Formulations MaxSAT pour la programmation non linéaire en nombres entiers

Zhifei Zheng, Sami Cherif, Rui Sá Shibasaki, Chu-Min Li, Jialu Zhang
Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

{zhifei.zheng;sami.cherif;rui.sa.shibasaki;chu-min.li;jialu.zhang}@u-picardie.fr

Résumé

Cet article présente de nouvelles formulations basées sur la Satisfiabilité Maximum (MaxSAT) pour encoder des problèmes de Programmation Non Linéaire en Nombres Entiers (NLIP) avec des fonctions polynomiales. Nous développons un cadre générique pour les fonctions polynomiales discrètes, basé sur trois techniques d'encodage d'entiers issues de la littérature. Notre approche traite tout d'abord chaque terme polynomial comme une unité atomique et démontre comment ces derniers peuvent être efficacement encodés grâce à des représentations compactes d'assignations d'entiers. De plus, nous présentons deux méthodes distinctes de décomposition basées sur les degrés des termes polynomiaux. Ces travaux ont donc le potentiel d'étendre l'applicabilité des solveurs MaxSAT à un éventail plus large de problèmes d'optimisation.

Mots-clés

Satisfiabilité Maximum, Programmation Non Linéaire en Nombres Entiers, Polynômes, Encodages

Abstract

This paper introduces novel Maximum Satisfiability (MaxSAT) formulations for encoding Nonlinear Integer Programming (NLIP) problems with polynomial functions. Specifically designed for discrete polynomial functions, we develop a generic framework based on three established integer encoding techniques from the literature. Our approach treats each polynomial term as an atomic unit and demonstrates how these can be efficiently encoded through compact representations of integer assignments. Additionally, we present two distinct decomposition methods based on polynomial term degrees. This preliminary work lays a foundation for future research and has the potential to extend the applicability of modern MaxSAT solvers to a wider range of optimization problems.

Keywords

Maximum Satisfiability, Nonlinear Integer Programming, Polynomials, Encodings

1 Introduction

La satisfiabilité propositionnelle (SAT) et la satisfiabilité maximum (MaxSAT) sont deux formalismes puissants dotés d'une grande expressivité en logique propositionnelle. Étant donné une formule propositionnelle sous forme d'un ensemble de clauses, le problème SAT consiste à déterminer si cette formule est satisfiable [7]. À son tour, MaxSAT consiste à maximiser le nombre de clauses satisfaites dans la formule donnée. Ces dernières années, MaxSAT a connu un essor important avec des applications dans de nombreux domaines, notamment la vérification matérielle et logicielle [18, 33], la planification et l'ordonnancement [10, 35] et la bio-informatique [16, 17]. Ces applications illustrent la manière dont MaxSAT parvient à réduire l'écart entre les variables booléennes, éléments atomiques fondamentaux de la logique propositionnelle, et les variables entières, davantage représentatives des problématiques issues du monde réel. Par ailleurs, l'introduction de contraintes pseudo-booléennes confère à MaxSAT la capacité de traiter des problèmes de programmation linéaire en nombres entiers. Néanmoins, le potentiel de MaxSAT dans un cadre non linéaire reste peu exploré dans la littérature. Cet article propose donc d'étudier la formulation de problèmes de programmation non linéaire en nombres entiers, exprimés à l'aide de fonctions polynomiales, à travers le prisme de la satisfiabilité maximum.

La programmation non linéaire en nombres entiers (Nonlinear Integer Programming, NLIP) constitue un problème fondamental, avec de nombreuses applications dans des domaines variés [21, 13, 29], puisque de nombreuses situations réelles s'expriment naturellement sur des domaines discrets au sein de structures polynomiales complexes. Cependant, NLIP demeure un problème notoirement difficile, tant sur le plan théorique que pratique. Cette difficulté provient principalement de la combinaison des contraintes d'intégralité et de la non-linéarité des expressions polynomiales. En effet, il a été démontré dans [27] que : (i) l'optimisation entière d'un polynôme quadratique dans une région polygonale convexe est un problème NP-difficile ; (ii) lorsque les variables ne sont pas bornées, même avec une fonction objectif linéaire et des contraintes polynomiales, le problème devient incalculable par une fonction récursive dès lors qu'il implique jusqu'à 10 variables entières. En outre, il a également été établi dans [22] que cette incalculabilité persiste pour des fonctions objectif linéaires soumises à des contraintes quadratiques, dès lors que les variables ne sont pas bornées. Ainsi, nous restreignons notre étude aux cas où les variables entières sont bornées.

Dans cet article, nous proposons des formulations Max-

SAT pour la programmation non linéaire en nombres entiers, fondées sur des représentations entières de la littérature. Tout d'abord, nous traitons chaque terme polynomial comme une unité atomique indépendante, pour laquelle nous élaborons des représentations booléennes. En particulier, nous développons trois formulations différentes basées sur les encodages one-hot, unaire et binaire. De plus, nous examinons les décompositions possibles de l'ordre des termes polynomiaux pour établir des formulations plus compactes. Les formulations proposées dans cet article mettent en évidence le potentiel de MaxSAT pour aborder une classe plus large de problèmes d'optimisation au-delà de leur domaine d'application traditionnel.

Le reste de cet article est organisé comme suit. La section 2 présente les définitions et notations fondamentales de MaxSAT et NLIP ; et passe en revue des travaux connexes de la littérature. La section 3 considère chaque terme polynomial comme une unité atomique, utilisant les trois représentations différentes pour encoder un NLIP donné. La section 4 se focalise sur la décomposition du degré des termes polynomiaux donnés, réduisant ainsi la complexité des encodages. La section 5 présente une étude de cas sur les polynômes d'ordre supérieur comme exemple illustratif. Enfin, nous concluons et discutons des travaux futurs dans la section 6.

2 Préliminaires

2.1 Satisfiabilté maximum

La satisfiabilité maximum (MaxSAT) est l'extension de la satisfiabilité (SAT) en problème d'optimisation. Dans cet article, nous utilisons sa version générique, à savoir MaxSAT partiel pondéré [24, 4]. Ce problème prend en entrée une formule bipartite pondérée $\phi = H \cup S$ contenant des variables booléennes dans X. H est l'ensemble des clauses dures qui doivent être satisfaites, comme dans SAT. S est l'ensemble des clauses souples, constituées de clauses pondérées (C, W_C), où W_C est un poids entier positif associé à la clause C. L'objectif de MaxSAT partiel pondéré est de trouver une affectation des variables α qui maximise (resp. minimise) la somme des poids des clauses souples satisfaites (resp. falsifiées) tout en satisfaisant toutes les clauses dures. Formellement, soit $cost_\alpha(\phi)$ la somme des poids des clauses souples dans ϕ falsifiées par l'affectation α, le problème MaxSAT partiel pondéré cherche à obtenir $optimum(\phi) = \min_\alpha cost_\alpha(\phi)$.

Dans la suite, nous utiliserons simplement MaxSAT pour désigner la version partielle pondérée définie ci-dessus. Il convient de noter que dans MaxSAT, les poids sont considérés comme des entiers positifs. Cependant, dans cet article, nous permettons que les poids soient négatifs, car ils peuvent toujours être normalisé en poids positifs. Enfin, nous mentionnons que la littérature introduit les contraintes Pseudo-Booléennes (PB) [32], qui peuvent être formalisées comme suit :

$$\sum_j a_j \cdot l_j \rhd b$$

tel que $a_j \in \mathbb{N}, b \in \mathbb{N}, l_j$ est un littéral et $\rhd \in \{=, \geq, \leq\}$.

2.2 Programmation non linéaire en nombres entiers

Dans cette section, nous définissons formellement les composants de la Programmation Non Linéaire en Nombres Entiers (NLIP) en se focalisant sur les fonctions polynomiales et nous introduisons les terminologies pertinentes utilisées dans l'ensemble de cet article. Comme mentionné dans l'introduction, notre étude se limite aux variables entières bornées inférieurement par 0 et supérieurement par une constante fixée ub. Le domaine des variables entières est donc donné par $V = \{0, 1, \ldots, ub\}$. De plus, nous restreignons les coefficients à l'ensemble des entiers $\mathbb{Z}$, car les applications réelles nécessitant une précision infinie sont extrêmement rares.

Définition 1 (Terme polynomial)**.** *Un terme $T(\mathbf{X})$ sur les variables $\mathbf{X} = (X_1, \ldots, X_m)$ est appelé terme polynomial s'il a la forme suivante :*

$$T(\mathbf{X}) = c \cdot \prod_{q=q_1}^{q_k} X_q$$

avec $c \in \mathbb{Z}$, $k \in \mathbb{N}$ et $q_1, \ldots, q_k \in \{1, \ldots, m\}$.

À noter que, dans la définition ci-dessus, les indices k ne sont pas nécessairement distincts. Nous appelons un terme contenant une seule variable (par exemple X_1^2) un *terme pur*, et un terme contenant plusieurs variables différentes (par exemple X_1X_2) un *terme croisé*. Nous appelons k l'ordre de $T(\mathbf{X})$. En particulier, lorsque $k \geq 2$, nous qualifions ce terme de *terme de grand ordre*. Lorsque le contexte est clair, nous abrégeons $T(\mathbf{X})$ en T. Sans perte de généralité, pour une formulation d'un terme polynomial à m variables et d'ordre k, nous utiliserons également une représentation équivalente $T(\mathbf{X}) = c \cdot \prod_{q=1}^{m} X_q^{k_q}$, où $\sum_{q=1}^{m} k_q = k$. En introduisant la notion de termes polynomiaux, nous pouvons maintenant formaliser les fonctions polynomiales (non linéaires) avant de présenter NLIP.

Définition 2 (Fonction polynomiale)**.** *Une fonction $P(\mathbf{X})$ est dite polynomiale si elle peut s'écrire comme la somme d'une série de n termes polynomiaux :*

$$P(\mathbf{X}) = \sum_{p=1}^{n} T_p(\mathbf{X})$$

où $n \in \mathbb{N}$ et $T_1(\mathbf{X}), \ldots, T_n(\mathbf{X})$ sont des termes polynomiaux.

Étant donné $P(\mathbf{X})$, s'il existe p tel que $T_p(\mathbf{X})$ est un terme de grand ordre, alors $P(\mathbf{X})$ est appelée une *fonction polynomiale non linéaire*.

Définition 3 (NLIP)**.** *Un modèle de programmation non linéaire en nombres entiers (NLIP), comportant m variables entières et $\mathcal{M}$ contraintes, peut être formalisé de la manière suivante [25, 20] :*

$$\begin{aligned} \min \quad & f(\mathbf{X}) && \text{(NLIP)} \\ \textit{s.t.} \quad & g_i(\mathbf{X}) \leq b_i && i = 1, \ldots, \mathcal{M} \\ & \mathbf{X} \in \mathbb{X},\ \mathbb{X} \subset \{0, \ldots, ub\}^m \end{aligned}$$

où au moins l'une des fonctions f ou g_i est non linéaire ; $\mathbf{X}$ est un vecteur dont chacun des m composants est une variable entière positive ou nulle, bornée supérieurement par ub.

Dans le cadre de cette étude, nous restreignons toutes les fonctions f et g_i à des fonctions polynomiales.

Exemple 1. *Ci-dessous, on donne un exemple de problème NLIP à deux variables :*

$$\begin{aligned} \min \quad & x_1^2 + x_1 x_2 - 2x_2 \\ \textit{s.t.} \quad & x_1^2 - x_2 \leq 3 \\ & x_1 + x_2 \leq 5 \qquad\qquad x_1, x_2 \in \{0,1,2,3,4\} \end{aligned}$$

2.3 Travaux connexes

Nous avons déjà présenté la définition de la programmation non linéaire en nombres entiers (NLIP) [25, 20]. Lorsque les contraintes entières imposées aux variables sont restreintes à des contraintes booléennes, c'est-à-dire lorsque toutes les variables ne peuvent prendre que les valeurs dans $\{0,1\}$, le problème NLIP se transforme alors en un problème de programmation non linéaire en 0-1, également connu sous le nom d'optimisation pseudo-booléenne (PBO) [5, 9]. Selon la nature des contraintes, PBO peut être affiné en plusieurs sous-domaines. Par exemple, on distingue le PBO quadratique, où toutes les contraintes sont de degré deux. Le problème MaxSAT pondéré partiel [24, 4] traite également de l'optimisation sur des variables booléennes, mais dans le cadre de la logique propositionnelle, où les contraintes sont exprimées sous forme de clauses dures et la fonction objectif est représentée par des clauses souples.

Beaucoup d'efforts ont été consacrés dans la littérature pour combler le fossé entre les différents types d'optimisation mathématique. Par exemple, [19] présente une méthode simple pour transformer tout problème de programmation entière en un problème de programmation entière 0-1, en remplaçant chaque variable entière par une série de variables booléennes représentant collectivement l'entier d'origine en format binaire. Plusieurs travaux [31, 12, 3, 30] introduisent également des réductions de PBO non linéaire vers des polynômes multilinéaires, ou encore vers PBO quadratique et linéaire. Enfin, plusieurs méthodes existent pour transformer les contraintes PB en clauses [14, 28, 1], ce qui fait le lien entre PBO et MaxSAT. Nous évoquons ici également les travaux connexes de [2], qui explorent la formulation des problèmes linéaires à domaines finis sous forme de problèmes SAT équivalents, en proposant six méthodes de mappage différentes. Cette étude peut également être vue comme une réflexion sur l'encodage des variables entières à l'aide de variables booléennes. Il est également important de noter que, comme le souligne [32], la programmation pseudo-booléenne est un domaine qui n'a pas été aussi développé que celui de la satisfiabilité, ce qui explique pourquoi nous développons notre travail à travers le prisme de MaxSAT.

Dans cet article, nous présentons donc une transformation de bout en bout de NLIP vers MaxSAT. Notre travail s'appuie sur des travaux antérieurs portant sur les représentations booléennes des entiers, les contraintes pseudo-booléennes ainsi que les techniques de décomposition, afin d'introduire un schéma de formulation systématique de NLIP à l'aide de MaxSAT.

2.4 Représentation des entiers avec MaxSAT

Dans cette section, nous présentons différents encodages des entiers à l'aide de variables booléennes. Plus précisément, étant donné une variable entière naturelle bornée $X \in \mathbb{N}$ telle que $X \leq ub$, où ub est une borne supérieure constante, une série de variables booléennes x_i est utilisée pour représenter X, selon l'une des trois approches couramment employées dans la littérature : la représentation one-hot [15, 23], la représentation unaire [6, 8] et la représentation binaire [14, 36]. Il convient de noter que les représentations one-hot, unaire et binaire correspondent respectivement aux encodages standard (Standard mapping), régulier (Full regular mapping), et logarithmique (Full logarithmic mapping), tels que définis dans [2].

Étant donné un entier naturel $X \leq ub$, les représentations one-hot et unaire utilisent toutes les deux $ub+1$ variables booléennes pour représenter X. Dans une représentation one-hot, seul le bit dont l'indice correspond à la valeur de X est assigné à 1, tandis que tous les autres sont assignés à 0. En revanche, dans la représentation unaire, tous les bits dont l'indice est inférieur ou égal à la valeur de X sont assignés à 1, les autres étant assignés à 0. Il est important de noter que, puisque $X \in \mathbb{N}$ et que 0 constitue une borne inférieure implicite, la variable x_0 est toujours fixée à 1. En ce qui concerne la représentation binaire, elle utilise $\lfloor \log_2 ub \rfloor + 1$ variables booléennes pour coder X. Les bits correspondent à ceux de l'écriture binaire (c'est-à-dire en base 2) de la valeur de X. Ainsi, les bits sont assignés à 1 s'ils correspondent à des positions actives dans la représentation binaire de la valeur attribuée à X, et à 0 dans le cas contraire.

Exemple 2. *Supposons que la borne supérieure de la variable entière soit $ub = 6$, et que la valeur obtenue soit $X = 5$. Dans les représentations one-hot et unaire, il faut $ub+1 = 7$ variables booléennes pour encoder cette valeur. On obtient alors l'affectation suivante dans la représentation one-hot : $x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 1, x_6 = 0$; et l'affectation suivante dans la représentation unaire : $x_0 = 1, x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1, x_6 = 0$. En revanche, la représentation binaire ne nécessite que $\lfloor \log_2 6 \rfloor + 1 = 3$ variables. Étant donné que $(5)_{10} = (101)_2$, l'affectation correspondante est : $x_2 = 1, x_1 = 0, x_0 = 1$.*

Par la suite, lors de la représentation de la variable entière $X_q \in \mathbf{X}$ ($q \in \{1, 2, \ldots, m\}$), nous désignons les variables booléennes associées par $x_r^{(q)}$, où r est l'identifiant de la variable. Nous récapitulons l'ensemble des notations et des identifiants utilisés dans le Tableau 1. Sauf indication contraire, log désignera le logarithme en base 2. En se basant sur les représentations entières introduites précédemment, nous abordons désormais la formulation d'un

TABLE 1 – Notations

Notation	Description
Ensembles	
$\mathbb{N}$	Ensemble des entiers naturels : $\{0, 1, 2, \ldots\}$
$\mathbb{Z}$	Ensemble des entiers relatifs : $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
Variables et fonctions	
X	Une variable entière
$\mathbf{X}$	Un vecteur de variables entières
$T(\mathbf{X})$	Un terme polynomial sur $\mathbf{X}$ (abrégé en T)
$P(\mathbf{X})$	Une fonction polynomiale non linéaire sur $\mathbf{X}$
x	Une variable booléenne correspondant à X
Indices	
$p, p_1, p_2, \ldots$	Indice des termes dans une fonction
$q, q_1, q_2, \ldots$	Indices des variables entières dans un terme
$r, r_1, r_2, \ldots$	Indices des variables booléennes correspondant à un entier

problème général d'optimisation sous contraintes par une reformulation en MaxSAT.
Un modèle MaxSAT partiel pondéré se compose de clauses souples pondérées et de clauses dures, ce qui correspond dans le cadre de NLIP respectivement à la fonction objectif et aux contraintes : les premières décrivent l'objectif à optimiser, tandis que les secondes définissent l'espace de solution. Plus précisément, si toutes les clauses souples sont des clauses unitaires contenant un seul littéral (ce qui peut toujours être obtenu en remplaçant toute clause non unitaire par un littéral, et en ajoutant des contraintes dures assurant leur cohérence), chacune de ces clauses peut être vue comme un terme de la fonction objectif, avec son poids représentant le coefficient associé. L'exemple 3 illustre une transformation simple d'un modèle de programmation linéaire en un modèle MaxSAT.

Exemple 3. *Le programme linéaire booléen suivant :*

$$\begin{aligned} max \quad & 2x + 3y \\ s.c \quad & x + 2y \leq 10 \\ & x, y \in \{0, 1\} \end{aligned}$$

peut être représenté avec le modèle MaxSAT suivant :

$$\begin{aligned} \textit{Clauses souples :} \quad & (x, 2), (y, 3) \\ \textit{Clauses dures :} \quad & x + 2y \leq 10 \textit{ (constrainte PB)} \end{aligned}$$

Ainsi, étant donné une instance de NLIP avec des variables dans $\mathbf{X}$, l'objectif de la procédure de formulation est de transformer la fonction objectif et les contraintes en clauses. Essentiellement, ces tâches peuvent être réduites à la représentation de chaque terme polynomial $T(\mathbf{X})$. Dans la suite, nous abordons cette question sous du point de vue des fonctions objectif, bien qu'une procédure identique puisse être appliquée à la transformation des contraintes.

3 Formulations atomiques des termes non linéaires

Dans cette section, nous nous focalisons sur les termes polynomiaux, considérés comme des unités atomiques. Nous les encodons à l'aide des trois représentations : one-hot, unaire et binaire.

3.1 Formulation one-hot

La représentation one-hot constitue la formulation la plus directe. Nous commençons par le cas le plus simple, à savoir la représentation d'un terme pur sous forme one-hot :

$$T_{pure} = c \cdot X^k = \sum_{r=0}^{ub} c \cdot r^k \cdot x_r$$

Pour transformer une terme croisé avec deux variables, on peut également procéder comme suit :

$$T_{bi-cross} = c \cdot X_{q_1}^{k_1} X_{q_2}^{k_2} = \sum_{r_1=0}^{ub} \sum_{r_2=0}^{ub} c \cdot r_1^{k_1} r_2^{k_2} x_{r_1}^{(q_1)} x_{r_2}^{(q_2)}$$

On remarque ici que cette équation contient des termes booléens non linéaires de la forme $x_{r_1}^{(q_1)} x_{r_2}^{(q_2)}$, ce qui constitue une difficulté systématique lorsqu'on traite un terme croisé. Toutefois, il est possible de linéariser ces produits à l'aide d'une nouvelle série de variables booléennes auxiliaires. Prenons l'exemple de $T_{bi\text{-}cross}$: on peut introduire des variables auxiliaires z_{r_1,r_2} $\forall (r_1, r_2) \in V^2$, où $V = \{0, \ldots, ub\}$, et il suffit de rajouter les contraintes auxiliaires suivantes :

$$z_{r_1,r_2} \leftrightarrow \left(x_{r_1}^{(q_1)} \wedge x_{r_2}^{(q_2)} \right) \quad \forall (r_1, r_2) \in V^2$$

En généralisant la procédure ci-dessus, on peut obtenir la formulation suivante pour tout terme polynomial T comportant m variables distinctes et d'ordre k :

$$T = c \cdot \prod_{q=1}^{m} X_q^{k_q} \qquad (\text{such that } \sum_{q=1}^{m} k_q = k)$$

$$= \sum_{r_1=0}^{ub} \sum_{r_2=0}^{ub} \cdots \sum_{r_m=0}^{ub} c \cdot \left(\prod_{q=1}^{m} r_q^{k_q}\right) \cdot z_{r_1,r_2,\ldots,r_m} \quad \text{(OH-1)}$$

$$z_{r_1,r_2,\ldots,r_m} \leftrightarrow \bigwedge_{q=1}^{m} x_{r_q}^{(q)} \quad \forall (r_1, r_2, \ldots, r_m) \in V^m \quad \text{(OH-2)}$$

$$\sum_{r \in V} x_r^{(q)} = 1 \quad \forall q \in \{q_1, \ldots, q_m\} \quad \text{(OH-3)}$$

Les équations OH-1 à OH-3 constituent ainsi la représentation one-hot pour tout terme polynomial. Lorsque T appartient à la fonction objectif donnée, on considère l'équation OH-1 comme les clauses souples suivantes :

$$\left(z_{r_1,r_2,\ldots,r_m}, c \cdot \left(\prod_{q=1}^{m} r_q^{k_q}\right)\right) \quad \forall (r_1, r_2, \ldots, r_m) \in V^m$$

tandis que OH-2 et OH-3 sont considérées comme contraintes dures.

Exemple 4. *Supposons $T = X_1^2 X_2$ et $ub = 3$. Nous utilisons des variables booléennes $x_r^{(1)}$ pour représenter X_1 et $x_r^{(2)}$ pour représenter X_2 en encodage one-hot. Nous introduisons d'abord les variables booléennes auxiliaires*

z_{r_1,r_2} *pour tout* $(r_1, r_2) \in V^2 = \{0, 1, 2, 3\}^2$ *et ajoutons les contraintes suivantes :*

$$\sum_{r_1 \in V} x_{r_1}^{(1)} = 1 \quad et \quad \sum_{r_2 \in V} x_{r_2}^{(2)} = 1$$

$$z_{r_1,r_2} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(2)}) \quad \forall (r_1, r_2) \in \{0, 1, 2, 3\}^2$$

Nous pouvons alors représenter T *comme suit :*

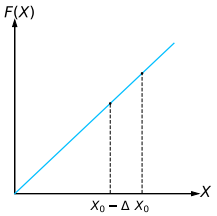
$$\begin{aligned} T = &(0 \times 0) \cdot z_{0,0} + (1 \times 0) \cdot z_{1,0} + (4 \times 0) \cdot z_{2,0} + (9 \times 0) \cdot z_{3,0} \\ &+ (0 \times 1) \cdot z_{0,1} + (1 \times 1) \cdot z_{1,1} + (4 \times 1) \cdot z_{2,1} + (9 \times 1) \cdot z_{3,1} \\ &+ (0 \times 2) \cdot z_{0,2} + (1 \times 2) \cdot z_{1,2} + (4 \times 2) \cdot z_{2,2} + (9 \times 2) \cdot z_{3,2} \\ &+ (0 \times 3) \cdot z_{0,3} + (1 \times 3) \cdot z_{1,3} + (4 \times 3) \cdot z_{2,3} + (9 \times 3) \cdot z_{3,3} \end{aligned}$$

L'avantage de la représentation one-hot réside dans sa simplicité. Cependant, pour encoder un terme comportant m variables et d'ordre k, elle nécessite un nombre de variables auxiliaires de l'ordre de $O(ub^m)$, ainsi que $O(ub^m)$ clauses pour l'encodage des contraintes (et jusqu'à $O(ub^k)$ dans le pire des cas où $k = m$).

3.2 Formulation unaire

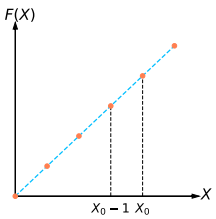
Dans cette section, on présente la formulation unaire, fondée sur la variation incrémentale de la valeur d'un terme lorsque les valeurs de ses variables changent. Cela correspond essentiellement à la notion de *différence* définie dans la suite. La théorie des différences remonte à l'application des méthodes de différences finies en analyse discrète [11, 34]. Les différences sont aux fonctions discrètes ce que les dérivées sont aux fonctions continues. De même, les sommes préfixées, définies ci-dessous, sont aux fonctions discrètes ce que les intégrales sont aux fonctions continues. L'analogie entre ces concepts est illustrée dans les figures 1 et 2, qui seront étayés plus en détail dans la Proposition 3.

Les différences offrent un cadre naturel pour décomposer les fonctions discrètes en contributions incrémentales, ce qui est particulièrement utile pour représenter les termes polynomiaux dans les formulations MaxSAT. En comprenant comment les valeurs des fonctions changent lorsque les variables augmentent d'une unité, nous pouvons encoder efficacement les variables entières à l'aide de variables booléennes dans une représentation unaire. Formellement, la *différence* d'un terme polynomial T est définie comme suit :



$$F'(X) = \lim_{\Delta \to 0} \frac{F(X) - F(X - \Delta)}{\Delta}$$
$$\int_0^{X_0} F'(X)dX = F(X_0) - F(0)$$

FIGURE 1 – Fonction continue



$$\widetilde{F}(X) = F(X) - F(X - 1)$$
$$\sum_{X=1}^{X_0} \widetilde{F}(X) = F(X_0) - F(0)$$

FIGURE 2 – Fonction discrète

Définition 4 (Différence d'un terme)**.** *Pour un terme polynomial* $T(\mathbf{X}) = c \cdot \prod_{q=1}^{m} X_q^{k_q}$*, sa différence* $\widetilde{T}$ *au point* $\mathbf{X} = (r_1, \ldots, r_m)$ *est définie par :*

$$\widetilde{T}(r_1, \ldots, r_m) = \begin{cases} 0 & si\ \exists q \in \{1, \ldots, m\}\ tel\ que\ r_q = 0; \\ \sum\limits_{S \subseteq \{1,\ldots,m\}} (-1)^{|S|} T(r'_1, \ldots, r'_m) & sinon \end{cases}$$

où $r'_q = r_q - 1$ *si* $q \in S$ *et* $r'_q = r_q$ *si* $q \in \{1, \ldots, m\} \setminus S$.

En se basant sur la définition de la différence et sur la structure des termes polynomiaux, la Proposition 1 décrit l'échelle de $\widetilde{T}$ et la Proposition 2 établit une forme inductive de la fonction de différence pour les termes polynomiaux, en fonction de leur degré.

Proposition 1. *Étant donné* $T(\mathbf{X})$ *d'ordre* k*, l'ordre de* $\widetilde{T}(\mathbf{X})$ *est inférieur à* k*.*

Preuve. *L'ensemble* $\{1, \ldots, m\}$ *a* 2^m *sous-ensembles distincts, y compris* 2^{m-1} *sous-ensembles de cardinalité impaire et* 2^{m-1} *sous-ensembles de cardinalité paire, par conséquent les coefficients des termes d'ordre le plus élevé s'annulent mutuellement dans la somme.*

Proposition 2. *Étant donné un terme polynomial* $T(\mathbf{X}) = c \cdot \prod_{q=1}^{m} r_q^{k_q}$ *et un point fixe* $\mathbf{X} = (r_1, \ldots, r_m)$ *tel que* $X_q \neq 0\ \forall q \in \{1, \ldots, m\}$*, on a :*

$$\widetilde{T}(X) = (r_m^{k_m} - (r_m - 1)^{k_m}) \cdot \widetilde{T}(r_1, \ldots, r_{m-1})$$

où $T(r_1, \ldots, r_{m-1}) = c \cdot \prod_{q=1}^{m-1} r_q^{k_q}$.

Preuve. *On peut démontrer ce résultat en raisonnant séparément sur les cas où* $m \in S$ *ou* $m \notin S$*, comme suit :*

$$\begin{aligned} &\widetilde{T}(r_1, \ldots, r_m) = \sum_{S \subseteq \{1,\ldots,m\}} (-1)^{|S|} T(r'_1, \ldots, r'_m) \\ =&\sum_{S \subseteq \{1,\ldots,m-1\}} (-1)^{|S|} T(r'_1, \ldots, r'_{m-1}) \cdot r_m^{k_m} \\ &+ \sum_{S \subseteq \{1,\ldots,m-1\}} (-1)^{|S|+1} T(r'_1, \ldots, r'_{m-1}) \cdot (r_m - 1)^{k_m} \\ =&\widetilde{T}(r_1, \ldots, r_{m-1}) \cdot r_m^{k_m} - \widetilde{T}(r_1, \ldots, r_{m-1}) \cdot (r_m - 1)^{k_m} \\ =&(r_m^{k_m} - (r_m - 1)^{k_m}) \cdot \widetilde{T}(r_1, \ldots, r_{m-1}) \end{aligned}$$

Cette formule récursive nous permet de calculer la fonction de différence pour des termes de grand ordre, et sera également utile dans les discussions ultérieures. Ayant défini la notion de différence, nous introduisons à présent celle de *somme cumulative*, reflétant l'analogie entre les dérivées et les intégrales dans le calcul différentiel continu. Cette construction sera utilisée pour reconstruire les termes polynomiaux à partir de leurs formes différentielles dans le cadre de l'encodage unaire.

Définition 5 (somme cumulative des différences d'un terme)**.** *La somme cumulative des différences d'un terme polynomial $T(\mathbf{X})$ en $\mathbf{X} = (X_1, \ldots, X_m)$ est définie par :*

$$PreSum_T(\mathbf{X}) = \sum_{r_1=1}^{X_1} \sum_{r_2=1}^{X_2} \cdots \sum_{r_m=1}^{X_m} \widetilde{T}(r_1, \ldots, r_m)$$

La proposition suivante révèle que la relation entre la *somme cumulative* et la *différence* est analogue à la relation entre l'intégrale et la dérivée.

Proposition 3. *Pour tout terme polynomial T et tout $\mathbf{X} = (X_1, \ldots, X_m)$, on a $PreSum_T(\mathbf{X}) = T(\mathbf{X})$.*

Preuve. *Cette proposition peut être démontrée par récurrence sur la dimension m de $\mathbf{X}$. Lorsque $m = 1$, on a $\widetilde{T}(X) = T(X) - T(X-1)$. Ainsi*

$$PreSum_T(X) = \sum_{r=1}^{X} (T(r) - T(r-1)) = T(X) - T(0) = T(X),$$

ce qui montre que la proposition est vérifiée. Supposons maintenant que la propriété soit vraie pour m, et montrons qu'elle l'est aussi pour $m+1$. Étant donné $\mathbf{X} = (X_1, \ldots, X_{m+1})$, on a :

$$\begin{aligned}
&PreSum_T(\mathbf{X}) = \sum_{r_1=1}^{X_1} \cdots \sum_{r_m=1}^{X_m} \left(\sum_{r_{m+1}=1}^{X_{m+1}} \widetilde{T}(r_1, \ldots, r_{m+1}) \right) \\
&= \sum_{r_1=1}^{X_1} \cdots \sum_{r_m=1}^{X_m} \left(\sum_{r_{m+1}=1}^{X_{m+1}} (r_{m+1}^{k_{m+1}} - (r_{m+1}-1)^{k_{m+1}}) \cdot \widetilde{T}(r_1, \ldots, r_m) \right) \\
&= \sum_{r_1=1}^{X_1} \cdots \sum_{r_m=1}^{X_m} \left(\widetilde{T}(r_1, \ldots, r_m) \cdot \sum_{r_{m+1}=1}^{X_{m+1}} (r_{m+1}^{k_{m+1}} - (r_{m+1}-1)^{k_{m+1}}) \right) \\
&= \left(\sum_{r_1=1}^{X_1} \cdots \sum_{r_m=1}^{X_m} \widetilde{T}(r_1, \ldots, r_m) \right) \cdot {X_{m+1}}^{k_{m+1}} \\
&= T(r_1, \ldots, r_m) \cdot {X_{m+1}}^{k_{m+1}} = T(r_1, \ldots, r_{m+1}) = T(\mathbf{X})
\end{aligned}$$

Sur la base des concepts de *différence* et de *somme cumulative*, nous pouvons développer la formulation MaxSAT fondée sur l'encodage unaire pour les termes polynomiaux. Commençons par le cas simple d'un terme pur contenant une seule variable entière X :

$$T_{\text{pure}}(X) = c \cdot X^k$$

Nous avons alors :

$$\widetilde{T}_{\text{pure}}(X = r) = T_{\text{pure}}(r) - T_{\text{pure}}(r-1) = c \cdot (r^k - (r-1)^k)$$

D'où, pour un X quelconque :

$$T_{\text{pure}} = \sum_{r=1}^{ub} \widetilde{T}(r) \cdot x_r = \sum_{r=1}^{ub} c \cdot (r^k - (r-1)^k) \cdot x_r$$

Un exemple correspondant est donné ci-dessous.

Exemple 5. *Soit $T = X^2$ un terme contenant une variable entière X bornée par $ub = 4$. On suppose que l'on utilise des variables booléennes x_r pour représenter X sous encodage unaire. Ainsi, on obtient $T(X) = 1x_1 + 3x_2 + 5x_3 + 7x_4$ comme indiqué dans Table 2.*

X	0	1	2	3	4
$T = X^2$	0	1	4	9	16
$\widetilde{T}$	-	1	3	5	7

TABLE 2 – Différence unidimensionnelle

La représentation unaire pour les termes purs repose sur la différence unidimensionnelle. Lorsqu'un terme polynomial contient deux variables différentes, on utilise alors la différence bidimensionnelle, comme illustré dans l'Exemple suivant.

Exemple 6. *Supposons que $T = X_1^2 X_2$ et que la borne supérieure soit $ub = 3$. Nous utilisons les variables booléennes $x_r^{(1)}$ pour représenter X_1 et $x_r^{(2)}$ pour représenter X_2 selon la représentation unaire. La valeur originale de T ainsi que les valeurs de la différence bidimensionnelle sont données respectivement dans les Tables 3 et 4.*

X_1 \ X_2	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	4	8	12
3	0	9	18	27

TABLE 3 – Valeurs du terme

X_1 \ X_2	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	3	3	3
3	0	5	5	5

TABLE 4 – Différence bidimensionnelle

Étant donné une paire de valeurs (X_1, X_2), la somme cumulative bidimensionnelle de la coordonnée correspondante dans la Table 4 donne exactement la valeur de T. Par exemple, si $X_1 = 3$ et $X_2 = 2$, alors :

$$T(X_1, X_2) = \sum_{r_1=1}^{X_1} \sum_{r_2=1}^{X_2} \widetilde{T}(r_1, r_2) = 1{+}1{+}3{+}3{+}5{+}5 = 18$$

On remarque que la représentation one-hot pour ce cas entraînerait un poids maximal des clauses souples allant jusqu'à 27, tandis que dans la représentation unaire, le poids maximal n'est que de 5.

En généralisant la procédure ci-dessus, on peut obtenir le modèle suivant pour tout terme polynomial T d'ordre k comportant m variables :

$$\begin{aligned}
T &= c \cdot \prod_{q=1}^{m} X_q^{k_q} \qquad \left(\text{tel que } \sum_{q=1}^{m} k_q = k\right) \\
&= \sum_{r_1=1}^{ub} \sum_{r_2=1}^{ub} \cdots \sum_{r_m=1}^{ub} \widetilde{T}(r_1, \ldots, r_m) z_{r_1, \ldots, r_m} \quad \text{(U-1)}
\end{aligned}$$

avec

$$z_{r_1, \ldots, r_m} \leftrightarrow \bigwedge_{q=1}^{m} x_{r_q}^{(q)} \quad \forall (r_1, \ldots, r_m) \in V^m \quad \text{(U-2)}$$

$$x_r^{(q)} \rightarrow x_{r-1}^{(q)} \quad \forall (q, r) \in N \times V \quad \text{(U-3)}$$

Les équations U-1 à U-3 constituent la représentation unaire de tout terme polynomial. Lorsque T est un élément de la fonction objectif donnée, on considère U-1 comme des contraintes souples, tandis que U-2 et U-3 sont considérées

comme des contraintes dures. La représentation unaire nécessite $O(ub^m)$ variables auxiliaires z et $O(ub^m)$ clauses pour l'encodage des contraintes (et $O(ub^k)$ dans le pire des cas où $k = m$) pour encoder un terme avec m variables et d'ordre k.

Dans la résolution de MaxSAT pondéré, la variance des poids des clauses souples constitue un facteur important influençant la difficulté du problème. Un travail récent [26] propose une approche innovante qui traite ces défis dans MaxSAT pondéré par un mécanisme de verrouillage et de déverrouillage partiel des poids souples. Ce mécanisme met également en évidence un problème majeur : lorsque les poids souples présentent de grandes disparités, le nombre de schémas de partitionnement possibles des poids augmente considérablement, ce qui complexifie l'espace de recherche.

Le principal avantage apporté par la représentation unaire réside donc dans la réduction de l'échelle des poids des clauses souples. Comme démontré dans la Proposition 1, la représentation unaire garantit un ordre de grandeur du poids souple ($O(ub^{k-1})$) toujours inférieur à celui obtenu par l'encodage one-hot ($O(ub^k)$).

3.3 Formulation binaire

Dans cette sous-section, nous présentons la représentation binaire applicable à tout terme polynomial. Nous explorons progressivement les cas des termes d'ordre un, d'ordre deux, puis des ordres supérieurs. Supposons qu'il existe un terme linéaire $T_{pure} = cX$ que l'on souhaite représenter de manière binaire. On aura ainsi :

$$T_{pure} = c \cdot X = \sum_{r \in \{0,\dots,\lfloor \log ub \rfloor\}} c \cdot 2^r x_r$$

La représentation binaire consiste donc à décomposer tout entier X comme la somme d'une série de puissances de 2, ce qui permet d'encoder un terme linéaire avec $O(\log ub)$ variables. Passons maintenant à un autre exemple où l'ordre est égal à 2. Étant donné $T = X_1 X_2$, la représentation binaire de ce terme peut être formulée comme suit :

$$\begin{aligned} X_1X_2 =& \Big(\sum_{r_1=0}^{\lfloor \log ub \rfloor} 2^{r_1} x_{r_1}\Big)\Big(\sum_{r_2=0}^{\lfloor \log ub \rfloor} 2^{r_2} x_{r_2}\Big) \\ =& \sum_{r_1=0}^{\lfloor \log ub \rfloor} \sum_{r_2=0}^{\lfloor \log ub \rfloor} 2^{r_1+r_2} x_{r_1} x_{r_2} \qquad \text{(B-qua-cross)} \end{aligned}$$

L'équation B-qua-cross peut être généralisée pour formuler tout terme polynomial T avec m variables différentes et d'ordre k. Dans cette formulation, nous commençons par reformuler $\prod_{q=1}^{m} X_q^{k_q}$ en $\prod_{q=q'_1}^{q'_k} X_q$, où l'ensemble $\{q'_1, q'_2, \dots, q'_k\}$ contient chaque indice de variable répété autant de fois que sa puissance dans le terme original.

$$T = c \cdot \prod_{q=1}^{m} X_q^{k_q} = c \cdot \prod_{q=q'_1}^{q'_k} X_q \qquad (\text{tel que } \sum_{1}^{m} k_q = k)$$

$$= \sum_{r_1=0}^{\lfloor \log_2 ub \rfloor} \sum_{r_2=0}^{\lfloor \log_2 ub \rfloor} \cdots \sum_{r_k=0}^{\lfloor \log_2 ub \rfloor} c \cdot \Big(\prod_{q=q'_1}^{q'_k} 2^{r_q}\Big) \cdot z_{r_1,r_2,\dots,r_k} \tag{B-1}$$

$$z_{r_1,\dots,r_k} \leftrightarrow \bigwedge_{q=q'_1}^{q'_k} x_{r_q}^{(q)} \quad \forall (r_1, \dots, r_k) \in \{0, \dots, \lceil \log ub \rceil\}^k \tag{B-2}$$

Les équations B-1 et B-2 constituent la représentation binaire de tout terme polynomial. Lorsque T est un élément de la fonction objectif considérée, l'équation B-1 est prise comme contrainte souple, tandis que l'équation B-2 est considérée comme contrainte dure.

Exemple 7. *Supposons que $T = X_1^2 X_2$ et que $ub = 3$. Nous utilisons des variables booléennes $x_r^{(1)}$ pour représenter X_1 et $x_r^{(2)}$ pour représenter X_2 dans l'encodage binaire. Comme $\lfloor \log ub \rfloor + 1 = \lfloor \log 3 \rfloor + 1 = 2$, nous avons besoin de 2 variables booléennes pour chaque variable entière, indexées par 0 et 1. La représentation binaire nécessite de réécrire $T = X_1 X_1 X_2$. Nous introduisons alors les variables booléennes auxiliaires z_{r_1,r_2,r_3} pour tout $(r_1, r_2, r_3) \in \{0,1\}^3$ et ajoutons les contraintes suivantes :*

$$z_{r_1,r_2,r_3} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(1)} \wedge x_{r_3}^{(2)}) \quad \forall (r_1, r_2, r_3) \in \{0,1\}^3$$

Nous pouvons alors représenter T comme suit :

$$\begin{aligned} T =& 2^0 \cdot z_{0,0,0} + 2^1 \cdot z_{0,0,1} + 2^1 \cdot z_{0,1,0} + 2^2 \cdot z_{0,1,1} \\ &+ 2^1 \cdot z_{1,0,0} + 2^2 \cdot z_{1,0,1} + 2^2 \cdot z_{1,1,0} + 2^3 \cdot z_{1,1,1} \end{aligned}$$

Par exemple, supposons que $X_1 = 3 = 2^1 + 2^0$ et que $X_2 = 2 = 2^1$, Ainsi, $x_1^{(1)} = 1$, $x_0^{(1)} = 1$, $x_1^{(2)} = 1$, $x_0^{(2)} = 0$ et donc on a :

$$\begin{aligned} T(X_1, X_2) =& \sum_{r_1=0}^{\lfloor \log_2 ub \rfloor} \sum_{r_2=0}^{\lfloor \log_2 ub \rfloor} \sum_{r_3=0}^{\lfloor \log_2 ub \rfloor} \Big(\prod_{q=1}^{3} 2^{r_q}\Big) \cdot z_{r_1,r_2,r_3} \\ =& 2^1 \cdot z_{0,0,1} + 2^2 \cdot z_{0,1,1} + 2^2 \cdot z_{1,0,1} + 2^3 \cdot z_{1,1,1} \\ =& 2 + 4 + 4 + 8 = 18 \end{aligned}$$

ce qui correspond bien à $3 \cdot 3 \cdot 2 = (1+2) \cdot (1+2) \cdot (0+2)$.

Nous notons ici une différence majeure des formulations one-hot et unaire. La formulation binaire met principalement l'accent sur le degré k, plutôt que sur le nombre de variables distinctes m impliquées dans l'expression, ce qui explique pourquoi elle commence par la reformulation de $\prod_{q=1}^{m} X_q^{k_q} = \prod_{q=q'_1}^{q'_k} X_q$. L'exemple suivant illustre la nécessité de cette approche.

Exemple 8. *Soit* $T(X) = X^2$. *On note que* $T(4) = 4^2 = 16$, *mais* $T(1+3) = 1^2 + 3^2 = 10 \neq 16$. *Ainsi, on peut seulement reformuler ce terme comme suit :*

$$X^2 = (\sum_{r=0}^{\lfloor \log ub \rfloor} 2^r x_r)^2 = \sum_{r=0}^{\lfloor \log ub \rfloor} 2^{2r} x_r + 2 \sum_{0 \leq r_1 < r_2 \leq \lfloor \log ub \rfloor} 2^{r_1+r_2} x_{r_1} x_{r_2} \quad \text{(B-qua-pure)}$$

L'Équation B-qua-pure peut être considérée comme un cas particulier de l'Équation B-qua-cross.

Pour encoder un terme avec m variables et un ordre k, la représentation binaire nécessite :

— $O((\log ub)^k)$ variables auxiliaires z

— $O((\log ub)^k)$ clauses pour encoder les contraintes.

Nous remarquons que, dans la représentation binaire, c'est l'ordre k qui affecte la complexité, contrairement aux représentations one-hot et unaire où c'est le nombre de variables distinctes m qui a un impact ; cependant, elle parvient à réduire la base de la complexité exponentielle à $\log ub$ au lieu de ub dans les encodages précédents.

Nous résumons les complexités des trois représentations présentées dans Table 5. Les trois méthodes de représentation offrent différents compromis en termes d'avantages et d'inconvénients. La représentation one-hot offre une implémentation simple, mais souffre d'une complexité élevée à mesure que le nombre de variables augmente. L'avantage clé de la représentation unitaire est la réduction de l'échelle du poids des clauses souples. La représentation binaire offre l'encodage le plus compact avec une mise à l'échelle logarithmique, mais devient plus complexe lorsqu'il s'agit de termes de grand ordre contenant seulement quelques variables différentes. Ces caractéristiques guident le choix des méthodes d'encodage appropriées en fonction de la structure spécifique des termes polynomiaux dans les applications pratiques. Malgré la complexité quantifiée, il est également important que la représentation soit compatible avec l'information sémantique originale du problème formulé.

4 Stratégies de décomposition pour les termes non linéaires

4.1 Au-delà du traitement atomique

Dans la section précédente, nous avons exploré les formulations atomiques basées sur trois représentations différentes, consistant à énumérer toutes les affectations possibles des valeurs des variables pour le terme polynomial donné avec m variables et d'ordre k, entraînant des complexités en termes de variables/contraintes de $O(ub^m)$, $O(ub^m)$ et $O((\log ub)^k)$ respectivement. Avec l'augmentation du nombre de variables (l'ordre), la complexité augmente de manière exponentielle et conduit à une explosion combinatoire. Cette explosion est initialement générée par le fait de traiter le terme polynomial comme une instance atomique et d'énumérer toutes les valeurs possibles. Dans cette section, nous essayons de décomposer un terme en plusieurs sous-unités, dont le produit reste égal au terme original. Plus précisément, étant donné un terme polynomial $T(\mathbf{X})$, la décomposition séquentielle présentée dans la section 4.2 calcule itérativement chaque produit cumulatif de $T(\mathbf{X})$, tandis que la décomposition binaire présentée dans la section 4.3 calcule récursivement le produit des unités voisines.

Aspect	One-hot	Unaire	Binaire
Variables Auxiliaires	$O(ub^m)$	$O(ub^m)$	$O((\log ub)^k)$
Contraintes	$O(ub^m)$	$O(ub^m)$	$O((\log ub)^k)$
Poids maximal	$O(ub^k)$	$O(ub^{k-1})$	$O(ub^k)$

TABLE 5 – Analyse de complexité des différentes formulations atomiques, où m est le nombre de variables distinctes dans le terme, $k \geq m$ est l'ordre du terme et ub est la borne supérieure des variables entières.

Exemple 9. *Considérons un terme polynomial* $T = X_1X_2X_3X_4X_5X_6X_7X_8$. *Les Figures 3 et 4 illustrent comment la décomposition séquentielle et la décomposition binaire traiteraient ce terme de manière différente. Comme montré dans la Figure 3, la décomposition séquentielle traite les variables de manière séquentielle de gauche à droite, tandis que la décomposition binaire présentée dans la Figure 4 permet un traitement parallèle à chaque niveau. Les deux méthodes nécessitent* 7 *multiplications.*

Les deux méthodes de décomposition laissent deux degrés de liberté : le choix de la représentation des entiers et le plan de la décomposition. En ce qui concerne le premier choix, les trois méthodes de représentation des entiers — one-hot, unitaire et binaire — peuvent être appliquées dans chacune des stratégies de décomposition. Par souci de clarté et de concision, nous concentrons notre discussion sur la tech-

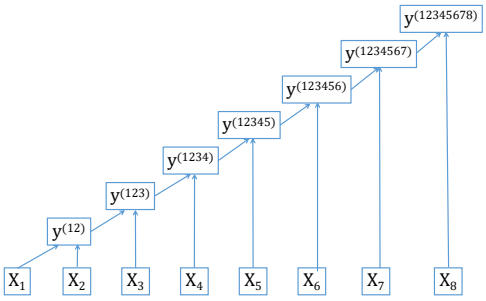


FIGURE 3 – Décomposition séquentielle de l'ordre

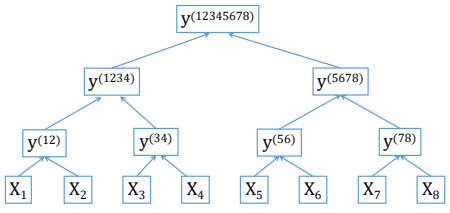


FIGURE 4 – Décomposition binaire de l'ordre

nique de représentation binaire. Le second choix concerne la manière de structurer l'arbre de décomposition lui-même et, plus spécifiquement, l'ordre et le regroupement des variables. Ce volet reste une question ouverte avec un potentiel d'optimisation important. Il est à noter que la représentation binaire offre une flexibilité supplémentaire : lorsque plusieurs termes polynomiaux partagent des sous-groupes de variables communes, ces sous-groupes peuvent être représentés par les mêmes nœuds dans leurs arbres de décomposition respectifs, réduisant ainsi potentiellement la redondance dans l'encodage. L'exemple présenté plus tard dans la section 5 illustrera cette pertinence.

De plus, afin de simplifier la discussion suivante, nous introduisons la notation $Mul(x^{(1)}, x^{(2)})$. Supposons que $x^{(1)}$ et $x^{(2)}$ soient deux séries de variables binaires représentant respectivement les deux entiers X_1 borné par ub_1 et X_2 borné par ub_2 sous la représentation binaire. $Mul(x^{(1)}, x^{(2)})$ définit la formulation binaire du produit de ces deux entiers. Selon l'Équation B-qua-cross, on a :

$$Mul(x^{(1)}, x^{(2)}) = \sum_{r_1=0}^{\lfloor \log ub_1 \rfloor} \sum_{r_2=0}^{\lfloor \log ub_2 \rfloor} 2^{r_1+r_2} z_{r_1 r_2} \quad \text{(Mul-B-1)}$$

$$z_{r_1 r_2} \leftrightarrow (x^{(1)}_{r_1} \wedge x^{(2)}_{r_2}) \quad \text{(Mul-B-2)}$$
$$\forall (r_1, r_2) \in \{0, \dots, \lfloor \log ub_1 \rfloor\} \times \{0, \dots, \lfloor \log ub_2 \rfloor\}$$

Nous remarquons que chaque appel de Mul utilise implicitement la formulation binaire pour un terme quadratique, et nécessite donc $O(\log ub_1 \cdot \log ub_2)$ variables auxiliaires et clauses.

4.2 Décomposition séquentielle

Dans cette section, nous présentons la décomposition séquentielle, qui traite le terme polynomial de manière séquentielle, calcule chaque produit cumulatif, et introduit une nouvelle série de variables y_r pour chaque opération de multiplication. Formellement, étant donné un terme polynomial $T(\mathbf{X}) = \prod_{q=1}^{m} X_q^{k_q}$, nous le réécrivons d'abord sous la forme $\prod_{q=q'_1}^{q'_k} X_q$ (tel que $\sum_{q=1}^{m} k_q = k$), puis nous continuons la procédure comme suit :

$$T(\mathbf{X}) = \prod_{q=1}^{m} X_q^{k_q} = \prod_{q=q'_1}^{q'_k} X_q \quad \text{(tel que } \sum_{q=1}^{m} k_q = k\text{)}$$

$$\sum_{r=0}^{\lfloor \log(ub^2) \rfloor} 2^r y_r^{(q'_1 q'_2)} = Mul(x^{(q'_1)}, x^{(q'_2)}) \quad \text{(LD-1)}$$

$$\sum_{r=0}^{\lfloor \log(ub^{\mathcal{K}}) \rfloor} 2^r y_r^{(q'_1 \dots q'_{\mathcal{K}})} = Mul(y^{(q'_1 \dots q'_{\mathcal{K}-1})}, x^{(q'_{\mathcal{K}})})$$
$$\forall \mathcal{K} \in \{3, ..., k\} \quad \text{(LD-2)}$$

$$T(\mathbf{X}) = \sum_{r=0}^{\lfloor \log(ub^k) \rfloor} 2^r y_r^{(q'_1 \dots q'_k)} \quad \text{(LD-3)}$$

Les équations LD-1 à LD-3 constituent la représentation de la décomposition séquentielle des termes de grand ordre. L'équation LD-3 décrit le remplacement du terme de grand ordre apparu par la somme pondérée des variables booléennes représentées en binaire $y^{(q'_1 \dots q'_k)}$. Les équations LD-1 et LD-2 sont utilisées pour maintenir de manière itérative le produit cumulatif du terme donné avec un total de $k-1$ multiplications, chaque contrainte de celles-ci pouvant être formulée comme une contrainte pseudo-booléenne avec jusqu'à $O(k^2 \cdot \log^2 ub)$ littéraux et un poids maximal inférieur à ub^k. L'échelle du nombre de clauses utilisées est corrélée à la méthode d'encodage PB choisie. Par exemple, si on considère l'encodage PB classique "binary merge" [28], qui parvient à encoder une contrainte pseudo-booléenne avec $O(\mathcal{N} \log^2(\mathcal{N}) \log(\mathcal{W}_{max}))$ clauses, où $\mathcal{N}$ est le nombre de littéraux et $\mathcal{W}_{max}$ est la valeur maximale des poids, cela mène finalement à une complexité globale de $O(k^4 \log^3 ub \cdot \log^2(k^2 \log^2 ub))$ clauses dans la formulation par décomposition séquentielle.

4.3 Décomposition binaire

La décomposition binaire offre une autre perspective pour traiter les termes de grand ordre, où nous divisons récursivement le produit en deux parties de taille similaire jusqu'à ce que chaque partie devienne une seule variable. L'idée principale est de diviser récursivement le produit en deux parties à peu près égales, en calculant le produit de chaque partie séparément, puis en les combinant. Cela forme une structure d'arbre binaire où chaque nœud interne représente le produit de ses deux enfants, comme illustré dans la Figure 4.

Étant donné un terme polynomial $T(\mathbf{X}) = \prod_{q=1}^{m} X_q^{k_q}$, nous le réécrivons d'abord sous la forme $\prod_{q=q'_1}^{q'_k} X_q$ (tel que $\sum_{q=1}^{m} k_q = k$), puis nous appliquons la décomposition binaire. Cette procédure commence par obtenir la valeur de $\mathcal{P}$, représentant l'ensemble de toutes les paires d'intervalles adjacents devant être multipliées dans l'arbre binaire. Formellement, $\mathcal{P}$ peut être défini comme suit :

$$\mathcal{P} = \{(\mathcal{I}, \mathcal{J}) \mid \mathcal{I}, \mathcal{J} \subset \{q'_1, q'_2, \dots, q'_k\}, \mathcal{I} \cap \mathcal{J} = \emptyset, \mathcal{I} \cup \mathcal{J} \in \mathcal{T}\}$$

où $\mathcal{T}$ est l'ensemble de tous les nœuds non-feuilles dans l'arbre de décomposition binaire, et pour chaque nœud $N \in \mathcal{T}$, il existe exactement une paire $(\mathcal{I}, \mathcal{J}) \in \mathcal{P}$ telle que $\mathcal{I} \cup \mathcal{J} = N$ et que $\mathcal{I}$ et $\mathcal{J}$ sont des enfants de N dans l'arbre.

$$T(\mathbf{X}) = \prod_{q=q'_1}^{q'_k} X_q = \prod_{q=q'_1}^{q'_{\lceil k/2 \rceil}} X_q \cdot \prod_{q=q'_{\lceil k/2 \rceil + 1}}^{q'_k} X_q$$

$$y_r^{(q)} = x_r^{(q)} \quad \forall q \in \{q'_1, ..., q'_k\},$$
$$\forall r \in \{0, \dots, \lfloor \log(ub^k) \rfloor\} \quad \text{(BD-1)}$$

$$\sum_{r=0}^{\lfloor \log(ub^{|\mathcal{I}|+|\mathcal{J}|}) \rfloor} 2^r y_r^{(\mathcal{I},\mathcal{J})} = Mul(y^{(\mathcal{I})}, y^{(\mathcal{J})})$$
$$\forall (\mathcal{I}, \mathcal{J}) \in \mathcal{P} \quad \text{(BD-2)}$$

$$T(\mathbf{X}) = \sum_{r=0}^{\lfloor \log(ub^k) \rfloor} 2^r y_r^{(q'_1 \dots q'_k)} \quad \text{(BD-3)}$$

Exemple 10. *Étant donné $k = 8$, l'ensemble $\mathcal{P}$ obtenu inclut les paires d'intervalles suivantes :*

— *Niveau 1 :* (q'_1, q'_2), (q'_3, q'_4), (q'_5, q'_6),(q'_7, q'_8)
— *Niveau 2 :* $(q'_1q'_2, q'_3q'_4)$, $(q'_5q'_6, q'_7q'_8)$
— *Niveau 3 :* $(q'_1q'_2q'_3q'_4, q'_5q'_6q'_7q'_8)$

L'équation BD-1 initialise les variables auxiliaires $y_r^{(q)}$ avec les variables d'origine $x_r^{(q)}$. L'équation BD-2 combine récursivement les paires de sous-produits en utilisant l'opération Mul défini précédemment, et l'équation BD-3 représente le résultat final. La principale différence entre la décomposition binaire et la décomposition séquentielle réside dans leurs modèles de multiplication. L'échelle du nombre de sommets dans un arbre binaire avec k sommets est $O(k)$, donc il y a $O(k)$ procédures de multiplication dans la procédure de décomposition binaire, ce qui est aligné avec la décomposition séquentielle proposée dans la sous-section précédente. De manière similaire à la décomposition séquentielle, lorsqu'on considère l'encodage PB avec la méthode "Binary Merger", la complexité finale devient $O(k^4 \log^3 ub \cdot \log^2(k^2 \log^2 ub))$. Bien que la complexité générale soit la même, nous notons que la décomposition binaire maintient des distances moyennes plus courtes entre les valeurs entières originales et le produit général, ce qui conduit intuitivement à des lignes d'inférence plus courtes lorsque le modèle est résolu.

5 Étude de cas

Dans cette section, nous étudions la pertinence de la formulation proposée sur la fonction polynomiale de grand ordre $P(\mathbf{X}) = X_1X_2X_3X_4 + X_3X_4X_5X_6 + X_1X_2X_5X_6$, où toutes les variables sont bornées par ub. Limités par l'espace, nous ne présentons ici que la formulation basée sur la décomposition binaire. Dans ce cas spécifique de P, un plan de partition est présenté dans la Figure 5. En conséquence, nous pouvons maintenir les ensembles $\mathcal{P}$ pour les trois termes respectivement, c'est-à-dire : $P_{1234} = \{(1,2),(3,4),(12,34)\}$, $P_{3456} = \{(3,4),(5,6),(34,56)\}$, et $P_{1256} = \{(1,2),(5,6),(12,56)\}$.
Pour chaque élément de $\mathcal{P}$, une nouvelle série de variables auxiliaires est introduite. Cependant, grâce à la structure spécifique et au plan de partitionnement choisi, nous parvenons à utiliser efficacement les variables auxiliaires intermédiaires $y^{(12)}, y^{(34)}$ et $y^{(56)}$, chacune servant de composant dans deux termes différents. À partir de ce plan de décomposition, nous pouvons construire les termes du polynôme en utilisant l'une des trois représentations suivantes : one-hot, unaire ou binaire, chacune impliquant une échelle différente de variables auxiliaires.
La formulation binaire dans le cadre de la décomposition binaire a été introduite dans la sous-section 4.3. Nous présentons ici simplement la formulation correspondante lorsqu'elle est appliquée au polynôme P.

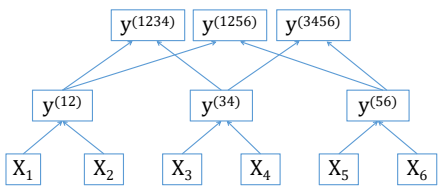


FIGURE 5 – Un plan de décomposition binaire compact pour un polynôme d'ordre supérieur

$$y_r^{(q)} = x_r^{(q)} \quad \forall (q,r) \in \{1,\dots,6\} \times \{0,\dots,\lfloor \log ub \rfloor\} \quad \text{(BD-B-1)}$$

$$\sum_{r=0}^{\lfloor \log(ub^{|\mathcal{I}|+|\mathcal{J}|}) \rfloor} 2^r y_r^{(\mathcal{I},\mathcal{J})} = Mul(y^{(\mathcal{I})}, y^{(\mathcal{J})})$$
$$(\mathcal{I},\mathcal{J}) \in \mathcal{P}_{1234} \cup \mathcal{P}_{3456} \cup \mathcal{P}_{1256} \quad \text{(BD-B-2)}$$

$$P(X) = \sum_{r=0}^{\lfloor \log(ub^4) \rfloor} 2^r (y_r^{(1234)} + y_r^{(3456)} + y_r^{(1256)}) \quad \text{(BD-B-3)}$$

L'équation BD-B-1 représente la base de la structure récursive de la décomposition binaire, tandis que l'équation BD-B-2 définit une étape récursive obtenue par la représentation binaire, reliant deux niveaux de variables auxiliaires. L'équation BD-B-3 décrit la reformulation finale du polynôme.
Nous remarquons que l'opération de multiplication (Mul) joue un rôle essentiel dans la formulation. En raison de contraintes d'espace, nous nous limitons dans la suite à la présentation de la formulation unaire, celle basée sur one-hot pouvant être modélisée de manière analogue. Pour obtenir la formulation correspondante, nous définissons d'abord l'opération de multiplication correspondante avec la formulation unaire comme suit :

$$Mul^{(U)}(x^{(1)}, x^{(2)}) = \sum_{r_1=0}^{ub_1} \sum_{r_2=0}^{ub_2} \widetilde{T}(r_1) \cdot \widetilde{T}(r_2) z_{r_1 r_2} \quad \text{(Mul-U-1)}$$

$$z_{r_1 r_2} \leftrightarrow (x_{r_1}^{(1)} \wedge x_{r_2}^{(2)})$$
$$\forall (r_1, r_2) \in \{0,\dots,ub_1\} \times \{0,\dots,ub_2\} \quad \text{(Mul-U-2)}$$

Ainsi, nous pouvons établir le plan de formulation suivant pour le polynôme $P(\mathbf{X})$ en utilisant la formulation unaire dans le cadre d'une décomposition binaire :

$$y_r^{(\mathcal{I},\mathcal{J})} \rightarrow y_{r-1}^{(\mathcal{I},\mathcal{J})} \quad \forall r \in \{1,\dots,ub^{|\mathcal{I}|+|\mathcal{J}|}\}$$
$$\forall (\mathcal{I},\mathcal{J}) \in \mathcal{P}_{1234} \cup \mathcal{P}_{3456} \cup \mathcal{P}_{1256} \quad \text{(BD-U-0)}$$

$$y_r^{(q)} = x_r^{(q)} \quad \forall (q,r) \in \{1,\dots,6\} \times V \quad \text{(BD-U-1)}$$

$$\sum_{r=0}^{ub^{|\mathcal{I}|+|\mathcal{J}|}} \widetilde{T}(r) \cdot y_r^{(\mathcal{I},\mathcal{J})} = Mul^{(U)}(y^{(\mathcal{I})}, y^{(\mathcal{J})})$$
$$(\mathcal{I},\mathcal{J}) \in \mathcal{P}_{1234} \cup \mathcal{P}_{3456} \cup \mathcal{P}_{1256} \quad \text{(BD-U-2)}$$

$$P(X) = \sum_{r_1=0}^{ub^4} \widetilde{T}(r)(y_r^{(1234)} + y_r^{(3456)} + y_r^{(1256)}) \quad \text{(BD-U-3)}$$

6 Conclusion

Dans cet article, nous avons étudié différentes formulations fondées sur MaxSAT pour les programmes entiers non linéaires comportant des termes polynomiaux. Nous avons d'abord présenté des encodages pour les termes polynomiaux, en proposant trois stratégies différentes, adaptées à divers cas d'usage et avec des complexités différentes. Nous nous sommes ensuite concentrés sur la décomposition des termes selon leur ordre, ce qui permet d'obtenir des formulations à complexité réduite. En combinant ces deux approches, nous proposons ainsi un schéma de formulation générique applicable à toute instance de programmation entière non linéaire incluant des termes polynomiaux.
Comme perspectives, il serait intéressant de continuer à explorer les trois autres méthodes de mappage proposées dans [2], ainsi que d'affiner la stratégie de sélection des méthodes de représentation/décomposition. Sur le plan pratique, après la maturité de la méthode de formulation proposée, une mise en œuvre approfondie des formulations pourrait être appliquée à différents problèmes du monde réel, élargissant ainsi le champ d'application des technologies modernes SAT et MaxSAT. Nous espérons également développer une bibliothèque permettant une utilisation plus facile des formulations et qui sera mise à disposition de la communauté.

Remerciements

Ce travail est partiellement soutenu par le projet ANR-24-CE23-6126 (BforSAT) et la chaire IA ANR-19-CHIA-0013-01 (MASSAL'IA) co-financée par l'agence nationale de recherche et le gestionnaire du réseau de distribution d'électricité Enedis.

Références

[1] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Bdds for pseudo-boolean constraints - revisited. In K. A. Sakallah and L. Simon, editors, *SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011*, volume 6695 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2011.

[2] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *SAT 2004, 10-13 May 2004, Vancouver, BC, Canada*, 2004.

[3] M. Anthony, E. Boros, Y. Crama, and A. Gruber. Quadratic reformulations of nonlinear binary optimization problems. *Math. Program.*, 162(1-2) :115–144, 2017.

[4] F. Bacchus, M. Järvisalo, and R. Martins. Maximum satisfiabiliy. In *Handbook of satisfiability*, pages 929–991. IOS Press, 2021.

[5] E. Balas and J. B. Mazzola. Nonlinear 0-1 programming : I. linearization techniques. *Math. Program.*, 30(1) :1–21, 1984.

[6] C. Barrett and S. Berezin. A proof-producing boolean search engine. *Program Commitee*, page 25, 2003.

[7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[8] M. Bofill, J. Coll, M. Garcia, J. Giráldez-Cru, G. Pesant, J. Suy, and M. Villaret. Constraint solving approaches to the business-to-business meeting scheduling problem. *J. Artif. Intell. Res.*, 74 :263–301, 2022.

[9] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discret. Appl. Math.*, 123(1-3) :155–225, 2002.

[10] S. Cherif, H. Sattoutah, C. Li, C. Lucet, and L. B. Devendeville. Minimizing working-group conflicts in conference session scheduling through maximum satisfiability (short paper). In P. Shaw, editor, *CP 2024, September 2-6, 2024, Girona, Spain*, volume 307 of *LIPIcs*, pages 34 :1–34 :11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[11] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100(1) :32–74, 1928.

[12] Y. Crama and P. L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2011.

[13] M. A. Duran and I. E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program.*, 39(3) :337, 1987.

[14] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.*, 2(1-4) :1–26, 2006.

[15] A. V. Gelder. Another look at graph coloring via propositional satisfiability. *Discret. Appl. Math.*, 156(2) :230–243, 2008.

[16] A. Graça, I. Lynce, J. Marques-Silva, and A. L. Oliveira. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In K. Horimoto, M. Nakatsui, and N. Popov, editors, *ANB 2010, Hagenberg, Austria, July 31- August 2, 2010*, volume 6479 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2010.

[17] J. Guerra and I. Lynce. Reasoning over biological networks using maximum satisfiability. In M. Milano, editor, *CP 2012, Québec City, QC, Canada, October 8-12, 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 941–956. Springer, 2012.

[18] A. Gupta, M. K. Ganai, and C. Wang. Sat-based verification methods and applications in hardware verification. In M. Bernardo and A. Cimatti, editors, *SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*, volume 3965 of *Lecture Notes in Computer Science*, pages 108–143. Springer, 2006.

[19] P. Hammer Ivànescu and S. Rudeanu. Boolean methods in operations research and related areas. *Oekonometrie und Unternehmensforschung (*, 7, 1968.

[20] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel. Nonlinear integer programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 561–618. Springer, 2010.

[21] M. HM. Portfolio selection : efficient diversification of investments, 1959.

[22] R. G. Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Oper. Res.*, 21(1) :221–224, 1973.

[23] W. Klieber and G. Kwon. Efficient cnf encoding for selecting 1 from n objects. In *Proc. International Workshop on Constraints in Formal Verification*, page 14, 2007.

[24] C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021.

[25] D. Li, X. Sun, et al. *Nonlinear integer programming*, volume 84. Springer, 2006.

[26] S. Li, C. Li, J. Coll, D. Habet, and F. Manyà. Improving the lower bound in branch-and-bound algorithms for maxsat. In T. Walsh, J. Shah, and Z. Kolter, editors, *AAAI-25, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 11272–11281. AAAI Press, 2025.

[27] J. A. D. Loera, R. Hemmecke, M. Köppe, and R. Weismantel. Integer polynomial optimization in fixed dimension. *Math. Oper. Res.*, 31(1) :147–153, 2006.

[28] N. Manthey, T. Philipp, and P. Steinke. A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In C. Lutz and M. Thielscher, editors, *KI 2014 : Advances in Artificial Intelligence, Stuttgart, Germany, September 22-26, 2014.*, volume 8736 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.

[29] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem : A survey and recent developments. 1994.

[30] E. Rodríguez-Heck. Linear and quadratic reformulations of nonlinear optimization problems in binary variables. *4OR*, 17(2) :221–222, 2019.

[31] I. G. Rosenberg. Reduction of bivalent maximization to the quadratic case. 1975.

[32] O. Roussel and V. Manquinho. Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1087–1129. IOS Press, 2021.

[33] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *FMCAD 2007, Austin, Texas, USA, November 11-14, 2007.*, pages 13–19. IEEE Computer Society, 2007.

[34] J. C. Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.

[35] P. Surynek. Lazy compilation of variants of multi-robot path planning with satisfiability modulo theory (SMT) approach. In *IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 3282–3287. IEEE, 2019.

[36] Z. Zheng, S. Cherif, and R. S. Shibasaki. Minimisation des pics d'énergie dans l'équilibrage des lignes d'assemblage par la satisfiabilité maximum. In *Actes JFPC 2024*, 2024.