur le problème CSP est équivalent à l'utilisation de la propagation unitaire sur une forme compilée de la formule propositionnelle correspondante dans le problème SAT. De notre point de vue, cette transformation établit un autre pont naturel entre SAT et CSP qui pourrait être bénéfique aux deux communautés.

Abstract

This paper presents a transformation of a SAT problem into another SAT problem which, in turn, can be directly transformed into a binary CSP instance. This transformation is based on a local model enumeration procedure which can be controlled to get an overall polynomial transformation. A method to reduce the size of the converted formula is proposed. We also show that maintaining k-consistency on the CSP problem is equivalent to using unit propagation on a compiled form of the equivalent SAT problem. This transformation draws another natural bridge between the SAT and the CSP problems which hopefully, should benefit both communities.

1 Introduction

Le problème de satisfiabilité (SAT) et le problème de satisfaction de contraintes (CSP) sont étroitement liés. Chacun tente d'affecter une valeur à un ensemble de variables de manière à satisfaire un ensemble de contraintes. Dans le problème SAT, les variables ne peuvent prendre que deux valeurs (vrai/faux) et les contraintes qui relient plusieurs variables interdisent certaines affectations de ces variables. Dans le problème CSP, une variable peut prendre de multiples valeurs et les contraintes qui relient deux ou plusieurs variables interdisent également certaines affectations. En général, le problème CSP peut être vu comme une généralisation du problème SAT.

Très souvent, le problème SAT est étudié sur des formules en forme normale conjonctive (CNF) tandis que le problème CSP est étudié sur des contraintes binaires.

Des transformations d'un problème à l'autre ont déjà été proposées [16, 6, 2, 3]. La plupart d'entre elles sont de simples réécritures syntaxiques. Dans cet article, nous présentons une autre transformation qui, à notre avis, montre une relation plus étroite entre les deux problèmes.

Après un rappel de quelques définitions et des transformations existantes, nous introduisons une nouvelle forme normale pour le problème SAT, forme à partir de laquelle la conversion en problème CSP sera immédiate. Nous présentons un algorithme polynomial pour transformer toute formule propositionnelle du problème SAT dans cette nouvelle forme normale. La conversion en instance CSP est ensuite présentée. Nous donnons alors un algorithme pour réduire la taille de la formule obtenue. Enfin, nous présentons une compilation logique qui permet à la propagation unitaire de maintenir la k-consistance.

2 Définitions et transformations existantes

Une variable propositionnelle est soit vraie, soit fausse. Un littéral est une variable propositionnelle ou sa négation. Une clause est une disjonction de littéraux. Une clause est vraie si et seulement si au moins l'un de ses littéraux est vrai. Une formule propositionnelle du problème

*Ce travail a été soutenu en partie par l'IUT de Lens, le CNRS et la région Nord/Pas-de-Calais dans le cadre du programme TACT.

SAT en forme normale conjonctive (CNF) est une conjonction de clauses. Une formule CNF est vrai ssi chacune de ses clauses est vraie. Un modèle partiel d'une formule est une affectation de certaines variables qui rend la formule vraie (autrement dit, on peut considérer qu'il s'agit d'une conjonction de littéraux qui implique la formule). Les modèles que nous considérons dans cet article sont tous partiels et nous les appellerons simplement modèles. Un modèle minimal est un modèle tel qu'aucun de ses sous-ensembles stricts ne soit un modèle. Une couverture de modèles d'une formule f est un ensemble M de modèles tel que tout modèle de f est subsumé par un élément de M. Un impliqué est une clause qui est conséquence logique de la formule (i.e. satisfaite par tous les modèles de la formule).

Une variable CSP peut prendre n'importe quelle valeur parmi l'ensemble D (domaine) des valeurs possibles pour cette variable. Une contrainte CSP est définie sur un ensemble de variables et interdit certaines affectations de ces variables. Quand toutes les contraintes sont définies sur exactement deux variables, le CSP est dit binaire. Une contrainte est satisfaite quand toutes les variables concernées par la contrainte ont une valeur qui est compatible avec les valeurs des autres variables de la contrainte. Une solution du problème CSP est une affectation des variables qui satisfait toutes les contraintes.

Un certain nombre de codages ont été proposés pour convertir un problème SAT en problème CSP (et inversement). Nous nous concentrons ici uniquement sur la conversion de SAT vers CSP [16, 6, 2, 3]. Dans cette présentation, b_i désigne une variable booléenne, l_i désigne un littéral, C_i désigne une clause, x_i désigne une variable CSP qui peut prendre une valeur de son domaine D_{x_i}. $\mathbb{T}$ et $\mathbb{F}$ représentent les constantes vrai et faux.

- **Dual encoding [16]** : pour chaque clause C_i du problème SAT, on crée une variable CSP x_i qui représente les manières possibles de satisfaire C_i. Le domaine de x_i est l'ensemble des affectations des k variables de C_i qui satisfont C_i (2^k-1 affectations). Des contraintes binaires sont définies entre chaque paire de variables CSP qui sont créées à partir d'une même variable SAT. Ces contraintes imposent que les variables SAT communes doivent avoir les mêmes valeurs dans les interprétations qui correspondent aux valeurs des variables CSP.

  Exemple 1: La formule du problème SAT $(a \vee b) \wedge (\neg b \vee c)$ correspond dans ce codage à un CSP ayant 2 variables. Le domaine de x_1 est $\{< a = \mathbb{T}, b = \mathbb{T} >, < a = \mathbb{T}, b = \mathbb{F} >, < a = \mathbb{F}, b = \mathbb{T} >\}$ qui sont les interprétations qui satisfont la première clause et le domaine de x_2 est $\{< b = \mathbb{F}, c = \mathbb{T} >, < b = \mathbb{F}, c = \mathbb{F} >, < b = \mathbb{T}, c = \mathbb{T} >\}$. Une contrainte binaire relie ces deux variables et stipule que $< a = \mathbb{T}, b = \mathbb{T} >, < a = \mathbb{F}, b = \mathbb{T} >$ sont incompatibles avec $< b = \mathbb{F}, c = \mathbb{T} >, < b = \mathbb{F}, c = \mathbb{F} >$ et que $< a = \mathbb{T}, b = \mathbb{F} >$ est incompatible avec $< b = \mathbb{T}, c = \mathbb{T} >$.

- **Hidden variable encoding [16]** : pour chaque clause C_i de la formule du problème SAT, on crée une variable CSP x_i qui représente les manières possibles de satisfaire C_i. Le domaine de x_i est l'ensemble des affectations des k variables de C_i qui satisfont C_i (2^k-1 affectations). Pour chaque variable SAT b_i, on crée aussi une variable x_{b_i} ayant comme domaine $\{\mathbb{T}, \mathbb{F}\}$. Des contraintes binaires sont définies entre un x_{b_j} et un x_i qui est associé à une clause qui contient b_j. Cette contrainte stipule que l'interprétation correspondant à la valeur de x_i doit donner à b_j une valeur qui soit la même que celle de x_{b_j}.

  Exemple 2: La formule du problème SAT $(a \vee b) \wedge (\neg b \vee c)$ correspond dans ce codage à un CSP ayant initialement deux variables. Le domaine de x_1 est $\{< a = \mathbb{T}, b = \mathbb{T} >, < a = \mathbb{T}, b = \mathbb{F} >, < a = \mathbb{F}, b = \mathbb{T} >\}$ qui sont les interprétations qui satisfont la première clause et le domaine de x_2 est $\{< b = \mathbb{F}, c = \mathbb{T} >, < b = \mathbb{F}, c = \mathbb{F} >, < b = \mathbb{T}, c = \mathbb{T} >\}$. Trois autres variables sont créées : x_a, x_b et x_c avec pour domaine $\{\mathbb{T}, \mathbb{F}\}$. Une contrainte relie x_1 à x_a et indique que $\{< a = \mathbb{T}, b = \mathbb{T} >, < a = \mathbb{T}, b = \mathbb{F} >\}$ sont incompatibles avec $x_a = \mathbb{F}$ et que $\{< a = \mathbb{F}, b = \mathbb{T} >\}$ est incompatible avec $x_a = \mathbb{T}$. Des contraintes similaires sont établies entre x_1 et x_b, x_2 et x_b, x_2 et x_c.

- **Literal encoding [16, 3]** : pour chaque clause C_i de la formule du problème SAT, on crée une variable CSP x_i qui représente un littéral de C_i qui est vrai. Des contraintes binaires sont définies entre chaque paire de variables CSP qui sont créées à partir de clauses qui contiennent des littéraux opposés. Ces contraintes stipulent que si la première variable x_i prend comme valeur l_k, alors la seconde variable x_j ne peut prendre $\neg l_k$ comme valeur.

  Exemple 3: La formule du problème SAT $(a \vee b) \wedge (\neg b \vee c)$ correspond dans ce codage à un CSP de 2 variables. Le domaine de x_1 est $\{a, b\}$ ce qui signifie que affecter a ou b à $\mathbb{T}$ satisfait la première clause. Le domaine de x_2 est $\{\neg b, c\}$. Une contrainte binaire est définie entre x_1 et x_2 et indique que b et $\neg b$ sont des valeurs incompatibles.

- **Extended literal encoding [6]** : pour chaque clause C_i de la formule du problème SAT, on crée une variable CSP x_i qui représente un littéral de C_i qui est vrai. Pour chaque variable SAT b_i, on crée aussi une variable x_{b_i} ayant pour domaine $\{\mathbb{T}, \mathbb{F}\}$. Des contraintes binaires sont définies entre un x_{b_j} et un x_i associé à une clause qui contient b_j. Cette contrainte indique que lorsque x_{b_j} est faux, x_i ne peut prendre b_j comme valeur.

Exemple 4: La formule du problème SAT $(a \vee b) \wedge (\neg b \vee c)$ correspond dans ce codage à un CSP qui initialement contient deux variables. Le domaine de x_1 est $\{a, b\}$ ce qui signifie que donner la valeur vrai à a ou b permet de satisfaire la première clause et le domaine de x_2 est $\{\neg b, c\}$. Trois autres variables sont créées x_a, x_b et x_c avec pour domaine $\{\mathbb{T}, \mathbb{F}\}$. Une contrainte relie x_1 à x_a et indique que $x_1 = a$ est incompatible avec $x_a = \mathbb{F}$. Une autre contrainte entre x_1 et x_b stipule que $x_1 = b$ est incompatible avec $x_b = \mathbb{F}$. Une troisième contrainte lie x_2 à x_b et indique que $x_1 = \neg b$ est incompatible avec $x_b = \mathbb{T}$. La dernière contrainte entre x_2 et x_c indique que $x_2 = c$ est incompatible avec $x_c = \mathbb{F}$.

– **Non-binary encoding [16]** : pour chaque variable b_i de la formule du problème SAT, on crée une variable CSP x_{b_i} qui représente la valeur prise par b_i. Le domaine de x_{b_i} est donc $\{\mathbb{T}, \mathbb{F}\}$. Pour chaque clause C_i de la formule du problème SAT, on définit une contrainte entre les x_{b_k} qui correspondent aux variables de C_i et qui interdit l'interprétation des x_{b_k} qui falsifient la clause.

Exemple 5: La formule du problème SAT $(a \vee b) \wedge (\neg b \vee c)$ correspond dans ce codage à un CSP de 3 variables. Le domaine de x_a, x_b et x_c est $\{\mathbb{T}, \mathbb{F}\}$. La première clause correspond à une contrainte entre x_a et x_b et interdit l'affectation $x_a = \mathbb{F}, x_b = \mathbb{F}$. La seconde clause correspond à une contrainte entre x_b et x_c qui interdit l'affectation $x_b = \mathbb{T}, x_c = \mathbb{F}$.

Ces codages d'une formule du problème SAT en un problème CSP sont à la fois simples et immédiats, mais ils sont aussi quelque peu artificiels parce qu'ils sont basés sur des transformations purement syntaxiques. À l'inverse, la conversion que nous proposons est fondée sur une transformation sémantique et de ce fait, elle devrait être moins sensible aux artefacts syntaxiques. Plus précisément, cette conversion calcule des modèles de sous-ensembles de la formule et les assemble pour former une instance CSP. De ce fait, les propriétés du problème CSP dépendent plus de la sémantique du problème que de sa représentation syntaxique. Dans certains cas simples, la conversion peut même résoudre directement la formule.

3 La Pigeon-Hole Normal Form

La transformation que nous décrivons génère une formule ayant une structure assez similaire à celle du problème des pigeons. De ce fait, nous introduisons d'abord une forme normale dite "des pigeons" (*pigeon-hole normal form* ou PHNF en abrégé) qui correspond à la notion de structure similaire au problème des pigeons.

Le problème des pigeons est un problème SAT bien connu qui a servi entre autres à prouver l'intraitabilité de la résolution ([8]). Ce problème est également souvent utilisé pour tester des extensions du formalisme SAT (les cardinalités par exemple [1]) mais aussi certaines méthodes de résolution (dont les symétries [10]). D'un certain point de vue, ce problème semble représenter la plupart des difficultés combinatoires qui se cachent derrière un problème SAT quelconque. Certaines personnes ont même émis la conjecture selon laquelle une instance SAT difficile contenait de manière cachée un problème des pigeons [5]. La transformation présentée ici vient étayer en partie ce point de vue.

Le problème des pigeons code dans une formule propositionnelle en CNF la possibilité de placer n_p pigeons dans n_h pigeonniers. Les règles de ce problème établissent que

1. chaque pigeon doit être dans un pigeonnier,
2. un pigeonnier ne peut contenir plus d'un pigeon,
3. un pigeon ne peut être dans plus d'un pigeonnier.

Bien sûr, le problème est satisfiable ssi $n_h \geq n_p$. La solution de ce problème est triviale quand on utilise des notions élémentaires d'arithmétique mais se révèle beaucoup plus ardue quand on utilise les règles d'inférences habituelles en logique propositionnelle. Quand on s'intéresse uniquement à la satisfiabilité de ce problème, la troisième règle peut être omise puisque qu'elle ne supprime que des interprétations qui sont des variantes d'autres solutions du problème [1].

Soit P_h^p la variable propositionnelle qui indique que le pigeon p est placé dans le pigeonnier h. La version de ce problème des pigeons destinée à prouver la satisfiabilité est codé sous forme CNF comme suit :

$$\begin{array}{ll} 1) & \bigwedge_{p=1}^{n_p} \bigvee_{h=1}^{n_h} P_h^p \\ 2) & \bigwedge_{\forall h, \forall p_1, p_2 \ p_1 \neq p_2} \neg P_h^{p_1} \vee \neg P_h^{p_2} \end{array}$$

On peut observer que ce codage contient d'abord un ensemble de longues clauses qui contiennent uniquement des littéraux positifs (sans littéraux communs) suivi d'un ensemble de clauses binaires contenant la négation de littéraux présents dans des clauses différentes du premier ensemble. Ces clauses binaires représentent en fait des exclusions mutuelles tandis que les clauses positives représentent les différentes possibilités pour placer chaque pigeon. À partir de cette observation, on définit la *pigeon-hole normal form* comme une relaxation de la structure de la CNF codant le problème des pigeons.

Définition 6: Une formule propositionnelle est dite en *pigeon-hole normal form* (PHNF) ssi

1. elle est sous forme normale conjonctive
2. la formule peut être partitionnée en deux ensembles de clauses P (positif) et B (binaire) tels que

[1] Il faut cependant noter que cette troisième règle ne peut être omise si l'on veut obtenir l'ensemble des modèles du problème.

(a) P contient des clauses de longueur quelconque formées uniquement à partir de littéraux positifs et telles qu'aucun littéral de P n'est présent dans plus d'une clause

(b) B contient uniquement des clauses binaires dont le premier littéral est la négation d'un littéral de $C_i \in P$ et le second littéral est la négation d'un littéral de $C_j \in P, i \neq j$

Une propriété intéressante de la PHNF est que, quand une formule est satisfiable, alors il existe un modèle où chaque clause de la partie P est satisfaite par un unique littéral. Cette propriété doit bien sûr être utilisée pour résoudre ce genre de formule. Elle découle du fait qu'il est strictement inutile d'assigner un second littéral à vrai dans une clause de P dans la mesure où cela ne peut pas aider à satisfaire une autre clause de P (il n'y a pas de littéral commun entre les clauses de P) et par ailleurs, rendre un littéral vrai ne peut pas augmenter le nombre de clauses satisfaites dans la partie B (parce des littéraux de B deviennent faux dans ce cas).

La section qui suit montre que cette PHNF est effectivement une forme normale puisque toute formule propositionnelle peut être convertie dans cette forme en préservant l'équivalence entre les formules du point de vue de la satisfiabilité.

4 Transformation d'une formule propositionnelle du problème SAT en PHNF

Proposition 7: Toute formule propositionnelle f peut être convertie en une formule f' en *pigeon-hole normal form* (PHNF) telle que f est satisfiable ssi f' est satisfiable et telle qu'il y ait en plus une bijection directe entre les modèles de f et ceux de f'.

Dans cette proposition, f et f' ne sont pas équivalentes au sens classique du terme puisqu'elles ne sont pas construites sur le même vocabulaire et que, de ce fait, elles ne peuvent avoir les mêmes modèles. Cependant, f et f' sont équivalentes au sens où f est satisfiable $\Leftrightarrow$ f' est satisfiable. De plus, il est facile d'obtenir les modèles de f à partir des modèles de f' et inversement (temps linéaire).

À partir d'une formule propositionnelle générale, la première étape est de la convertir en forme normale conjonctive (CNF). Cette transformation est linéaire à condition d'introduire de nouveaux symboles de variables [14].

On peut donc en toute généralité se restreindre au cas où la formule f à convertir est en CNF et qu'elle contient les clauses C_i pour $i \in \{1..n\}$. On partitionne cet ensemble de clauses en un ensemble de sous-ensembles S_j disjoints de clauses. La manière dont on partitionne la formule a peu d'importance si l'on ne se fixe pas d'autre but que celui d'obtenir une formule équivalente en PHNF. Nous verrons plus loin que si l'on souhaite effectuer le calcul en temps polynomial, il faudra imposer une condition supplémentaire sur chacun des S_j.

On note $S_j = \{C_{k_l}\}$ qui doit vérifier

$$\bigcup_j S_j = \{C_i, i \in \{1..n\}\} \text{ et } \forall i, \forall j S_i \cap S_j = \emptyset$$

Pour chaque ensemble S_j de clauses, on énumère une couverture de modèles de S_j en utilisant n'importe quelle méthode ad hoc. Soit $I(S_j) = I^j_1, \ldots I^j_{n_j}$ la couverture de modèles choisie pour S_j. Chaque modèle I^j_k est une conjonction de littéraux de la formule ($I^j_k = \bigwedge_i l_i$). Ces modèles sont des solutions locales du problème de satisfiabilité limité à S_j. Pour trouver une solution globale, il suffit de trouver une solution locale à chaque S_j qui soit compatible avec les solutions locales choisies pour les autres sous-ensembles. Une solution locale $I^{j_1}_{k_1}$ est incompatible avec une autre solution locale $I^{j_2}_{k_2}$ si et seulement si $\exists l$ t.q. $l \in I^{j_1}_{k_1}$ et $\neg l \in I^{j_2}_{k_2}$.

À partir de la couverture des modèles de S_j, on peut construire une formule f' qui est vraie ssi les solutions locales peuvent être assemblées pour former une solution globale. f' est construite avec de nouvelles variables propositionnelles comme suit. On associe à chaque modèle I^j_k une nouvelle variable propositionnelle v^j_k et l'on définit f' comme

$$\bigwedge_j \bigvee_k v^j_k$$
$$\wedge$$
$$\bigwedge_{j_1,k_1,j_2,k_2 \text{ t.q. } I^{j_1}_{k_1} \text{ et } I^{j_2}_{k_2} \text{ sont incompatibles}} \neg v^{j_1}_{k_1} \vee \neg v^{j_2}_{k_2}$$

Cette formule est évidemment en *pigeon-hole normal form*. Par construction, elle est satisfiable si et seulement si la formule initiale f est satisfiable. De plus, un modèle de f' se transforme facilement en un modèle de f en remplaçant chaque v^j_k qui est à vrai par les littéraux de I^j_k. Inversement, un modèle de f est facilement converti en un modèle de f' en le décomposant en une conjonction des modèles locaux nécessaires.

Il a été précisé que $I(S_j)$ était une couverture de modèles de S_j. Une telle couverture n'est bien évidemment pas unique. Il est clair qu'il faut privilégier une couverture qui contienne un nombre restreint de modèles pour obtenir une représentation PHNF qui sera plus courte et, selon la méthode utilisée, pour pouvoir accélérer le calcul de cette couverture. On peut en particulier opter pour une couverture de modèles minimaux mais l'on verra par la suite qu'un autre type de couverture pourra présenter quelque avantage.

Il y a bien sûr diverses méthodes pour générer une couverture de modèles minimaux de S_j. La littérature contient

de nombreux algorithmes à ce sujet. Quand les ensembles S_j sont relativement petits, on peut utiliser des méthodes très simples telles que les méthodes syntaxiques fondées sur la distributivité de $\wedge$ sur $\vee$ ou des méthodes sémantiques fondées sur des variantes de la procédure de Davis et Putnam [13, 4]. Si les ensembles S_j sont plus grands, il vaut mieux utiliser de meilleurs algorithmes tels que [15, 9].

Exemple 8: Soit $f = (a \vee b \vee c) \wedge (d \vee e) \wedge (\neg a \vee \neg b) \wedge (a \vee b)$ On choisit $S_1 = (a \vee b \vee c) \wedge (d \vee e)$ et $S_2 = (\neg a \vee \neg b) \wedge (a \vee b)$. On énumère une couverture de modèles minimaux de S_1 et S_2 et l'on assigne à chacun d'eux une nouvelle variable propositionnelle :

var.	modèle minimal
v_1^1	$a \wedge d$
v_2^1	$a \wedge e$
v_3^1	$b \wedge d$
v_4^1	$b \wedge e$
v_5^1	$c \wedge d$
v_6^1	$c \wedge e$
v_1^2	$\neg a \wedge b$
v_2^2	$a \wedge \neg b$

La transformation en PHNF donne f'

$$
\begin{array}{l}
v_1^1 \vee v_2^1 \vee v_3^1 \vee v_4^1 \vee v_5^1 \vee v_6^1 \\
v_1^2 \vee v_2^2 \\
\hline
\neg v_1^1 \vee \neg v_1^2 \\
\neg v_2^1 \vee \neg v_1^2 \\
\neg v_3^1 \vee \neg v_2^2 \\
\neg v_4^1 \vee \neg v_2^2
\end{array}
$$

Il est facile d'obtenir un modèle de f' en choisissant $v_6^1 = \mathbb{T}, v_2^2 = \mathbb{T}$ et toutes les autres variables comme étant fausses ce qui signifie que $c \wedge e \wedge a \wedge \neg b$ est un modèle de f.

5 Complexité de la transformation

Proposition 9: Toute formule propositionnelle f peut être convertie **en temps polynomial** en une formule f' en *pigeon-hole normal form* (PHNF) telle que f est satisfiable ssi f' est satisfiable et telle qu'il y ait en plus une bijection directe entre les modèles de f et ceux de f'.

La méthode de transformation que nous avons présentée précédemment commence par une mise sous forme CNF qui est polynomiale à condition de pouvoir introduire de nouveaux symboles de variables [14]. Elle partitionne ensuite l'ensemble de clauses (temps linéaire). Une fois que les modèles de chaque S_j sont calculés, la création de f' est polynomiale.

Malheureusement, un calcul sans précaution des modèles de S_j est exponentiel. Un exemple simple est celui où k clauses de longueur n avec des littéraux tous positifs et différents ont une couverture de modèles qui contient n^k modèles minimaux. Cependant, il est possible de se fixer à l'avance une limite qui soit polynomiale et s'arranger pour choisir des ensembles S_j de sorte à ne jamais avoir un nombre de modèles plus grand que cette borne polynomiale.

Supposons que l'on décide d'obtenir moins de m modèles par ensemble S_j. Il est possible d'utiliser un algorithme incrémental pour obtenir les modèles minimaux des k premières clauses de S_j. Soit cur le nombre courant de modèles de ces premières k clauses. Avant de calculer les modèles minimaux des premières $k+1$ clauses, on peut facilement obtenir une estimation du nombre maximum de modèles que l'on obtiendra en multipliant cur par la taille de la clause de numéro $k+1$. Si cette estimation dépasse notre limite m, il suffit de cesser d'ajouter des clauses à S_j et de placer la clause numéro $k+1$ dans un nouvel ensemble S_{j+1}. Ce faisant, le nombre total de modèles minimaux que l'on génère est au plus m multiplié dans le pire des cas par le nombre de clauses de la CNF, ce qui donne bien une borne polynomiale. Bien sûr, m doit être une limite raisonnable et le plus petit m que l'on puisse choisir pour toujours réussir la transformation doit être la longueur de la plus longue clauses de la CNF (auquel cas, il est possible que certains S_j ne contiennent qu'une seule clause).

À partir du moment où l'on garantit qu'à aucun moment le nombre de modèles que l'on manipule ne dépasse la limite polynomiale m que l'on s'est fixée, non seulement le résultat de l'algorithme a une taille polynomiale comme indiqué ci-dessus, mais également le calcul peut être effectué en temps polynomial. En effet, si l'on opte pour un algorithme aussi simple que le calcul de la distributivité du $\wedge$ sur le $\vee$ et qu'à tout instant le nombre de modèles est majoré par m (conformément à ce que l'on s'impose), alors quand on calcule la distributivité des modèles courants sur une clause de taille l, on génère au plus lm résultats intermédiaires qui seront éventuellement supprimés pour obtenir finalement au plus m modèles. De ce fait, cet algorithme a une complexité en temps qui est bornée par le nombre de clauses multiplié par la taille de la plus longue clause et enfin multiplié par la borne m fixée à l'avance.

Adapter la taille des ensembles S_j est clairement une astuce grossière pour éviter l'explosion combinatoire. Cependant, cela nous fournit incontestablement un algorithme polynomial. Le principe ici est de découper les calculs en morceaux suffisamment petits pour qu'ils soient polynomiaux. Ce découpage n'est en aucun cas une approximation et la formule que l'on obtient *in fine* est bien équivalente à la formule initiale.

La borne m que l'on choisit est un paramètre important de la conversion. Plus m est grand, plus lente sera la conversion mais en même temps on éliminera lors de ce pré-traitement plus d'interprétations qui falsifient f et la résolution de f' sera plus facile. À l'inverse, plus m est

petit, plus rapide sera la conversion mais en même temps on éliminera très peu d'interprétations qui falsifient f et la résolution de f' sera plus difficile.

6 De la PHNF au problème CSP

Une fois que f a été transformée en PHNF, la conversion en problème CSP est immédiate. Il faut se souvenir que la PHNF se compose d'un ensemble P de clauses positives et d'un ensemble B de clauses binaires et négatives qui représentent des exclusions.

À chaque clause $C_i = \{l_1, \dots l_n\}$ de P, on associe une variable x_i de domaine $D_i = \{l_1, \dots l_n\}$ comme dans le *literal encoding*. L'ensemble B des clauses représente les contraintes entre les variables x_i. Il suffit de lister les clauses binaires $\neg l_i \vee \neg l_j$ qui contiennent un littéral de x_i et un littéral de x_j et de les transformer en une contrainte binaire qui stipule que $x_i = l_i$ et $x_j = l_j$ sont incompatibles. S'il y a c clauses dans l'ensemble P qui sont toutes de taille inférieure à m, il y a dans le pire des cas $c(c-1)m^2/2$ clauses binaires qui seront générées pour représenter les exclusions entre les $c(c-1)/2$ paires de variables. Cette conversion est donc bien polynomiale.

D'un certain point de vue, la transformation globale proposée est une généralisation de certaines des conversions présentées dans la première partie de cet article. Il faut cependant noter que la transformation proposée ici est fondée sur un algorithme sémantique et devrait de ce fait être plus résistante aux artifices syntaxiques que l'on peut trouver dans certaines formules. De plus, la borne m qui est un paramètre de la transformation nous laisse choisir la quantité de pré-traitement effectuée. Une faible valeur de m donnera une transformation plutôt syntaxique tandis qu'une grande valeur de m pourra éventuellement déterminer la satisfiabilité de la formule pendant la phase de prétraitement.

Le *literal encoding* [16, 3] qui est la conversion la plus proche de celle que l'on propose peut être vue comme un cas particulier de notre transformation où chaque paquet de clause ne contient qu'une unique clause.

Nous présentons maintenant un algorithme pour effectuer une conversion plus intelligente et réduire la taille de f'.

7 Réduction de la taille de la formule convertie

Dans l'algorithme de la section précédente, la manière dont on partitionne la CNF n'avait pas fait l'objet d'une attention particulière – hormis le fait qu'il fallait fixer une borne au nombre de modèles de chaque paquet S_j. Toutefois, le choix des clauses rassemblées dans un paquet a une influence capitale sur la taille finale de f'. Un mauvais choix peut aboutir à des domaines dont la taille serait beaucoup plus grande que ce que l'on peut obtenir avec un choix intelligent.

Exemple 10: Soit $f = (a \vee b) \wedge (c \vee d) \wedge (\neg a \vee \neg b) \wedge (\neg c \vee \neg d)$. Si l'on sélectionne les deux premières clauses dans S_1 et les deux dernières dans S_2, on obtient des domaines de taille 4. En revanche, si l'on place les clauses construites sur $\{a, b\}$ dans S_1 et les clauses construites sur $\{c, d\}$ dans S_2 le domaine des variables aura seulement une taille de 2.

Ce qu'il faut faire, c'est rassembler dans un même paquet les clauses qui ont le plus petit nombre de modèles. Certes, il ne s'agit pas de résoudre exactement ce problème d'optimisation parce qu'on souhaite avant tout avoir une transformation qui reste polynomiale dans son ensemble. Nous utilisons donc une approximation qui permet de réduire le nombre de modèles de chaque S_j en rassemblant ensemble les clauses qui sont sémantiquement liées.

Nous présentons ici une telle approximation polynomiale qui se fonde sur les propriétés de distributivité des connecteurs logiques. L'idée de cet algorithme est de sélectionner d'abord la plus petite clause (parce qu'isolément, elle a le plus petit nombre de modèles minimaux) et de calculer l'ensemble de ses modèles minimaux (qui est simplement l'ensemble de ses littéraux). Puis, à chaque étape, on sélectionne une clause et l'on multiplie l'ensemble de modèles obtenu jusqu'ici par la clause choisie pour obtenir le nouvel ensemble de modèles. Cela signifie que, pour chaque modèle I et pour chaque littéral l de la clause choisie, on génère un nouveau modèle $I \cup l$ (distributivité). Bien évidemment, les interprétations contradictoires ou subsumées sont retirées à chaque étape. Il est intéressant de noter que, quand un modèle I et une clause C partagent un littéral commun l ($l \in I \cap C$) alors tous les modèles générés seront subsumés par I. Ce type de modèle est appelé modèle raccourci. Pour accélérer le test de subsomption, il faut noter que seuls les modèles raccourcis peuvent subsumer un autre modèle.

Pour réduire le nombre de modèles que l'on génère, il faut sélectionner une clause qui, vraisemblablement, va donner le plus petit nombre de modèles. Pour obtenir une estimation raisonnable, on prend en compte le nombre d'interprétations contradictoires qui seront supprimées ainsi que le nombre de modèles raccourcis. Nous ignorons toutefois les autres cas de subsomption (leur estimation serait trop coûteuse en temps). De ce fait, nous estimons le nombre de modèles qu'une clause va générer par le score ci-dessous :

$$\begin{aligned} score(C) \quad &= \text{card(modèles partageant un littéral avec } C) \\ &+ \textstyle\sum_{\text{I modèle t.q. } I \cap C = \emptyset} \text{card(littéraux de } C \text{ qui} \\ &\quad \text{ne sont pas l'opposé de littéraux de } I) \end{aligned}$$

Un calcul incrémental de ces scores est assez facile à implanter et se révèle assez efficace. À chaque étape, la clause choisie est celle qui a le plus petit score.

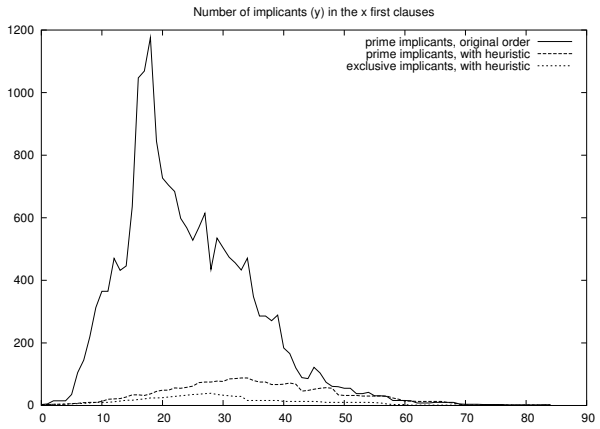

FIG. 1 – Nombre de modèles des premières x clauses d'une formule aléatoire en fonction de x

Une autre technique pour réduire le nombre de modèles est de générer des modèles exclusifs (i.e. tels que pour chaque paire I,I' de modèles, I' contient au moins un littéral qui est l'opposé d'un littéral de I). Cela donne des modèles qui sont moins nombreux, mais plus longs. Générer ce type de modèle ne requiert qu'une modification mineure de l'algorithme précédemment présenté. L'étape de distributivité d'une clause sur un modèle est simplement modifiée comme suit : pour chaque modèle I et pour chaque littéral l de la clause choisie, on génère un nouveau modèle $I \cup l \cup N$ où N est l'ensemble des opposés des littéraux précédemment ajoutés à I. Par exemple, la multiplication d'un modèle $a \wedge b$ par une clause $c \vee d \vee e$ donne $a \wedge b \wedge c$, $a \wedge b \wedge \neg c \wedge d$ et $a \wedge b \wedge \neg c \wedge \neg d \vee e$.

La figure 1 représente en fonction de x le nombre de modèles qu'ont les premières x clauses d'une formule 3-CNF aléatoire (20 variables, 85 clauses) et ce, avec ou sans l'heuristique de choix de la clause et en utilisant des modèles minimaux ou exclusifs. Ce graphique est donné pour une seule formule mais il est représentatif de la forme de ce graphe pour une quelconque formule aléatoire. On constate expérimentalement que l'heuristique fournie réussit effectivement à réduire le nombre de modèles et que l'utilisation de modèles exclusifs réduit également le nombre de modèles dans la couverture.

La transformation d'une formule propositionnelle du problème SAT en un problème CSP est raisonnablement efficace. Le tableau 2 fournit le temps de conversion, le nombre de variables CSP obtenues et le nombre de couples interdits pour quelques instances SAT issues des jeux de test de la compétition SAT 2003 [11]. Ce tableau fournit ces données pour différentes bornes sur la taille des domaines CSP (m) et avec la génération de modèles minimaux ou exclusifs. On peut y vérifier qu'une borne plus grande sur la taille des domaines nécessite plus de temps de calcul mais génère moins de variables. De la même manière, l'utilisation de modèles exclusifs génère moins de variables mais plus de conflits entre variables.

8 k-consistance, propagation unitaire et compilation logique

Dans cette section, nous détaillons quelques techniques de compilation logique qui peuvent être appliquées à la PHNF pour permettre à un prouveur SAT de maintenir automatiquement la k-consistance. Cela montre que maintenir la k-consistance sur un problème CSP est équivalent à utiliser la propagation unitaire sur la formule correspondante du problème SAT compilée pour rendre la propagation unitaire plus complète. Nous rappelons d'abord quelques définitions :

Définition 11: Un problème CSP est k-consistant ssi pour tout ensemble de $k-1$ variables distinctes et toute instanciation consistante I de ces $k-1$ variables, il existe une extension de I à toute k-ième variable telle que le k tuple est consistant.

Maintenir l'arc-consistance qui est le nom commun de la 2-consistance revient à supprimer du domaine d'une variable une valeur qui n'a plus de support[2] dans le domaine d'une autre variable. Le maintien de la chemin-consistance qui est le nom habituel de la 3-consistance revient à modifier une contrainte entre deux variables de sorte à interdire une assignation de ces variables qui n'aurait pas de support commun dans le domaine d'une troisième variable.

Le principe général pour maintenir la k-consistance consiste à détecter que, quand $k-1$ variables sont instanciées, une k-ième variable n'a plus dans son domaine de valeur compatible avec l'instanciation de ces $k-1$ variables. Quand cette situation est détectée, une contrainte d'arité $k-1$ doit être mise en place entre les $k-1$ variables pour interdire l'instanciation qui ne peut être étendue à la k-ième variable.

L'application de ce principe au point de vue de la satisfiabilité est assez simple. Soit $l_1 \vee l_2 \vee \ldots l_n$ la clause de la PHNF représentant les valeurs possibles de la k-ième variable CSP et soit m_i le littéral représentant la valeur assignée à la variable $x_i, i \in 1..k-1$. Quand il n'y a pas d'assignation de la k-ième variable qui soit compatible avec l'instanciation des autres variables, cela signifie que pour tout $j \in 1..n$ il y a un impliqué $\neg l_j \vee C_j$ de la PHNF tel que C_j contient uniquement l'opposé de certains littéraux m_i. Pour obtenir une nouvelle contrainte qui va interdire cette instanciation des $k-1$ variables, il suffit d'appliquer la règle de résolution à la clause $l_1 \vee l_2 \vee \ldots l_n$ et à l'ensemble de clauses $\neg l_j \vee C_j$. Il faut noter que la résolvante que l'on obtient est bien sûr un impliqué de la PHNF et contient au plus $k-1$ littéraux négatifs issus d'au plus $k-1$ clauses.

[2] i.e. plus de valeur compatible

Formule			max. dom.=20, minim.			max. dom.=20, excl.			max. dom.=60, minim.			max. dom.=60, excl.		
name	#var	#cl	time	#var	#confl.	time	#var	#confl.	time	#var	#confl.	time	#var	#confl.
x1.1_64	190	506	0.04	47	14464	0.03	46	12632	0.07	34	129876	0.07	34	140800
urqh3x5	89	1040	0.16	190	102706	0.06	66	84045	0.13	35	90411	0.13	33	128925
am_4_4	433	1458	0.18	94	50437	0.00	89	53017	0.34	56	310749	0.33	53	369972
urqh5x5	153	2016	0.55	383	232312	0.39	129	201540	0.42	66	210950	0.50	62	356415
unif-r4.25-v600-c2550-01	600	2550	1.29	674	403560	1.88	486	823668	2.49	436	2296751	3.03	326	6053936
hidden-k3-s0-r6-n500-01	500	3000	1.69	776	642544	1.76	546	1297596	2.68	494	3685871	4.06	363	9901419

FIG. 2 – Conversion d'une formule du problème SAT en un problème CSP

Exemple 12: Soit $a \vee b \vee c$ la clause représentant une variable CSP v_1 de domaine a, b, c. Soit v_2 une variable CSP instanciée avec la valeur d et v_3 instanciée avec la valeur e. Supposons que $v_3 = e$ soit incompatible avec $v_1 = a$ et que $v_2 = d$ soit incompatible avec $v_1 = b$ et avec $v_1 = c$. La PHNF contient les clauses binaires qui correspondent à ces contraintes : $\neg a \vee \neg e, \neg b \vee \neg d$ et enfin $\neg c \vee \neg d$. Pour maintenir la consistance de chemin, il faut ajouter une contrainte pour interdire $v_2 = d$ et $v_3 = e$. Cela s'obtient en appliquant le principe de résolution aux clauses $a \vee b \vee c, \neg a \vee \neg e, \neg b \vee \neg d$ et $\neg c \vee \neg d$ pour produire $\neg d \vee \neg e$.

Pour assurer la k-consistance par propagation unitaire sur la PHNF, il est nécessaire d'être capable de produire tous les littéraux unitaires qui sont conséquences des impliqués décrits ci-dessus et de l'instanciation courante des variables. Pour atteindre ce but,il suffit d'effectuer une compilation logique :

Proposition 13: La k-consistance peut s'obtenir par propagation unitaire sur une formule f en *pigeon-hole normal form* quand f contient tous ses impliqués dérivés d'une clause $l_1 \vee l_2 \vee \ldots l_n$ en remplaçant par des étapes de résolution certains de ces littéraux par au plus $k-1$ littéraux négatifs issus d'au plus $k-1$ clauses.

Cette compilation logique garantit que, quelle que soit la manière dont le domaine d'une variable CSP est réduite, et pour toute instanciation de $k-1$ autres variables, soit la clause correspondant à la variable CSP peut encore être satisfaite ou bien la propagation unitaire sur la formule compilée aboutit à une contradiction.

Il est intéressant de noter que nous n'avons pas besoin de tous les impliqués décrits dans la proposition. En fait, tout ce qu'il faut c'est que la propagation unitaire soit complète pour la production de littéraux unitaires. Il est connu (voir par exemple [12]) que seules les résolvantes contenant un littéral fusionné sont nécessaires pour la complétude de la propagation unitaire.

Quand on s'intéresse uniquement à l'arc-consistance (2-consistance), la compilation logique à appliquer est particulièrement simple et efficace. En fait, pour garantir que la propagation unitaire va maintenir l'arc-consistance sur la PHNF, il suffit d'ajouter les impliqués obtenus à partir d'une clause positive $l_1 \vee l_2 \vee \ldots l_n$ en effectuant des résolutions avec le plus grand nombre possible de clauses binaires $\neg l_i \vee \neg l$ où $\neg l$ est un littéral commun à chacune de ces clauses binaires. On peut noter que s'il n'y a qu'une seule clause binaire de la forme $\neg l_i \vee \neg l$, l'impliqué généré est inutile puisque la résolvante ne contiendra pas de littéral fusionné.

Exemple 14: Soit A et B deux variables CSP de domaine $1, 2, 3$ et la contrainte $A < B$. Un codage direct de ce CSP en PHNF donne $a_1 \vee a_2 \vee a_3, b_1 \vee b_2 \vee b_3, \neg a_1 \vee \neg b_1, \neg a_2 \vee \neg b_1, \neg a_2 \vee \neg b_2, \neg a_3 \vee \neg b_1, \neg a_3 \vee \neg b_2, \neg a_3 \vee \neg b_3$. La propagation unitaire sur cette formule ne maintient pas l'arc-consistance (elle ne prouve pas $\neg a_3$ par exemple). La compilation logique que nous proposons ajoute deux clauses qui sont $\neg a_3, \neg a_2 \vee b_3$. Ce sont les seules résolvantes avec fusion dont nous ayons besoin. De ce fait, $\neg a_3$ est maintenant directement prouvé ce qui signifie que $A = 3$ n'a pas de support sur B.

[7] propose de maintenir l'arc-consistance sur une formule du problème SAT correspondant à un problème CSP en utilisant le *support encoding*. L'exemple donné ci-dessus correspond au cœur de l'exemple de ce papier. On peut remarquer que les clauses que nous ajoutons sont un sous-ensemble des clauses utilisées dans le *support encoding* ($b_2 \vee b_3 \vee \neg a_1$ ajouté dans [7] est inutile dans notre transformation parce que nous avons conservé les clauses binaires (clauses de conflits)). En fait, la compilation logique que nous proposons effectue le même travail que le *support encoding*. Cela donne un nouvel éclairage sur le *support encoding* qui n'apparaît plus comme un codage ad hoc pour assurer l'arc-consistance mais qui se révèle être une simple compilation logique pour rendre plus complète la propagation unitaire.

La compilation logique nécessaire pour assurer la chemin-consistance sur une formule PHNF n'est pas très difficile (comme on peut le voir sur l'exemple 12). Comme il s'agit d'une compilation logique, on peut espérer que le maintient de la chemin-consistance sur la version compilée de la formule sera efficace, ce qui devra être vérifié par des expériences à venir.

9 Conclusion

Nous avons présenté dans cet article une autre transformation d'une formule propositionnelle du problème SAT en un problème CSP. La caractéristique majeure de cette transformation est qu'elle est basée sur la sémantique de la formule et non sur sa syntaxe comme dans le cas des autres conversions. De plus, le processus de conversion est polynomial et aboutit à la définition d'une nouvelle forme normale pour les formules propositionnelles nommée *pigeon-hole normal form* (PHNF). Nous pensons que la PHNF pourrait faciliter la détection de structures particulières dans un problème (par exemple la contrainte de cardinalité atmost(1,. . .) se manifeste en une clique de clauses binaires dans une PHNF). Une autre caractéristique de la transformation est qu'elle est paramétrée par une borne m. Cette borne permet de choisir combien de travail sera effectué dans la phase de pré-traitement. Elle permet aussi de choisir de manière graduelle entre une conversion quasiment purement syntaxique et une conversion sémantique qui peut aller jusqu'à la résolution complète du problème dans certains cas. Nous avons également donné une heuristique pour réduire la taille de la formule obtenue en ordonnant les clauses de manières à obtenir le moins de modèles locaux possibles. Enfin, nous avons présenté une compilation logique qui permet à la propagation unitaire de maintenir la k-consistance.

Dans notre esprit, cette conversion tisse un nouveau lien entre le problème SAT et le problème CSP. Parce que cette conversion est sémantique, elle nous paraît particulièrement pertinente. De plus, elle fournit à la communauté CSP une nouvelle source de benchmarks du monde réel en traduisant les benchmarks SAT existants (industriels ou académique) dans le formalisme CSP. Un point particulièrement excitant est que cette traduction est paramétrée par la borne m qui définit directement la taille maximum des domaines dans la représentation CSP.

Enfin, nous pensons que cette transformation ouvre de nouvelles perspectives concernant la collaboration de solveurs. En fait, un problème initial (qui peut être exprimé dans de multiples formalismes) pourrait être partitionné en sous-problèmes qui seraient résolus par des prouveurs spécialisés (par exemple Horn, cardinalités, pseudo-booléens,. . .). La PHNF serait alors un moyen d'agréger les résultats de ces solveurs spécialisés et un algorithme de type CSP permettrait alors d'obtenir la solution globale du problème.

Références

[1] Belaïd Benhamou, Lakhdar Saïs, and Pierre Siegel. Two proof procedures for a cardinality based language in propositional calculus. In *Proc. of STACS 1994*, volume 775 of *LNCS*, pages 71–82, Caen (France), February 1994. Springer Verlag.

[2] Hachemi Bennaceur. The satisfiability problem regarded as a constraint satisfaction problem. In Wolfgang Wahlster, editor, *Proc. of ECAI 1996*, pages 155–160, Budapest, Hungary, August 1996. John Wiley & Sons.

[3] Hachemi Bennaceur. A Comparison between SAT and CSP Techniques. *International Journal of Constraints*, 9 :123–138, 2004.

[4] Thierry Castell. *Consistance et déduction en logique propositionnelle*. Thèse de doctorat, Université Paul Sabatier, Toulouse, janvier 1997.

[5] Douglas D. Edwards. Research summary, December 1996. Available from http ://www.cs.cornell.edu/-home/selman/papers-ftp/96-sat-survey-edwards.pdf.

[6] Ian P. Gent, Patrick Prosser, and Toby Walsh. The extended literal encoding of SAT into CSP. Technical Report APES-73-2003, APES Research Group, November 2003. Available from http ://www.dcs.st-and.ac.uk/˜apes/apesreports.html.

[7] I.P. Gent. Arc consistency in SAT. In *Proc. of ECAI 2002*, pages 121–125. IOS Press, 2002.

[8] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39 :297–308, 1985.

[9] Alex Kean and George Tsiknis. An Incremental Method for Generating Prime Implicants/Implicates. *Journal of Symbolic Computation*, 9 :185–206, 1990.

[10] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22 :253–275, 1985.

[11] D. Le Berre and L. Simon. The essentials of the SAT 2003 competition. In *Proc. SAT 2003*, volume 2919 of *LNCS*, pages 452–467. Springer-Verlag, 2003.

[12] O. Roussel and Ph. Mathieu. A new method for knowledge compilation : the achievement by cycle search. In *Proc. of CADE-13*, pages 493–507, New Brunswick (NJ-USA), 1996.

[13] Olivier Roussel. *L'achèvement des bases de connaissances en calcul propositionnel et en calcul des prédicats*. PhD thesis, Université des Sciences et Technologies de Lille, France, Janvier 1997.

[14] Pierre Siegel. *Représentation et utilisation de la connaissance en calcul propositionnel*. PhD thesis, Université de Aix-Marseille II, Faculté des Sciences de Luminy, France, 1987. Thèse d'Etat.

[15] James R. Slagle, Chin Liang Chang, and Richard C.T. Lee. A New Algorithm for Generating Prime Implicants. *IEEE Transactions on Computers*, C-19(4) :304–310, April 1970.

[16] Toby Walsh. SAT v CSP. In *Proc. of CP-2000*, volume 1894 of *LNCS*, pages 441–456. Springer-Verlag, 2000.