Résolution étendue et largeur de réfutation de formules SAT

Nicolas Prcovic
LSIS - Aix-Marseille Universités
nicolas.prcovic@lsis.org

Résumé

On appelle résolution de largeur k, une restriction de la résolution interdisant de produire des résolvantes de taille supérieure ou égale à $k+1$. Nous montrons que la résolution étendue (ie, incorporant la règle d'extension) de largeur 3 des instances 3-SAT est complète. De plus, nous montrons que le problème des pigeons peut se résoudre en temps polynômial grâce à la résolution étendue de largeur 3. Au final, nous obtenons donc un système de preuve dont la combinatoire est fortement réduite mais qui reste complet et plus efficace que la résolution sur certains problèmes.

Abstract

We call k-width Resolution, a restriction of Resolution forbiding the production of resolvents of size greater than or equal to $k+1$. We show that 3-width Extended Resolution (ie, Resolution incorporating the extension rule) on 3-SAT instances is complete. Furthermore, we show that the pigeonhole problem can be solved in polynomial time by 3-width Extended Resolution. We thus obtain a proof system whose combinatorics is greatly reduced but remains complete and more powerful than Resolution on some problems.

1 Introduction

Les systèmes de preuve basés sur la résolution sont utilisés pour permettre de trouver la réfutation d'une formule booléenne supposée insatisfiable. Cependant, certaines familles d'instances [7, 13, 4] nécessitent de produire un nombre exponentiel de résolvantes. Dans ces cas-là, beaucoup de résolvantes produites ont beaucoup de littéraux. La résolution étendue [12] ajoute une régle supplémentaire à la résolution, consistant à ajouter les trois clauses correspondant à $x \Leftrightarrow l_1 \vee l_2$ à la formule, où seule x est une nouvelle variable. Dans ce cadre, Cook [5] exhibe une réfutation de longueur polynômiale du problème des pigeons. Personne n'a jamais trouvé d'instances nécessitant la production d'un nombre exponentiel de résolvantes avec la résolution étendue. Mais bien que rendant un système de preuve par résolution plus puissant, la règle d'extension introduit une source supplémentaire d'explosion combinatoire. En effet, le nombre potentiel de résolvantes que l'on peut produire n'est plus une exponentielle du nombre de variables de la formule mais du nombre total de variables, qui inclut les variables ajoutées par les extensions. Or, le nombre d'extensions que l'on peut faire n'est pas nécessairement polynômial. C'est pourquoi les rares systèmes l'incluant (GUNSAT [2], GlucosEr [1], [8]) restreignent beaucoup l'application de cette règle afin que son avantage, le possible racourcissement de la longueur de la réfutation, ne soit pas contrebalancé par son inconvénient, le traitement des clauses supplémentaires. Dans cet article, nous proposons un changement de perspective sur la question : plutôt que de limiter l'application de la règle d'extension, nous limitons l'application de la règle de résolution à la production de clauses de taille 3 au maximum. Ce qui légitime cette approche est que l'ajout d'extensions est fait pour réduire la longueur de la preuve, donc il faut supposer que la taille des résolvantes sera petite.

Après avoir rappelé quelques notions sur le problème SAT (section 2), donné les rapports entre longueur et largeur de preuve (section 3) et introduit la résolution étendue (section 4), nous allons démontrer que, malgré cette restriction sur la largeur des preuves, ce système de preuve reste complet (section 5) et qu'il permet de donner une réfutation de longueur polynômiale du problème des pigeons (section 6).

2 Notions préliminaires

Une *formule booléenne*, sous sa forme dite *CNF*, est un ensemble de clauses. Une clause est un ensemble de littéraux. Un littéral est soit une variable booléenne, soit sa négation. On définit var(l) comme étant la variable du littéral l. Les variables peuvent être affectées à la valeur vrai ou faux. Une clause représente une disjonction de l'ensemble de ses littéraux. Une formule SAT représente une conjonction de ses clauses. Un *modèle* d'une formule ϕ est une affectation de variables qui est telle que ϕ est vraie. Une formule n'ayant pas de modèle est dite *insatisfiable*. La clause vide, notée $\square$, est toujours fausse et si une formule la contient alors elle est insatisfiable.

Les systèmes formels de preuve propositionnelle permettent de dériver d'autres formules à partir d'une formule ϕ, grâce à des règles d'inférence. En particulier, le système de Robinson [11], que nous appellerons `Res`, permet de dériver des clauses induites à partir des clauses d'une formule ϕ grâce à la règle de *résolution* :

$$C_1, C_2 \vdash C_1 \backslash \{l\} \cup C_2 \backslash \{\bar{l}\}$$

La clause inférée par résolution de C_1 avec C_2 sera appelé *résolvante* sur var(l) de C_1 et C_2. L'intérêt de `Res` est qu'il permet d'établir des réfutations de formules (ie, des preuves de leur insatisfiabilité) dans la mesure où une formule est insatisfiable si et seulement si il existe une suite de résolutions qui dérive la clause vide. `Res` fonctionne par saturation de l'application de sa règle : si une formule est satisfiable, `Res` le détecte quand plus aucune résolvante non redondante ne peut être dérivée.

Une dérivation est une suite d'applications de la règle de résolution permettant de dériver une clause. Elle constitue la preuve que cette clause est une conséquence logique de la formule. Elle peut se représenter grâce à un arbre binaire dont les feuilles sont des clauses de ϕ, les nœuds internes sont des résolvantes et la racine est la clause dérivée. Une dérivation de la clause vide s'appelle une *réfutation*.

3 Longueur et largeur de preuve

On appelle *longueur d'une preuve*, le nombre de résolvantes qu'elle contient. On appelle *taille* d'une clause le nombre de littéraux qu'elle contient. On appelle *largeur d'une preuve* la taille de la plus grande clause apparaissant dans cette preuve.

Les liens entre longueur et largeur de preuve sont maintenant assez bien établis. Il est clair qu'une preuve étroite est forcément courte. Plus précisément, si la largeur de la preuve est k alors la preuve ne peut contenir que de l'ordre de $\Theta(n^k)$ résolvantes différentes, où n est le nombre de variables de la formule. Si k est une constante alors la longueur de la preuve est polynômialement bornée. Dans [3], les auteurs montrent complémentairement que les preuves courtes sont forcément étroites. A partir de cette relation entre largeur et longueur de preuve, il est possible de définir des algorithmes de recherche de réfutations qui restreignent la largeur des résolvantes produites. Par exemple, le pré-traitement de Satz[10] (avant une résolution complète à la DPLL) consiste à dériver toutes les résolvantes possibles de taille 3 au maximum jusqu'à saturation (ou dérivation éventuelle de la clause vide). Plus généralement, une manière de garantir une complexité polynômiale consiste à restreindre la résolution à la production de résolvantes de taille inférieure à une borne donnée. Evidemment, cela se fait au détriment de la complétude dans la mesure où il est parfois nécessaire de dériver des grandes clauses avant d'obtenir la clause vide. On peut aussi définir une méthode complète qui restreint la résolution à la dérivation de résolvantes de taille k maximum mais qui incrémente la valeur de k à chaque fois qu'il y a saturation [3]. Dans cet article, nous explorons une autre voie. Au lieu d'augmenter la valeur de k, que nous fixons une fois pour toute à 3, nous nous plaçons dans la cadre de la résolution étendue, qui va nous permettre de rajouter des clauses permettant de continuer à ne dériver que d'autres résolvantes de taille 3.

4 La résolution étendue

La résolution étendue [12] consiste à ajouter à `Res` la *règle d'extension* en plus de la règle de résolution. Nous appellerons `ER` ce nouveau système de preuve. En toute généralité, la règle d'extension permet d'ajouter (à l'ensemble des clauses) les clauses correspondant à la formule $x \Leftrightarrow F$, où x est une nouvelle variable et F est n'importe quelle formule portant sur des variables existant déjà. A partir de maintenant, nous distinguerons les nouvelles variables introduites par des extensions, dites *variables d'extension*, des variables de la formule initiale, dites *variables d'entrée*.

On peut toujours se restreindre à ce que F soit uniquement du type $l_1 \vee l_2$ où l_1 et l_2 sont des littéraux portant sur des variables déjà présentes. En effet, il est facile de décomposer récursivement n'importe quelle formule F en une composition de disjonctions de sous-formules, éventuellement négatives. En associant chaque sous-formule à une nouvelle variable, on obtient un ensemble d'extensions de type $x_i \Leftrightarrow (l_1 \vee l_2)$ au lieu d'une seule du type $x \Leftrightarrow F$. Ce type d'extension portant sur la disjonction de deux littéraux préexistant sera appelée *extension binaire disjonctive*. Une extension $x \Leftrightarrow (l_1 \vee l_2)$ consiste à ajouter les clauses

$\overline{x} \vee l_1 \vee l_2$, $x \vee \overline{l_1}$ et $x \vee \overline{l_2}$. Il faut noter qu'il est aussi parfaitement possible de se restreindre à des extensions binaires conjonctives (i.e., de type $x \Leftrightarrow (l_1 \wedge l_2)$, qui consiste à ajouter les clauses $x \vee \overline{l_1} \vee \overline{l_2}$, $\overline{x} \vee l_1$ et $\overline{x} \vee l_2$).

L'intérêt de rajouter la règle d'extension est qu'elle rend le système plus puissant dans la mesure où certains problèmes dont la preuve est de longueur nécessairement exponentielle dans `Res` ont une preuve polynômiale dans `ER`. C'est le cas du fameux problème des pigeons [7, 5]. En dehors de ce problème particulier, on peut identifier des cas plus généraux où une extension permet un racourcissement de preuve. Considèrons une preuve contenant des séquences de résolutions qui font que $C_1 \vee C$ est utilisée pour dériver $C_1 \vee C'$ et que $C_2 \vee C$ est utilisée pour dériver $C_2 \vee C'$ grâce à la même suite de clauses qui permet de passer de C à C' (C_1, C_2, C et C' sont des clauses). On peut vérifier que grâce aux clauses émanant de $x \Leftrightarrow C_1 \wedge C_2$, on dérive d'abord $x \vee C$ à partir de $C_1 \vee C$ et $C_2 \vee C$, puis on effectue *une seule fois* la séquence de résolutions sur C permettant de dériver $x \vee C'$, à partir de quoi on peut à nouveau dériver $C_1 \vee C'$ et $C_2 \vee C$ (encore grâce aux clauses émanant de la même extension). Cette idée est mise en œuvre explicitement dans [1], que les auteurs appellent *compression de preuve* (et implicitement dans [8]) lors de la phase d'apprentissage d'une clause dans un solveur CDCL. Dans cet article, nous nous plaçons dans le cadre de la résolution étendue à l'aide d'extensions binaires disjonctives, qui permet un autre type de transformation de preuve, comme nous le verrons dans la section suivante.

Une question est de savoir quelles extensions il est intéressant d'effectuer afin de réduire la longueur d'une preuve. Mais une question encore plus cruciale est de savoir quelles extensions il est toujours inutile d'effectuer. Nous abordons ce point maintenant.

Définition 1 *Développement d'une extension*
On appelle développement d'une extension $x \Leftrightarrow F$, *l'expression d'une extension dans laquelle on remplace récursivement chaque occurence de variable d'extension de F par la disjonction binaire de l'extension qui l'a introduite, jusqu'à ce que F ne contiennent plus que des littéraux portant sur des variables d'entrée.*
Le développement CNF d'une extension $x \Leftrightarrow F$ *consiste à la mise sous forme CNF du développement de F.*

Exemple : étant données les extensions $x \Leftrightarrow (a \vee b)$ et $y \Leftrightarrow (\bar{x} \vee c)$, le développement de $z \Leftrightarrow (\bar{y} \vee d)$ est $z \Leftrightarrow (\overline{a \vee b} \vee c \vee d)$ et son développement CNF est $z \Leftrightarrow ((a \vee b \vee d) \wedge (\bar{c} \vee d))$.

Définition 2 *Niveau d'une variable, d'une extension*
Etant donnée une formule ϕ, on définit récursivement le niveau *N(x) d'une variable x ainsi :*
- *Si x est une variable d'entrée, N(x) = 0.*
- *Soit x la variable introduite par l'extension $x \Leftrightarrow (l_1 \vee l_2)$. On a N(x) = max(N(var($l_1$)), N(var($l_2$))) + 1.*

Le niveau d'une extension *est égal au niveau de la variable qu'elle introduit.*

Proposition 1 *n étant le nombre de variables d'entrée, si le niveau d'une extension est supérieur à $n \log n$ alors il existe une extension dont le développement CNF est identique mais dont le niveau est inférieur.*

Preuve 1 *On peut construire n'importe quel développement CNF contenant n variables comme composition récursive de disjonctions binaires. Pour toute formule CNF à n variables, chacune de ses clauses contient au plus n littéraux et la formule contient au plus 2^n clauses. Par dichotomie, on peut exprimer n'importe quel clause de taille n grâce à une composition récursive de clauses binaires. La hauteur de cette décomposition est $\log n$. De même, par dichotomie, on peut représenter un ensemble de 2^n clauses en une composition récursive de conjonctions binaires de clauses. La négation de cette composition récursive est une composition récursive de disjonctions de négations de clauses. La hauteur de cette composition récursive est n. Donc en tout, la hauteur maximale de décomposition disjonctive récursive d'une formule CNF est $n \log n$. Comme on peut associer une variable d'extension à chaque disjonction binaire représentant une sous-formule, une extension dont le développement CNF porte sur n variables ne nécessite qu'un niveau de $n \log n$ au maximum.*

Puisqu'on peut se limiter à n'ajouter que des extensions de niveau inférieur à $n \log n$, le nombre d'extensions possibles non redondantes (ie, dont le développement CNF diffère de celui des autres extensions) est borné.

5 Résolution étendue et largeur de réfutation restreinte

Bien que `ER` soit théoriquement plus puissant que `Res`, i.e. permet de trouver des preuves plus courtes, en pratique elle pose des difficultés qui l'ont empêché jusqu'à récemment d'être performante. Le choix des extensions à effectuer est difficile à faire. Et quand bien même les bons choix seraient faits, les résultats ne seraient pas automatiquement meilleurs. Par exemple, quand nous avons ajouté les $\Theta(n^3)$ clauses issues de la règle d'extension proposés par Cook aux $\Theta(n^2)$ clauses

du problème des n pigeons, nous avons expérimenté qu'aucun type de solveurs (qu'il soit "moderne", que ce soit DP60 [6] ou de la SL-résolution [9]) ne résolvait plus rapidement (bien au contraire) le problème alors que son temps de résolution devenait théoriquement polynômial. A moins d'avoir un bon moyen de choisir les résolvantes à produire, il y a le risque de produire à peu près les mêmes résolvantes qu'auparavant et beaucoup d'autres encore. Notre idée est donc simple : puisque nous rajoutons des extensions pour que la longueur de la preuve soit courte, nous devons supposer que la largeur de la preuve sera petite.

Le principe est de faire de la résolution étendue sans jamais produire de clause de taille supérieure ou égale à 4. Nous allons d'abord présenter sa réalisation à travers un exemple très simple. Soit la formule $\phi = \{a \vee b \vee c, \bar{a} \vee d \vee e, \bar{b} \vee d \vee e\}$. Pour obtenir la résolvante $c \vee d \vee e$, on est obligé de générer une résolvante quaternaire (ie, de taille 4) (cf le 1er arbre de la figure 1). Or, en ajoutant une extension $x \Leftrightarrow (d \vee e)$ ou $x \Leftrightarrow (c \vee d)$, on peut dériver $c \vee d \vee e$ uniquement via des résolvantes de taille 3 (cf les deux derniers arbres de la figure 1).

Plus fondamentalement, l'idée est que lorsqu'une résolution entre deux clauses ternaires $a \vee b \vee c$ et $\bar{c} \vee d \vee e$ génère une clause quaternaire $a \vee b \vee d \vee e$, on va faire en sorte de se ramener à une dérivation de clause ternaire. Soit $l_1, l_2 \in \{a, b, d, e\}$ et $\{l_3, l_4\} = \{a, b, d, e\} \setminus \{l_1, l_2\}$. Grâce aux clauses $x \vee \bar{l_1}$ et $x \vee \bar{l_2}$ de l'extension $x \Leftrightarrow (l_1 \vee l_2)$, nous dérivons la clause devenue ternaire $x \vee l_3 \vee l_4$ car $l_1 \vee l_2$ a été remplacé par x. Ensuite, nous attendons qu'on ait dérivé une clause binaire $x \vee l$ à partir de $x \vee l_3 \vee l_4$ pour pouvoir remplacer x par $l_1 \vee l_2$ (une fois qu'il ne peut plus produire de clause quaternaire) : la résolution entre $x \vee l$ et $\bar{x} \vee l_1 \vee l_2$ produit alors $l \vee l_1 \vee l_2$.

La figure 2 montre les deux types de dérivations possibles : celle où l_1 et l_2 sont dans la même clause (en haut) et celle où l_1 et l_2 sont dans deux clauses différentes (en bas).

Nous verrons qu'il est toujours possible d'éviter de produire des clauses quaternaire de cette façon tout en restant complet. Mais ceci n'est d'abord possible que si les formules CNF que nous cherchons à réfuter sont des instances 3-SAT (i.e., ne contiennent que des clauses de taille 3). Dans le cas contraire, il est toujours possible de ramener une formule CNF à une instance 3-SAT équivalente du point de vue de la satisfiabilité. La technique standard consiste à remplacer une clause $C = a_1 \vee a_2 \vee ... \vee a_k$ par $k-2$ clauses ternaires $a_1 \vee a_2 \vee \overline{x_2}$, $x_2 \vee a_3 \vee \overline{x_3}$, ..., $x_{k-3} \vee a_{k-2} \vee \overline{x_{k-2}}$ et $x_{k-2} \vee a_{k-1} \vee a_k$, où les $k-3$ variables x_i sont nouvelles. Cela se fait simplement en ajoutant les extensions $x_2 \Leftrightarrow (a_1 \vee a_2)$ et $\forall i, 3 \leq i < k-1, x_i \Leftrightarrow (a_{i-1} \vee x_{i-1})$. En effet, il suffit

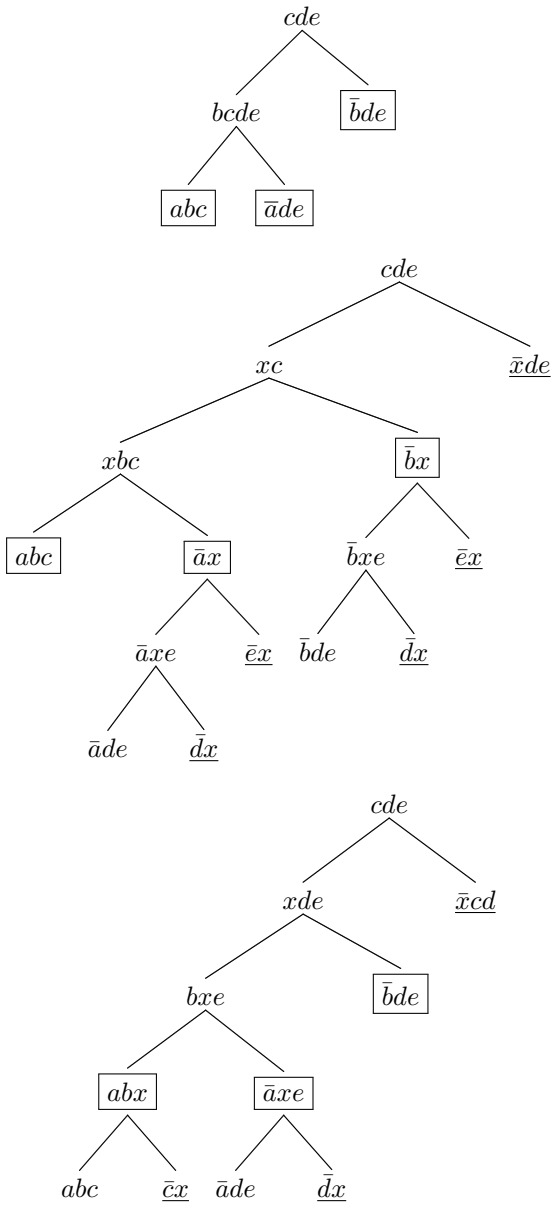

Fig. 1

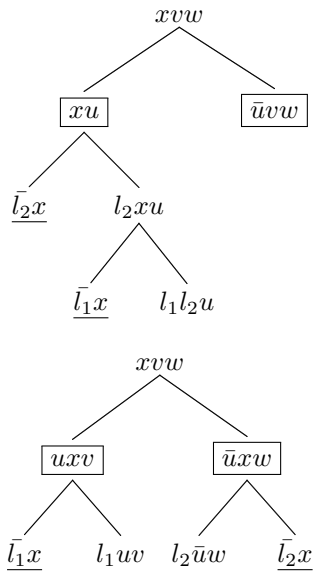

FIG. 2

de remarquer que les clauses ternaires ainsi ajoutées sont celles du type $x_i \vee a_{i+1} \vee \overline{x_{i+1}}$ qui remplacent C, tandis qu'en appliquant linéairement la suite de $2k-4$ résolutions entre les clauses binaires ajoutées et C, on dérive la dernière clause de remplacement $x_{k-2} \vee a_{k-1} \vee a_k$.

Définition 3 *Généalogie d'un littéral pour une résolvante*
Dans une preuve par résolution, la généalogie G d'un littéral x pour une résolvante C se définit inductivement ainsi :
- *$C \in G$.*
- *Si $C_1 \in G$ et que C_1 est la résolvante de C_2 et d'une autre clause et que x est un littéral de C_2 alors $C_2 \in G$.*

Définition 4 *Source de généalogie*
Dans une preuve par résolution, une source de généalogie G est une clause de G qui est aussi une clause d'entrée.

L'ensemble des sources d'une généalogie G d'une preuve faite à partir d'une formule ϕ est donc $G \cap \phi$.

Définition 5 *Amincissement de preuve arborescente*
L'amincissement d'une preuve (par résolution) arborescente, dont la racine et les feuilles sont de taille inférieure ou égale à 3 et dont au moins une des résolvantes est de taille 4, consiste en sa modification de la façon suivante. Considérons la racine de la preuve et remontons éventuellement l'un des chemins qui y mène jusqu'à trouver la première clause de taille 4, disons $a \vee b \vee c \vee d$. Sans perte de généralité, considérons que cette clause permet de produire la résolvante $b \vee c \vee d$ (avec une clause contenant $\overline{a}$ et éventuellement un certain nombre de littéraux parmi b, c et d). Considérons l'extension $x \Leftrightarrow (b \vee c)$ qui correspond aux clauses $x \vee \overline{b}$, $x \vee \overline{c}$ et $\overline{x} \vee b \vee c$. La modification de la preuve consiste à :
- *insérer une résolution entre $x \vee \overline{b}$ et chacune des sources de la généalogie de b pour la clause $b \vee c \vee d$. Ceci a pour effet de remplacer toutes les occurences de b par x dans toutes les clauses de la généalogie (à part les sources). En particulier, $b \vee c \vee d$ devient $x \vee c \vee d$.*
- *insérer une résolution entre $x \vee \overline{c}$ et chacune des sources de la généalogie de c pour la clause $x \vee c \vee d$. Ceci a pour effet de remplacer toutes les occurences de c par x dans toutes les clauses de la généalogie (à part les sources). En particulier, $x \vee c \vee d$ devient $x \vee d$ (les deux occurences de x fusionnent.*
- *insérer une résolution entre la résolvante $x \vee d$ et $\overline{x} \vee b \vee c$ pour obtenir à nouveau $b \vee c \vee d$.*

Un exemple d'amincissements se trouve en figure 3. Dans toutes les figures, nous soulignons les clauses qui sont ajoutées par les extensions.

Proposition 2 *Un amincissement de preuve arborescente contient au moins une clause de taille supérieure ou égale à 4 de moins qu'avant l'amincissement.*

Preuve 2 *Nous considérons l'amincissement tel qu'il est présenté dans la définition précédente. Il y a toujours une clause quaternaire $a \vee b \vee c \vee d$ s'il y a des clauses de taille supérieur à 4. En effet, une résolution ne peut produire une résolvante dont la taille est inférieure de plus d'une unité à la plus petite des deux clauses qu'on résout. Donc en partant d'une clause de taille supérieure à 4, on est obligé de passer par une clause de taille 4 pour aboutir à une clause de taille inférieure à 4.*

L'insertion des clauses $x \vee \overline{b}$ et $x \vee \overline{c}$ dans la preuve permet de renommer toutes les occurences de b et de c par x dans toutes les clauses (à part les sources) de la généalogie de a et b pour la clause $a \vee b \vee c \vee d$. Plus précisément, dans le cas où une clause contenait $b \vee c$, ce dernier est remplacé par x, ce qui réduit sa taille d'une unité. C'est le cas au moins de $a \vee b \vee c \vee d$ (de taille 4), qui devient $a \vee x \vee d$ (de taille 3). Par ailleurs, aucune autre clause de taille supérieure ou égale à 4 n'a pu apparaître car seules une clause ternaire et des clauses binaires ont été introduites, que l'effet de la clause ternaire a été de produire une clause ternaire tandis que l'effet des clauses binaires a été de renommer des littéraux.

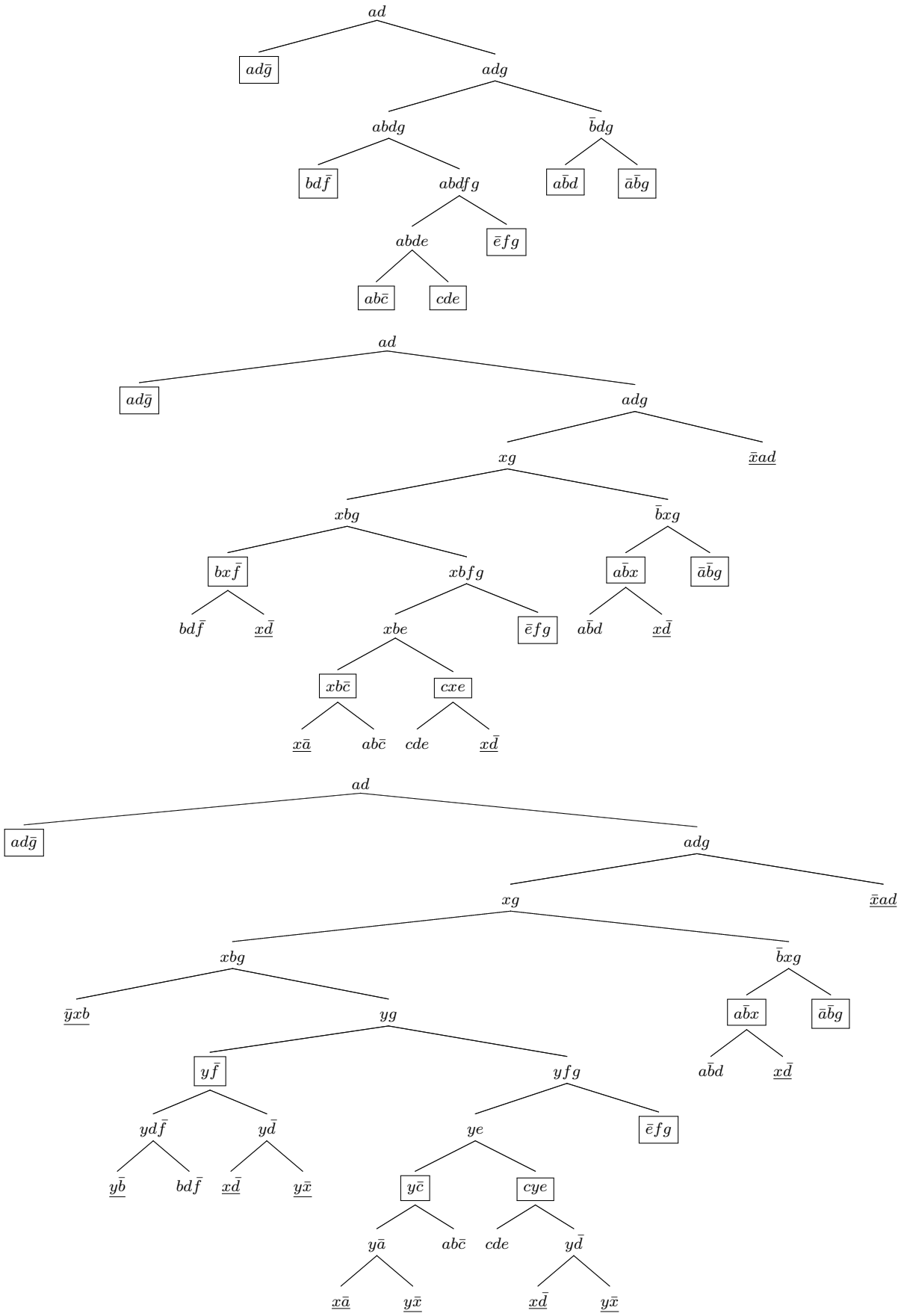

FIG. 3: Deux amincissements successifs d'une dérivation arborescente menant à une dérivation de largeur 3. Les deux extensions sont $x \Leftrightarrow (a \vee d)$ et $y \Leftrightarrow (x \vee b)$.

Proposition 3 *La résolution étendue de largeur 3 de formules 3-SAT est complète.*

Preuve 3 *Nous partons du fait qu'il existe une réfutation arborescente par résolution pour toute instance 3-SAT insatisfiable. D'après la proposition 2, un amincissement permet de décroître d'au moins une unité le nombre de résolvantes de taille supérieure ou égale à 4 d'une preuve dont la clause dérivée est de taille stritement inférieure à 4. En réitérant cette procédure d'amincissement, on élimine petit à petit toutes les résolvantes de taille 4 ou plus jusqu'à obtenir une réfutation de largeur 3.*

Nous appellerons `ER3` le système de preuve `ER` restreignant la dérivation aux résolvantes de taille 3 maximum.

Nous nous intéressons maintenant à la nouvelle longueur des preuves (i.e., le nombre de nœuds des arbres) obtenues après une suite d'amincissements.

Lorsqu'on effectue un premier amincissement grâce à une extension $x \Leftrightarrow (l_1 \vee l_2)$, on insère une résolution avec $\overline{x} \vee l_1 \vee l_2$ et un certain nombre de résolutions entre une clause binaire, $x \vee \overline{l_1}$ ou $x \vee \overline{l_2}$, et une feuille de l'arbre de preuve. Le second type de résolution remplace une feuille contenant une clause C de l'arbre par un sous-arbre composé d'une racine et de deux feuilles : C et la clause binaire. La racine de ce sous-arbre est la clause C dont un littéral (l_1 ou l_2) a été remplacé par x. Lors des amincissements suivants, ce sous-arbre pourra aussi être modifié par des insertions de résolutions avec des clauses binaires. Ce qui était initialement une feuille (avant le premier amincissement) est devenu un sous-arbre dont la racine contient la clause de cette feuille dont les littéraux ont été renommés par les insertions de résolutions.

Définition 6 *Sous-arbre de renommage*
Un sous-arbre de renommage *est un sous-arbre de preuve représentant la façon dont une feuille d'un arbre de preuve a eu ses littéraux renommés par une suite d'amincissements.*

Il faut remarquer qu'un littéral peut être renommé en un littéral déjà présent dans la même clause. Dans ce cas, les littéraux fusionnent et la clause diminue sa taille d'une unité. En conséquence, la clause étant à la racine d'un sous-arbre de renommage peut être plus courte que la clause initiale dont elle représente le résultat des renommages successifs. Dans toutes les figures, nous avons encadré les racines des arbres de renommages.

Proposition 4 *Lorsqu'une suite d'amincissements d'une preuve arborescente réduit sa largeur à 3, la longueur de cette preuve est inférieur à $(3L+1)L$, où L est la longueur de la preuve avant amincissements.*

Preuve 4 *Chaque amincissement dimininuant d'au moins une unité le nombre de résolvantes au moins quaternaires, le nombre d'amincissements à effectuer est inférieur à L, la longueur de la preuve (le nombre de nœuds de l'arbre) initiale. Le nombre global d'insertions de résolutions avec une clause ternaire de type $\overline{x} \vee l_1 \vee l_2$ est donc inférieur à L (car il y en a une par amincissement). Evaluons maintenant le nombre global d'insertions de résolutions avec des clauses binaires de type $x \vee \overline{l_1}$ (ou $x \vee \overline{l_2}$). Il est égal à la somme du nombre de nœuds des sous-arbres de renommage. Or, à chaque amincissement, chaque sous-arbre de renommage pourra éventuellement intégrer au maximum trois résolutions. Voici pourquoi.*
Au départ, un sous-arbre de renommage T ne contient qu'une feuille correspondant à une clause ternaire. Après un amincissement impliquant une extension $x \Leftrightarrow (l_1 \vee l_2)$, T n'est modifié que si sa racine contient l_1 (ou l_2). En effet, une résolution n'est inséré à partir d'une feuille de T que si c'est une source de généalogie de l_1 (ou l_2) et donc tous les nœuds sur le chemin entre cette feuille et le nœud contenant la clause de taille 4 qu'on veut amincir doivent contenir l_1 (ou l_2), ce qui doit donc être le cas de la racine de T. Si la racine de T contient à la fois l_1 et l_2 alors ces deux littéraux seront remplacés par une seule occurence de x et la taille de la clause à la racine sera décrémentée. Quel que soit le nombre d'amincissements, cela ne peut arriver que deux fois par sous-arbre de renommage : une première fois la clause ternaire à la racine devient binaire et une deuxième fois elle devient unaire. Examinons maintenant les cas où un seul des littéraux parmi l_1 et l_2 est présent dans la racine de T, disons l_1. Trois cas sont possibles :

- *La clause de la racine de T est ternaire (elle n'a jamais été réduite). Montrons par récurrence qu'un littéral ne peut avoir qu'une seule occurence parmi l'ensemble des littéraux des feuilles de T. Au début, T ne contenait qu'une seule clause, dont tous les littéraux étaient différents. Ensuite, à chaque amincissement, au pire une clause binaire $y \vee \overline{l}$ était insérée une seule fois car seule une feuille de T pouvait contenir l. Après insertion, il y a une seule occurence de y (nouvelle variable) et une seule occurence de $\overline{l}$ car il n'y en avait pas avant (sinon, il aurait fallu qu'il y ait une deuxième occurence de l dans une clause qui aurait fait disparaître $\overline{l}$ par résolution, car la racine de T ne contient pas $\overline{l}$).*
  Comme un littéral ne peut avoir qu'une seule occurence parmi l'ensemble des littéraux des feuilles de T, il ne peut y avoir au pire qu'une insertion de la clause binaire $x \vee \overline{l_1}$ dans T.
- *La clause de la racine de T est binaire (elle a*

été réduite une fois). Avant d'être réduite, chaque littéral de feuille n'avait qu'une occurence. Lorsqu'elle a été réduite par une insertion de $y \vee \overline{l}$ et une insertion de $y \vee \overline{l'}$, un seul littéral se trouve alors en double dans les feuilles de T : y. A partir de là, à chaque amincissement, il y a eu au pire deux insertions dans T, lorsque le renommage s'est fait sur y ou un de ses "renommeurs". Notons le fait que renommer y (ou un de ses "renommeurs") dans le sous-arbre de renommage l'empêche d'être renommé plus tard dans ce même sous-arbre car il a disparu de la racine donc il n'est plus source de généalogie. Donc, si l_1 est un des "renommeurs" de y, $x \vee \overline{l_1}$ est inséré deux fois dans T et x remplace l_1 dans la racine de T.

- *La clause de la racine de T est unaire (elle a été réduite deux fois). Avant d'être une deuxième fois réduite, chaque littéral de feuille avait au maximum deux occurences. Lorsqu'elle a été réduite une deuxième fois par une insertion de $y \vee \overline{l}$ et une insertion de $y \vee \overline{l'}$, la racine de T était $l \vee l'$. Au pire, l avait deux occurences dans les feuilles de T et l' une seule (ou l'inverse). En effet, l était un "renommeur" de la variable ayant été ajoutée deux fois lors de la première réduction de la racine de T contrairement à l' (sinon il y aurait eu deux réductions). Il y a donc eu en tout au maximum deux insertions de $y \vee \overline{l}$ et une insertion de $y \vee \overline{l'}$ dans T, lors de la deuxième réduction. Seul le littéral y a trois occurences dans les feuilles de T. Ensuite, à chaque amincissement, trois clauses (ou aucune) sont insérées dans T et la clause unaire de la racine est remplacée par un "renommeur" de y. Donc, dans notre cas, la racine de T est la clause l_1, la clause $x \vee \overline{l_1}$ est insérée trois fois dans T et la racine de T devient x.*

Puisqu'à chaque amincissement, une clause ternaire est inséré dans l'arbre de preuve et qu'au maximum 3 clauses binaires sont insérés dans chaque sous-arbre de renommage, qu'il y a moins de L sous-arbres de renommage et moins de L amincissements, à la fin de la série d'amincissements qui permet à l'arbre de preuve d'avoir une largeur 3, la nouvelle longueur de la preuve sera inférieure à (3L+1)L.

Une conséquence est que si une preuve arborescente est de longueur polynômiale alors elle reste polynômiale après être amincie jusqu'à avoir 3 de largeur. Ceci va nous aider à montrer qu'il existe une preuve polynômiale du problème des pigeons dans ER3.

6 Preuve polynômiale pour les pigeons

Tout d'abord, nous allons rappeler la preuve polynômiale dans ER du problème des pigeons qu'avait donné Cook [5] (sans qu'il ne l'ait complètement explicité).

Le problème des pigeons PHP(n) consiste à mettre n pigeons dans $n-1$ trous. P_{ij} signifie "le pigeon i est dans le trou j". La liste des clauses est :

- $\forall i,\ 1 \leq i \leq n,$: $P_{i1} \vee \ldots \vee P_{i,n-1}$
- $\forall i, j, k,\ 1 \leq k \leq n-1,\ 1 \leq i < j \leq n$: $\overline{P_{ik}} \vee \overline{P_{jk}}$

Voici le principe de la preuve par résolution étendue tel qu'il apparaît dans l'article de Cook. On va exprimer la satisfiabilité de PHP(n) en fonction de la satisfiabilité de PHP($n-1$). L'idée est qu'on peut obtenir une solution de PHP($n-1$) à partir d'une solution de PHP(n) en gardant les pigeons dans les mêmes trous à part que le pigeon qui se trouvait dans le trou $n-1$ (qui n'existe pas dans PHP($n-1$)) prend la place du pigeon n (qui n'existe pas dans PHP($n-1$)). Q_{ij} signifie "le pigeon i est dans le trou j" pour PHP($n-1$). On a donc les relations suivantes entre les P_{ij} et les Q_{ij} : $\forall i, j,\ 1 \leq i \leq n-1,\ 1 \leq j \leq n-2$: $Q_{ij} \Leftrightarrow (P_{ij} \vee (P_{i,n-1} \wedge P_{nj}))$. Remarquons que cette extension ternaire peut être exprimée grâce à deux extensions binaires disjonctives : $Q_{ij} \Leftrightarrow (P_{ij} \vee \overline{R_{ij}})$ et $R_{ij} \Leftrightarrow (\overline{P_{i,n-1}} \vee \overline{P_{nj}})$ mais nous garderons la formulation de Cook par simplicité. Sous forme CNF, on obtient donc les clauses :
(1) $Q_{ij} \vee \overline{P_{ij}}$,
(2) $Q_{ij} \vee \overline{P_{i,n-1}} \vee \overline{P_{nj}}$,
(3) $\overline{Q_{ij}} \vee P_{ij} \vee P_{i,n-1}$ et
(4) $\overline{Q_{ij}} \vee P_{ij} \vee P_{nj}$.
Ce sont les clauses d'extensions qu'on ajoute à PHP(n). Il s'avère qu'en effectuant un nombre polynômial (en $\Theta(n^3)$) de résolutions, on génère les clauses CNF de PHP($n-1$) à partir de celles de PHP(n) et des clauses d'extensions. On se ramène donc en un temps polynômial de PHP(n) à PHP($n-1$). Il suffit de réitérer le processus de manière récursive (ajouter de nouvelles clauses d'extensions pour se ramener à chaque fois à un problème de taille juste inférieur) pour aboutir à la génération (en temps polynômial) des clauses de PHP(2) qui ne nécessitent que 3 résolutions pour obtenir la clause vide.

Voici maintenant le détail de la résolution avec clauses d'extensions non explicités dans l'article de Cook.

La figure 4 indique comment on obtient chaque clause $\overline{Q_{ik}} \vee \overline{Q_{jk}}$ en sept résolutions.

Voici comment on obtient chaque clause $Q_{i1} \vee \ldots \vee Q_{i,n-2}$ en $2n-2$ résolutions. On note C_j la clause $Q_{i1} \vee \ldots \vee Q_{ij} \vee P_{i,j+1} \vee \ldots \vee P_{i,n-2}$ et D_j la clause $Q_{i1} \vee \ldots \vee Q_{ij} \vee P_{n,j+1} \vee \ldots \vee P_{n,n-2} \vee \overline{P_{i,n-1}}$. $\forall j$, C_j se génère par résolution entre C_{j-1} et $Q_{ij} \vee \overline{P_{ij}}$.

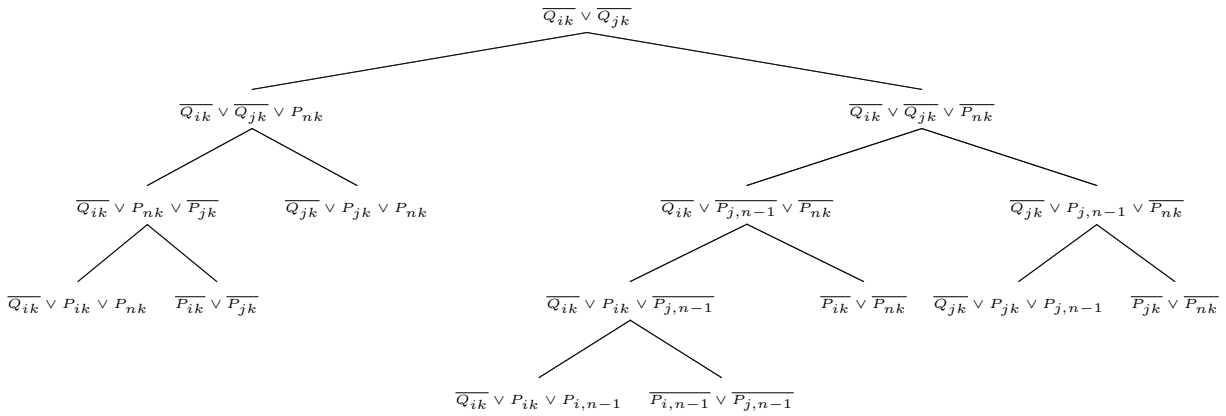

FIG. 4

Itérativement, on part de $P_{i1} \vee \ldots \vee P_{i,n-1} = C_0$ et on finit par générer $C_{n-2} = Q_{i1} \vee \ldots \vee Q_{i,n-2} \vee P_{i,n-1}$. $\forall j$, D_j se génère par résolution entre D_{j-1} et $Q_{ij} \vee \overline{P_{i,n-1}} \vee \overline{P_{nj}}$. Par résolution entre $P_{n1} \vee \ldots \vee P_{n,n-1}$ et $\overline{P_{i,n-1}} \vee \overline{P_{n,n-1}}$, on obtient D_0. Puis, itérativement, on finit par générer $D_{n-2} = Q_{i1} \vee \ldots \vee Q_{i,n-2} \vee \overline{P_{i,n-1}}$. Enfin, par résolution entre C_{n-2} et D_{n-2}, on obtient $Q_{i1} \vee \ldots \vee Q_{i,n-2}$.

La version 3-SAT du problème des pigeons se différencie uniquement par ses clauses n-aires positives $P_{i,1} \vee \ldots \vee P_{i,n-1}$ qui sont remplacées par $n-3$ clauses ternaires $P_{i,1} \vee P_{i,2} \vee \overline{X_{i,2}}$, $X_{i,2} \vee P_{i,3} \vee \overline{X_{i,3}}$, ..., $X_{i,n-4} \vee P_{i,n-3} \vee \overline{X_{i,n-3}}$ et $X_{i,n-3} \vee P_{i,n-2} \vee P_{i,n-1}$.

Proposition 5 *La version 3-SAT du problème des pigeons se résout en temps polynômial dans* `ER3`.

Preuve 5 *D'abord, on peut remarquer que la preuve polynômiale de Cook est une composition de dérivations arborescentes. Les seules dérivations arborescentes qui ne sont pas de largeur 3 sont celles qui permettent de dériver les clauses $q_i = Q_{i1} \vee \ldots \vee Q_{i,n-2}$ à partir des clauses $p_i = P_{i1} \vee \ldots \vee P_{i,n-1}$ et $p_n = P_{n1} \vee \ldots \vee P_{n,n-1}$ (et d'autres clauses, binaires ou ternaires). Pour obtenir une preuve de largeur 3, il nous suffit donc de trouver les dérivations arborescentes qui vont dériver les clauses ternaires équivalentes à q_i ($Q_{i,1} \vee Q_{i,2} \vee \overline{Y_{i,2}}$, $Y_{i,2} \vee Q_{i,3} \vee \overline{Y_{i,3}}$, ..., $Y_{i,n-5} \vee Q_{i,n-4} \vee \overline{Y_{i,n-4}}$ et $Y_{i,n-4} \vee Q_{i,n-3} \vee Q_{i,n-2}$) à partir des clauses ternaires équivalentes à p_i, p_n et les mêmes autres clauses binaires et ternaires qu'auparavant. Pour ceci, nous construisons d'abord des dérivations de largeur non borné. Pour obtenir les clauses ternaires équivalentes à q_i à partir des clauses ternaires équivalentes à p_i et p_n :*

- *nous dérivons p_i et p_n (nous les reconstituons) à partir de leurs clauses ternaires équivalentes (en $n-2$ résolutions chacune),*
- *puis nous dérivons q_i comme auparavant,*
- *puis à partir de q_i et des extensions $Y_{i,2} \Leftrightarrow (Q_{i,1} \vee Q_{i,2})$ et $\forall j, 3 \leq j < n-4$, $Y_{i,j} \Leftrightarrow (Q_{i,j-1} \vee Y_{i,j-1})$, nous dérivons linéairement (en $2n-6$ résolutions) $Y_{i,n-4} \vee Q_{i,n-3} \vee Q_{i,n-2}$.*

Les autres clauses que nous voulons ($Q_{i,1} \vee Q_{i,2} \vee \overline{Y_{i,2}}$, $Y_{i,2} \vee Q_{i,3} \vee \overline{Y_{i,3}}$, ..., $Y_{i,n-5} \vee Q_{i,n-4} \vee \overline{Y_{i,n-4}}$) sont celles données par les extensions. Nous avons donc les dérivations arborescentes de longueur polynômiale que nous voulions, à part que leur largeur est en général supérieure à 3. Mais comme nous avons vu (proposition 4) que nous pouvons les amincir jusqu'à ce qu'elles soient de largeur 3 en n'augmentant leur longueur que de manière polynômiale, nous pouvons remplacer toutes les dérivations arborescentes trop larges par leur version amincie de largeur 3. Nous pouvons en conclure qu'il existe bien une réfutation polynômiale du problème des pigeons dans `ER3`.

Le cas du problème des pigeons peut se généraliser à tout problème exponentiel dans `Res`, polynômial dans `ER` et dont la dérivation dans `ER` peut se décomposer en sous-dérivations arborescentes dont les racines et les feuilles contiennent toutes des clauses de taille 3 maximum. Par la même technique d'amincissements successifs de chaque sous-dérivation arborescente, on peut obtenir une dérivation globale de largeur 3 de longueur polynômiale.

Il existe donc des problèmes dont la réfutation est de longueur polynômiale dans `ER3` alors qu'elle est exponentielle dans `Res`. Cela signifie que la résolution étendue, même si on restreint la largeur de ses preuves, peut rester plus efficace sur certains problèmes que la résolution standard. Il n'est évidemment pas exclu qu'il existe d'autres problèmes pour lesquels le phénomène inverse soit vrai mais cela légitime qu'on considère `ER3` comme une alternative possible à `Res`.

Un schéma général d'algorithme pourrait être le suivant. En partant d'un ensemble B contenant toutes les clauses d'entrée d'une formule CNF, on complète B avec des résolvantes de taille 3 maximum jusqu'à saturation. Si la clause vide n'a pas été trouvée alors on ajoute à B les 3 clauses d'une extension binaire disjonctive non redondante et on recommence. Cet algorithme est complet car si la formule est satisfiable, il y a aura un moment où on ne pourra plus ajouter d'extensions non redondantes (on aura ajouté toutes les extensions possibles de niveau inférieur à $n \log n$) et on pourra alors conclure à la satisfiabilité de la formule.

Chaque phase de saturation par résolution peut produire de l'ordre de $\Theta((n+n')^3)$ résolvantes où n est le nombre de variables d'entrée et n' est le nombre actuel de variables d'extension. Celà est à comparer avec les $\Theta((n+n')^n)$ résolvantes potentielles qui pourraient être produites si on ne limitait pas la taille des résolvantes. Evidemment, n' n'est pas nécessairement borné par une fonction polynômiale. Par contre, si une formule n'a besoin qu'on lui ajoute qu'un nombre polynômialement borné d'extensions, cet algorithme est susceptible de lui trouver une preuve de longueur polynômiale en un temps polynômial, du moment que les bons choix d'extensions sont faits.

Conclusion et perspectives

En démontrant que l'on pouvait restreindre la résolution étendue à la dérivation de résolvantes ternaires tout en restant complet, nous avons diminué très fortement la combinatoire de ce système de preuve. En montrant que dans ce cadre le problème des pigeons reste réfutable en temps polynômial, nous avons établi que la résolution étendue de largeur 3 est une alternative possible à la résolution standard pour un certain nombre de problèmes dont la quantité reste à définir. Pour l'instant, une question théorique reste ouverte : `ER3` est-elle aussi puissante que `ER`, i.e. lorsque `ER` est capable de trouver une preuve de longueur polynômiale à un problème, `ER3` le peut-il toujours aussi ? En effet, le résultat de polynômialité sur la longueur des preuves amincies ne concerne que les arbres et pas toutes les preuves sous forme de DAG[1].

Dans cet article, nous n'avons pas abordé d'aspect pratique. Si on veut que la résolution étendue de largeur 3 soit efficace, il reste à trouver une bonne heuristique de choix d'extension dans ce contexte. C'est cet aspect qu'il faudra aussi maintenant explorer.

[1] Depuis la soumission de cet article, nous avons établi que `ER3` est aussi puissante que (i.e., p-simule) `ER`. Ce résultat n'étant pas encore formellement validé par la communauté suite à une procédure de relecture, nous ne l'avons pas intégré à la version finale de cet article.

Remerciements

Ce travail a été soutenu par un programme blanc de l'ANR (projet UNLOC).

Références

[1] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. In *AAAI*, 2010.

[2] Gilles Audemard and Laurent Simon. Gunsat : A greedy local search algorithm for unsatisfiability. In *IJCAI*, pages 2256–2261, 2007.

[3] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2) :149–169, 2001.

[4] Vasek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4) :759–768, 1988.

[5] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8 :28–32, 1976.

[6] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3) :201–215, 1960.

[7] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39 :297–308, 1985.

[8] Jinbo Huang. Extended clause learning. *Artif. Intell.*, 174(15) :1277–1284, 2010.

[9] Robert A. Kowalski and Donald Kuehner. Linear resolution with selection function. *Artif. Intell.*, 2(3/4) :227–260, 1971.

[10] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *IJCAI (1)*, pages 366–371, 1997.

[11] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1) :23–41, 1965.

[12] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logics*, pages 115–125, 1968.

[13] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1) :209–219, 1987.