N° d'ordre : 5289

École Doctorale Mathématiques, Sciences de l'Information et de l'ingénieur

ULP-INSA-ENGEES

THÈSE

présentée pour obtenir le grade de

Docteur de l'Université Louis Pasteur-Strasbourg I
Spécialité : Informatique

par

Rachid SEGHIR

MÉTHODES DE DÉNOMBREMENT DE POINTS ENTIERS DE POLYÈDRES ET APPLICATIONS À L'OPTIMISATION DE PROGRAMMES

Soutenue publiquement le 07 décembre 2006

Membres du jury

Président du jury :	M. Patrice Quinton (Professeur, ENS de Cachan, Bruz)
Directeur de thèse :	Mme. Catherine Mongenet (Professeur, ULP, Strasbourg)
Co-Directeur de thèse :	M. Vincent Loechner (Maître de conférences, ULP, Strasbourg)
Rapporteur interne :	M. Pascal Schreck (Professeur, ULP, Strasbourg)
Rapporteur externe :	Mme. Christine Eisenbeis (Directeur de recherche INRIA, INRIA Futurs, Orsay)
Rapporteur externe :	M. Paul Feautrier (Professeur, ENS de Lyon, Lyon)

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection (UMR ULP-CNRS 7005)

A tous ceux qui me sont chers

Remerciements

Mes premiers remerciements vont naturellement à mes directeurs de thèse, Catherine Mongenet et Vincent Loechner, sans qui cette thèse n'aurait pu voir le jour. Je tiens à leur exprimer ma gratitude pour leur soutien, leurs précieux conseils et pour tout ce que j'ai pu apprendre avec eux durant ces trois années de thèse.

Je remercie l'ensemble des membres du jury qui m'ont fait l'immense plaisir de juger ce travail. Sincères remerciements au Président du jury, Patrice Quinton, Professeur à l'ENS de Cachan, aux rapporteurs Paul Feautrier, Professeur à l'ENS de Lyon, Christine Eisenbeis, Directeur de recherche à l'INRIA Futurs et Pascal Schreck, Professeur à l'université Louis Pasteur de Strasbourg.

Je remercie également Sven Verdoolaege, Kristof Beyls et Benoît Meister avec lesquels j'ai eu le plaisir de collaborer pour mener à bien une partie de ce travail.

Je voudrais aussi exprimer ma reconnaissance à Philippe Clauss pour m'avoir accueilli dans son équipe ICPS, ainsi qu'à tous les membres de cette équipe pour leur sympathie et pour l'ambiance conviviale qui nous a rassemblée durant toutes ces années.

Enfin, je ne saurais oublier de remercier et d'exprimer ma gratitude à mes parents, mes frères et mes sœurs pour le soutien continuel qu'ils m'apportent, ainsi qu'à Kamel, Yazid et tous mes amis et collègues.

Table des matières

Table des figures

Chapitre 1

Introduction

Le vingtième siècle fut marqué par l'apparition d'un outil formidable de calcul, c'est *l'ordinateur*. Tant attendu, l'avènement de la première génération d'ordinateurs, dans les années quarante, annonça le début d'une histoire bouleversante dans le monde de la technologie. Bien que les deux premières générations d'ordinateurs furent considérées comme un succès technologique sans précédent, l'explosion de l'utilisation de l'informatique n'a eu lieu qu'au début des années soixante, avec l'arrivée de la troisième génération d'ordinateurs, caractérisés par les circuits intégrés. En 1971, Intel dévoila le premier microprocesseur commercial, le 4004, qui regroupe la plupart des composants de calcul. C'est cet évènement qui marqua un changement radical du visage des ordinateurs et fit naître le micro-ordinateur moderne, ou l'ordinateur de la quatrième génération.

Depuis sa découverte, la technologie du processeur n'a cessé de produire des composants de plus en plus performants. Ceci est bien évidement motivé par une quête de puissance de calcul sans cesse croissante. Cependant, le processeur n'est absolument pas le seul responsable des performances d'un ordinateur. L'unité de stockage de données ou *mémoire centrale* y joue un rôle fondamental, puisqu'un processeur ne peut effectuer de calculs en l'absence de données qu'il doit chercher en mémoire. Malheureusement, la technologie de la mémoire ne peut suivre celle du processeur dans son essor. Si la vitesse des processeurs progresse d'environ 60% par an, celle des accès à la mémoire centrale ne progresse que d'environ 10% [35]. Ce qui fait que l'écart de la vitesse entre le processeur et la mémoire progresse de 45% chaque année. Pour amoindrir l'impact de cet écart, les ordinateurs modernes mettent en place des mémoires particulières, dites *mémoires caches*, entre l'unité de calcul du processeur et la mémoire centrale. Ce sont des mémoires de petite taille, mais ayant une grande vitesse d'accès aux données qu'elles contiennent. La mémoire centrale n'est donc accédée que si la donnée recherchée est absente au niveau du cache. Un autre défi, auquel les concepteurs d'architectures ont à faire face, est dû au fait que quelles que soient ses performances, un processeur ne peut réaliser plusieurs calculs au même moment, et ce même si différentes parties d'un programme peuvent être exécutées indépendamment les unes des autres. Ceci a motivé la conception des machines multiprocesseurs pour profiter du parallélisme inhérent à de nombreuses applications. Différentes parties d'un programme peuvent ainsi être exécutées au même moment, par les différents processeurs, ce qui peut diminuer considérablement le temps global de son exécution. Alternative au parallélisme, l'utilisation

de circuits dédiés, parfois reconfigurables, spécialisés dans le traitement d'une tâche est une autre évolution des architectures actuelles.

Les progrès réalisés en architecture ont souvent permis de répondre pertinemment aux besoins des programmes scientifiques. Néanmoins, l'avènement des applications modernes, de plus en plus exigeantes en performance et miniaturisation, telles que les applications scientifiques, multimédias ou embarquées, a montré que certains programmes ont un comportement qui ne leur permet pas de bénéficier pleinement des performances matérielles qui leurs sont offertes. Pour utiliser efficacement le matériel, il convient de transformer et d'optimiser ces programmes. Bien évidemment, il est irréaliste d'optimiser manuellement tous les programmes, vu leur complexité et la diversité des architectures sur lesquelles ils s'exécutent. Ce qui fait de l'optimisation *automatique* de programmes une solution nécessaire. Il s'agit de concevoir des méthodes de transformation de programmes, et de les intégrer dans des compilateurs *intelligents* qui permettent de générer, à partir d'un programme utilisateur, un code optimisé qui exploite le mieux possible les ressources matérielles mises à sa disposition.

Depuis trois décennies, de nombreuses méthodes d'optimisation automatique de programmes ont été développées. Le plus souvent, ces méthodes sont centrées sur l'analyse et la transformation des structures de contrôle itératives, communément appelées *nids de boucles*. Ceci est dû au fait que les boucles sont généralement les parties de programme qui portent plus de parallélisme, qui utilisent la plus grande partie de la mémoire, et qui consomment le plus de temps de calcul. De plus, les nids de boucles ne représentent qu'une petite partie de la taille totale des programmes, ce qui rend leur analyse beaucoup moins fastidieuse que lorsqu'on considère le programme dans sa totalité. Par nature, la plupart des nids de boucles qui apparaissent dans des programmes sont *affines*, c'est-à-dire qu'ils s'expriment par des indices bornés par des expressions affines, et des accès à des tableaux par des fonctions affines des indices. Puisque le domaine d'itération d'un nid de boucles affines peut être décrit par les points entiers appartenant à un *polytope* (polyèdre borné) [84], de nombreuses méthodes d'optimisation de programmes formalisent les problèmes d'analyse et de transformation de nids de boucles par le calcul du nombre de points entiers dans des polytopes, et de leurs images affines. Lorsque les nids de boucles sont *paramétrés*, i.e., dépendent d'un ensemble de constantes symboliques non connues au moment de la compilation, les problèmes d'analyse et d'optimisation de nids de boucles se réduisent souvent au comptage de points entiers dans des polytopes *paramétrés*. Dans ce cas, le résultat du comptage est donné par un ou plusieurs polynômes ayant des coefficients *périodiques*. Ces polynômes sont connus sous le nom de *quasi-polynômes d'Ehrhart* [52, 39].

Conjointement aux recherches menées dans le contexte de l'analyse et de l'optimisation de programmes, d'autres recherches ont visé les domaines de la théorie des nombres, de la programmation linéaire en nombres entiers et de l'optimisation combinatoire. Ces recherches ont tenté d'apporter des réponses aux différents problèmes soulevés par les méthodes d'optimisation automatique de programmes. Deux classes d'algorithmes ont été proposées : (i) les algorithmes de comptage de points entiers dans des polytopes, parmi lesquels on distingue les algorithmes paramétrés [115, 39, 17, 9] et les algorithmes non paramétrés [14, 46, 26, 113] ; (ii) les algorithmes de calcul et de dénombrement des images affines de $\mathbb{Z}$-polytopes. On y distingue aussi les algorithmes paramétrés [115, 28, 144, 149, 104] et non paramétrés [156, 13, 26, 113]. Certains de ces

algorithmes sont assez efficaces, mais restent loin de satisfaire toutes les applications. D'autres algorithmes sont exponentiels, difficile à implémenter, ou ne permettent de traiter que des cas particuliers. Nous reviendrons plus en détail sur ces algorithmes dans les prochains chapitres.

L'usage à grande envergure des algorithmes de comptage des points entiers dans des polytopes et dans leurs transformations a motivé le développement permanent de nouvelles approches, de plus en plus efficaces, mais la recherche de la solution optimale reste au jour d'aujourd'hui un problème ouvert.

Dans le cadre de cette thèse, nous nous sommes intéressés aux méthodes d'optimisation de programmes qui soulèvent les problèmes de comptage de points à coordonnées entières dans des ensembles paramétrés. Pour cela, nous avons développé trois algorithmes de comptage.

1. *Un algorithme de dénombrement de points entiers d'un polytope paramétré :* cet algorithme est une généralisation, au cas paramétré, de l'algorithme de Barvinok [14], basé sur la décomposition d'un polytope en cônes unimodulaires. L'idée principale est la représentation du nombre de points entiers d'un polytope par la somme des fonctions génératrices rationnelles (paramétrées) de ses cônes générateurs, pour trouver un ou plusieurs quasi-polynômes d'Ehrhart.
2. *Un algorithme de dénombrement des unions de $\mathbb{Z}$-polytopes paramétrés :* un $\mathbb{Z}$-polytope est l'intersection d'un polytope et d'un *lattice* entier (réseau régulier de points entiers)[1]. L'algorithme proposé utilise le principe d'inclusion-exclusion : la cardinalité de deux ensembles est égale à la cardinalité du premier plus la cardinalité du deuxième moins la cardinalité de leur intersection. Ce principe s'applique, évidemment, à un nombre quelconque de $\mathbb{Z}$-polytopes. Après application du principe d'inclusion-exclusion, chaque $\mathbb{Z}$-polytope résultant est transformé, puis réécrit en fonction de ses paramètres originaux. Ensuite le nombre de points entiers dans chaque polytope est calculé par l'algorithme ci-dessus.
3. *Un algorithme de calcul et de dénombrement des images affines de $\mathbb{Z}$-polytopes paramétrés :* cet algorithme est inspiré de l'élimination de variables existentielles de Fourier-Motzkin. W. Pugh [116] a démontré que l'image affine d'un $\mathbb{Z}$-polytope peut être représentée par une formule de Presburger, i.e., un ensemble de contraintes linéaires, liées par les quantificateurs existentiel et universel, et les connecteurs logique (*et*, *ou* et *non*). Dans notre contexte, le problème se réduit à l'élimination des variables existentielles. Pour ce faire, nous commençons par éliminer les égalités de la formule impliquant des variables existentielles, ce qui permet d'éliminer un premier ensemble de ces variables. Ensuite nous éliminons le reste des variables existentielles en éliminant les points de l'image rationnelle qui ne possèdent pas d'antécédents entiers. Le résultat de cet algorithme est donné par une union de $\mathbb{Z}$-polytopes, où le nombre de points est calculé par l'algorithme précédent.

Ces algorithmes permettent de répondre à différents problèmes soulevés par les méthodes d'analyse et d'optimisation de nids de boucles affines, tels que : le nombre

[1] Bien que le mot *lattice* soit anglais, il est largement utilisé par la communauté francophone pour désigner un réseau régulier de points. Le mot *treillis* est aussi utilisé, mais, à l'origine, ce dernier fait intervenir une relation d'ordre, et ne correspond pas tout à fait aux lattices que nous manipulons. Tout au long de ce mémoire nous utiliserons le mot *lattice* plutôt que *réseau régulier de points* ou *treillis*.

de localisations mémoire ou de lignes de cache touchées par un nid boucles, le nombre d'éléments d'un tableau accédés entre deux instants, le nombre de défauts de cache générés par un morceau de programme, le nombre d'éléments d'un tableau accédés avant un élément donné, le nombre d'octets transmis entre deux tâches parallèles, et le nombre de fois qu'une instruction est exécutée avant une itération donnée. Dans le cadre de cette thèse, nous avons appliqué les algorithmes ci-dessus au calcul de la linéarisation de tableaux, qui permet la compression de données et l'amélioration de la localité spatiale, et au calcul des distances de réutilisation de données pour améliorer la politique de remplacement de cache LRU. Les algorithmes proposés trouvent aussi des applications dans d'autre domaines, tels que les mathématiques et l'économie.

Le reste de ce mémoire est organisé comme suit.

Dans le chapitre 2, nous introduisons le modèle polyédrique, et nous faisons un tour d'horizon des méthodes d'optimisation automatique de programmes, en particulier celles dédiées à l'optimisation des accès mémoire. Ensuite nous étudions les polyèdres paramétrés et les quasi-polynômes d'Ehrhart qui introduisent la problématique du comptage de points entiers dans des ensembles convexes paramétrés.

Dans le chapitre 3, nous présentons un aperçu des méthodes existantes de dénombrement de points entiers dans des polytopes, et nous expliquons plus en détail l'algorithme de Barvinok [14], qui introduira notre algorithme traitant les polytopes paramétrés. Ensuite, nous présentons notre algorithme, nous étudions sa complexité et nous comparons ses performances avec celles de la méthode d'interpolation de Clauss-Loechner [39].

Dans le chapitre 4, nous nous intéressons au dénombrement des unions de $\mathbb{Z}$-polytopes. Nous commençons par introduire quelques définitions et l'approche de dénombrement par calcul des unions disjointes. Ensuite, nous présentons notre algorithme basé sur le principe d'inclusion-exclusion, nous étudions sa complexité et nous donnons quelques résultats expérimentaux.

Dans le chapitre 5, nous étudions les images affines de $\mathbb{Z}$-polytopes paramétrés. Nous commençons par présenter une méthode d'élimination des égalités d'un $\mathbb{Z}$-polytope. Puis, nous présentons un algorithme de projection d'un $\mathbb{Z}$-polytope. Nous donnons également un aperçu de quelques méthodes de calcul et de dénombrement des images affines de $\mathbb{Z}$-polytopes, et nous comparons les performances de notre méthode avec celles de la méthode de Verdoolaege et al. [144].

Enfin nous présentons quelques applications de nos algorithmes en optimisation automatique dans le chapitre 6, avant de conclure dans le chapitre 7.

Chapitre 2

Modèle polyédrique et quasi-polynômes d'Ehrhart

Dans ce chapitre, nous présentons d'abord le contexte général dans lequel se placent nos contributions, puis nous étudions les outils fondamentaux qui sont à la base de ces contributions, à savoir les polyèdres convexes paramétrés et les quasi-polynômes d'Ehrhart.

2.1 Modèle polyédrique

Le modèle polyédrique [2, 55, 88, 56, 117, 15] est un formalisme qui permet de représenter des calculs massivement parallèles en fournissant une représentation compacte. Ce modèle est fondé sur les systèmes d'équations récurrentes affines, définies sur des domaines polyédriques [107]. Une équation récurrente est une fonction qui détermine comment une valeur d'une variable est définie (récursivement), en un point de l'espace des indices, en fonction de sa valeur en d'autres points. Le domaine d'une équation récurrente est l'ensemble des points correspondant aux indices sur lesquels elle est définie. Les systèmes d'équations récurrentes ont d'abord été introduits aux architectures parallèles, par Karp et al. [75], sous une forme simple connue sous le nom de SURE (*Systems of Uniform Recurence Equations*), où les fonctions d'accès aux tableaux sont des fonctions *uniformes* (des translations). Ces systèmes ont par la suite été généralisés pour prendre en compte des fonctions d'accès *affines*. L'utilisation des systèmes d'équations récurrentes affines en parallélisation automatique remonte au travail de Kuck [84], qui a montré que le domaine d'itération d'un *nid de boucles* (ensemble de boucles imbriquées), avec des bornes inférieures et supérieures affines, peut être décrit par un *polyèdre* borné par un ensemble de demi-espaces. Chaque demi-espace correspond à une borne inférieure ou supérieure sur un indice. La dimension du polyèdre ainsi défini est égale à la profondeur du nid de boucles (le nombre de ses indices). Enfin, chaque point à coordonnées entières (vecteur entier) à l'intérieur du polyèdre correspond à une itération du nid de boucles. Lorsque le nombre d'itérations n'est pas fixe (ne peut pas être déterminé au moment de la compilation), on parle de nids de boucles *paramétrés*. Ce sont des nids de boucles qui comportent des constantes symboliques (paramètres) dans les expressions affines de leurs bornes. Un nid de boucles où toutes les instructions

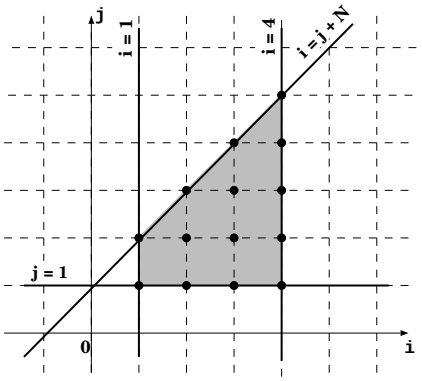


FIG. 2.1 – Représentation des itérations du nid de boucles de l'exemple 1 (pour $N = 1$).

se trouvent au niveau le plus interne est dit *parfait*. La forme générale d'un nid de boucles parfait de profondeur d est la suivante :

$$\begin{array}{l} \text{for } \; i_1 = l_1(\mathbf{p}) \; \text{ to } \; u_1(\mathbf{p}) \\ \quad \text{for } \; i_2 = l_2(i_1, \mathbf{p}) \; \text{ to } \; u_2(i_1, \mathbf{p}) \\ \qquad \cdots \\ \qquad\quad \text{for } \; i_d = l_d(i_1, \cdots i_{d-1}, \mathbf{p}) \; \text{ to } \; u_d(i_1, \cdots, i_{d-1}, \mathbf{p}) \\ \qquad\quad \cdots \end{array}$$

où i_j $(j = 1, \cdots, d)$ sont les indices du nid de boucles, $\mathbf{p}$ est un vecteur de paramètres et l_j, u_j $(j = 1, \cdots, d)$ sont des fonctions affines. Une fonction affine est de la forme : $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle + c$, où $\langle ., . \rangle$ est le produit scalaire de vecteurs. Les références aux données qui apparaissent dans le corps d'un nid de boucles *affine* sont celles qui accèdent soit à des tableaux par des fonctions affines des indices, soit à des données scalaires. Lorsque le nid de boucles n'est pas parfait, des instructions peuvent y apparaître à n'importe quel niveau de profondeur.

Exemple 1. Considérons le morceau de programme (nid de boucles) suivant :

```
for(i=1; i<=4; i++)
   for(j=1; j<=i+N; j++)
      S ...
```

Les itérations de ce nid de boucles correspondent aux points à coordonnées entières d'un polytope paramétré P_N :

$$P_N = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \;\middle|\; 1 \leq i \leq 4 \;\wedge\; 1 \leq j \leq i + N \right\},$$

où N est un paramètre entier. La représentation graphique des itérations de ce nid de boucles, lorsque $N = 1$, est donnée par la figure 2.1.

2.2 Méthodes d'optimisation automatique de programmes

Depuis sa découverte, le modèle polyédrique n'a cessé d'être à la base des méthodes de parallélisation automatique de programmes, de plus en plus performantes. Le succès flagrant de ce modèle est, sans doute, dû au fait qu'il vise directement les parties les plus coûteuses de programmes (les nids de boules). En effet, les boucles sont connues pour être les parties les plus consommatrices dans un programme, que ce soit en taille mémoire utilisée ou en temps d'exécution. Durant une trentaine d'années, un grand nombre de méthodes d'optimisation automatique de programmes ont été développées. On y trouve par exemple : la transformation de boucles en vue de leur parallélisation, telles que le *tiling* de boucles [153, 27, 132, 92] et la fusion de boucles [101, 91, 43, 145] ; l'optimisation des performances de la hiérarchie mémoire [3, 74, 97, 48, 24, 99, 32] ; l'estimation du temps d'exécution au pire cas (WCET) [93] ; la transformation de haut-niveau pour les application DSP (Digital Signal Processing) [61] ; et la conversion de boucles (software) en circuits (hardware) parallèles [138, 68].

De nombreuses méthodes d'optimisation de programmes sont basées sur l'analyse de nids de boucles affines. Parmi les questions posées par ce type d'analyse citons :

- Quel est le nombre de localisations mémoire touchées par une boucle ? [59, 156]
- Quel est le nombre de lignes de cache touchées par une boucle ? [59]
- Quel est le nombre d'éléments d'un tableau accédés entre deux instants ? [24]
- Quel est le nombre d'éléments d'un tableau qui sont en vie (qui ont déjà été utilisés et qui seront encore utilisés) à une itération donnée ? [1, 8, 81, 155]
- Quel est le nombre de défauts de cache générés par une boucle ? [41, 33, 64]
- Quel est le nombre d'éléments d'un tableau accédés avant un élément donné ? [97, 129]
- Quel est le nombre d'octets transmis entre deux tâches parallèles ? [28, 69, 135]
- Quel est le nombre de fois qu'une instruction est exécutée avant une itération donnée ? [138, 61]
- Quelle est la quantité de la mémoire dynamiquement allouée par un morceau de programme ? [29]

La réponse à ces différentes questions est formalisée par le nombre de solutions entières de systèmes d'inégalités linéaires et de leurs transformations par des fonctions affines entières. Le nombre de ces solutions est dual au nombre de points entiers de polytopes, éventuellement paramétrés, ou de leurs transformations. Ceci a motivé le développement des outils mathématiques permettant de répondre à de telles questions.

Bien que les algorithmes que nous proposons dans cette thèse (chapitres 3, 4 et 5) trouvent des applications dans plusieurs contextes, ils sont particulièrement utiles dans les méthodes d'optimisation des accès mémoire, de plus en plus nombreuses.

2.3 Méthodes d'optimisation des accès mémoire

L'optimisation du temps d'accès à la hiérarchie mémoire est un des grands soucis de la communauté d'optimisation de programmes. Ceci est principalement motivé par l'avènement des systèmes embarqués exigeant des mémoires plus petites et moins consommatrices en énergie électrique. De plus, l'écart entre les avancées hardware du processeur d'un coté et de la hiérarchie mémoire d'un autre ne cesse de grandir [123, 71].

Afin de faire face à la pénalité des accès à la mémoire centrale (MC), les ordinateurs modernes mettent en place une petite mémoire dite *cache* entre la MC et le processeur. La caractéristique principale des caches est qu'ils permettent un accès très rapide aux données qu'ils contiennent, et qu'ils ont une organisation particulière, par *lignes*. Malheureusement, leur taille relativement petite, ne permet pas de stocker toutes les données dont un programme a besoin. La solution consiste à faire des transferts de données entre la MC et le cache : toute donnée accédée par un programme est recherchée d'abord dans le cache. Si elle ne s'y trouve pas, elle sera recherchée dans la MC et copiée dans le cache. Bien évidement, ce transfert entre le cache et la MC, ne doit pas entraîner l'incohérence de données. À noter que la majorité des ordinateurs modernes assurent la gestion automatique de caches à placement direct, ou associatifs par ensembles de blocs de taille variable, et intégrant une politique d'éviction de lignes.

Lorsqu'une donnée accédée par un programme ne se trouve pas dans le cache, on dit qu'il y a eu un *défaut de cache* (*cache miss*). Les travaux récents [33, 19] classifient les défauts de caches en trois catégories principales :

- les défauts *froids* (*cold misses*) qui apparaissent lorsque une donnée est accédée pour la première fois.
- les défauts de *capacité* (*capacity misses*) qui apparaissent lorsque beaucoup de données (plus que la taille du cache) sont accédées entre utilisation et réutilisation de la même donnée.
- les défauts de *conflit* (*conflict misses*) qui concernent uniquement les caches à accès direct ou associatif par ensemble de blocs. Ce genre de défauts de caches apparaissent lorsque, entre utilisation et réutilisation d'une même donnée, une autre donnée ayant la même adresse (en cache) est accédée.

Le but ultime d'un grand nombre de méthodes d'optimisation des accès mémoire est de réduire, au maximum, le nombre de défauts de cache. En effet, plus les défauts de cache se produisent, plus le système est ralenti. La plupart de ces méthodes visent à améliorer la localité *temporelle* et/ou *spatiale*. La localité temporelle consiste à faire plusieurs utilisations consécutives d'une même donnée. Tandis que la localité spatiale consiste à utiliser plusieurs données adjacentes (placées dans la même ligne de cache).

Kristof Byels [19] a résumé les techniques d'optimisation de programmes, visant à améliorer les performances du cache, en quatre catégories principales :

Méthodes réduisant les défauts de capacité

Considérons un nid de boucles contenant une donnée D sur laquelle N calculs seront effectués, et que d'autres calculs (sur d'autres données) interviennent entre deux utilisations consécutives de cette donnée. Si le cache n'est pas suffisamment grand pour contenir toutes les données intermédiaires, la donnée D sera forcement expulsée du cache avant qu'elle ne soit utilisée une deuxième fois. Cela veut dire que le nombre de défauts de cache qui vont se produire est égal à N. Si une méthode d'optimisation de programmes arrive à changer l'ordre d'exécution des instructions (en préservant la sémantique du programme) de telle façon à ce que la donnée D soit réutilisée une deuxième fois avant qu'elle ne soit expulsée du cache, le nombre de défauts de cache sera réduit à $\frac{N}{2}$. Parmi les méthodes réduisant les défauts de capacité on peut citer : l'optimisation de la localité temporelle [103, 132, 152, 96] et l'optimisation de la localité

spatiale [3, 74, 97, 48].

Méthodes réduisant les défauts de conflit

Les méthodes réduisant les défauts de conflit concernent, bien évidemment, les caches à accès direct ou associatifs par ensemble de blocs. Le but de ces méthodes est d'éviter que plusieurs données de l'espace de travail ne soient positionnées dans un espace restreint dans le cache. En effet, deux données ayant la même adresse dans le cache, ne peuvent y coexister : la présence de l'une entraîne l'expulsion de l'autre et par conséquent, un défaut de conflit. Si une méthode d'optimisation arrive à disperser les adresses des données sur toutes les lignes du cache, le nombre de défauts de conflit sera significativement baissé. Parmi les méthodes réduisant les défauts de conflit on peut citer celles basées sur la transformation du placement de données en mémoire [7, 121, 122, 32], et celles basées sur le changement de la taille de la tuile (*tile*) [40, 85].

Méthodes de calcul parallèle pour cacher la latence mémoire

Lorsque les défauts de cache ne peuvent être évités, certaines méthodes d'optimisation proposent de faire d'autres calculs parallèlement à la recherche de données en mémoire. Dans cette catégorie de méthodes, on peut citer : les méthodes de *préchargement* (*prefetching*) et les méthodes de *multithreading*. Le préchargement [31, 109, 99, 36] consiste à charger prématurément dans le cache, des données qui seront utilisées dans le futur proche. Le défi de ces méthodes est, bien évidemment, de ne pas précharger des données qui seront expulsées du cache avant qu'elles ne soient utilisées. Le *multithreading* [51, 141] quant à lui, permet de passer rapidement à l'exécution d'un autre thread pendant que le premier attend l'arrivée de la donnée dont il a besoin. La difficulté du *multithreading* réside dans la gestion des threads (partage de tâches, traitement de signaux, synchronisation...). Dans des cas défavorables, le multithreading peut entraîner une perte en performances globales du programme.

Méthodes d'amélioration de la stratégie de remplacement

Une donnée accédée par un programme peut être chargée dans différentes lignes du cache (si le cache est associatif ou associatif par ensemble de blocs). Lorsque toutes les lignes susceptibles de contenir cette donnée sont occupées, le contenu de l'une d'entre elles doit être *remplacé* par celui de la MC contenant la donnée accédée. Il existe plusieurs stratégies de remplacement, les plus connues sont FIFO (First In First Out), LRU (Least Recently Used) et LFU (Least Frequently Used). Le nombre de défauts de cache peut être réduit en améliorant la stratégie de remplacement. Certaines méthodes d'optimisation [73, 140, 23, 24] permettent de déterminer quelles sont les données qui ne seront plus utilisées pendant un certain temps (relatif à la taille du cache). Cette information est utilisée par le contrôleur de cache pour remplacer prématurément les données concernées, peu importe si elles sont récemment chargées ou les plus fréquemment utilisées : on parle de cache contrôlable par programme. D'autres méthodes [20, 21] proposent de combiner les idées des différentes méthodes précédentes pour aboutir à de meilleurs résultats.

Le reste de ce chapitre est consacré à la présentation des notions et résultats fondamentaux sur lesquels sont basés nos travaux, à savoir les *polyèdres paramétrés* et les *quasi-polynômes d'Ehrhart.*

2.4 Polyèdres convexes

2.4.1 Définitions

Définition 1. L'enveloppe linéaire *d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$ est l'ensemble donné par la combinaison linéaire*

$$\text{lin } X = \left\{ \sum_i \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in X,\ \lambda_i \in \mathbb{R} \right\}. \tag{2.1}$$

Définition 2. L'enveloppe positive *d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$ est l'ensemble donné par la combinaison positive*

$$\text{pos } X = \left\{ \sum_i \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in X,\ \lambda_i \geq 0 \right\}. \tag{2.2}$$

Définition 3. L'enveloppe affine *d'un ensemble de vecteurs $X \in \mathbb{Q}^d$ est l'ensemble donné par la combinaison affine*

$$\text{aff } X = \left\{ \sum_i \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in X,\ \sum_i \lambda_i = 1 \right\}. \tag{2.3}$$

Définition 4. L'enveloppe convexe *d'un ensemble de vecteurs $X \subset \mathbb{Q}^d$ est l'ensemble donné par la combinaison convexe*

$$\text{conv } X = \{ \sum_i \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in X,\ \lambda_i \geq 0,\ \sum_i \lambda_i = 1 \}. \tag{2.4}$$

Définition 5. *Un* **polyèdre rationnel** *$P \subset \mathbb{Q}^d$ est un ensemble de vecteurs $\mathbf{x}$ de dimension d, définis par un système d'égalités et d'inégalités*

$$P = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq \mathbf{c} \ \wedge\ A'\mathbf{x} = \mathbf{c}' \right\}, \tag{2.5}$$

avec $A \in \mathbb{Z}^{m \times d}$, $A' \in \mathbb{Z}^{m' \times d}$, $\mathbf{c} \in \mathbb{Z}^m$ et $\mathbf{c}' \in \mathbb{Z}^{m'}$, où m est le nombre des inégalités et m' est le nombre des égalités.

Note 1. *Un polyèdre rationnel peut aussi être donné sous la forme*

$$P = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq \mathbf{c} \right\}. \tag{2.6}$$

En effet, toute équation $\langle \mathbf{a}, \mathbf{x} \rangle = c$ peut être remplacée par deux inégalités $\langle \mathbf{a}, \mathbf{x} \rangle \geq c$ et $\langle \mathbf{a}, \mathbf{x} \rangle \leq c$, où $\langle \cdot, \cdot \rangle$ est le produit scalaire de vecteurs. Une équation représentée par des inégalités est dite équation implicite.

Définition 6. *La* **dimension** (géométrique) *d'un polyèdre rationnel $P \subset \mathbb{Q}^d$ est la dimension de son enveloppe affine, ou d'une manière équivalente, la dimension d de son espace ambiant $\mathbb{Q}^d$ moins le nombre d'équations (implicites), linéairement indépendantes, du système le définissant (2.6). Un polyèdre de dimension d est souvent appelé un* d-polyèdre.

Définition 7. *Un polyèdre de* **dimension pleine** *est un polyèdre dont la dimension (géométrique) est égale à la dimension de son espace ambiant. En d'autre termes, un polyèdre est de dimension pleine si le système d'inégalités le définissant ne comporte pas d'équations implicites.*

Définition 8. *Un* **rayon** *d'un polyèdre rationnel P est un vecteur non nul $\mathbf{r}$, tel que $(\mathbf{x} + \mu\mathbf{r}) \in P$ quels que soient $\mathbf{x} \in P$ et $\mu \in \mathbb{Q}$ avec $\mu \geq 0$. Un rayon représente une direction vers laquelle le polyèdre va à l'infini.*

Définition 9. *Une* **ligne** *d'un polyèdre rationnel P est un vecteur non nul $\mathbf{l}$, tel que $(\mathbf{x} + \lambda\mathbf{l}) \in P$ quels que soient $\mathbf{x} \in P$ et $\lambda \in \mathbb{Q}$. Une ligne est aussi appelée* rayon bidirectionnel.

Définition 10. *Les* **sommets** *d'un polyèdre rationnel sont ses points extrêmes, c'est-à-dire ses seuls points qui ne peuvent pas être donnés par des combinaisons convexes de ses autres points.*

Définition 11. *Une* **face** *F d'un polyèdre rationnel P (2.6) est l'intersection de P avec $\{\, \mathbf{x} \in \mathbb{Q}^d \mid A'\mathbf{x} = \mathbf{c}' \,\}$, où $A'\mathbf{x} \geq \mathbf{c}'$ est un sous système de $A\mathbf{x} \geq \mathbf{c}$. Les* **facettes** *d'un polyèdre rationnel sont ses $(n-1)$-faces (faces de dimension $n-1$), où n est la dimension (géométrique) du polyèdre. Les faces de dimension* 0 *sont les* sommets *du polyèdre. Enfin, l'ensemble vide $\emptyset$ est, par convention, la face de dimension -1 de tous les polyèdres.*

2.4.2 Décomposition de polyèdres

Définition 12. *Un* **polytope rationnel** *est un polyèdre rationnel borné, donné par l'enveloppe convexe de ses sommets*

$$Q = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = \sum_i \nu_i \mathbf{v}_i,\ \mathbf{v}_i \in \mathcal{V},\ \nu_i \geq 0, \sum_i \nu_i = 1 \right\}, \tag{2.7}$$

où $\mathcal{V}$ est l'ensemble de sommets du polytope.

Définition 13. *Un* **cône polyédrique rationnel** *est un polyèdre rationnel donné par l'enveloppe positive d'un nombre* fini *de vecteurs, dit* générateurs *du cône.*

$$C = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = \sum_i \mu_i \mathbf{g}_i,\ \mathbf{g}_i \in \mathcal{G},\ \mu_i \geq 0 \right\}, \tag{2.8}$$

où $\mathcal{G}$ est l'ensemble des vecteurs générateurs du cône. Un cône polyédrique peut aussi être donné par un système d'inégalités homogènes $\{\mathbf{x} \mid A\mathbf{x} \geq 0\}$ [126].

Propriété 1. *Un cône polyédrique rationnel peut être exprimé en fonction de l'enveloppe linéaire de ses lignes et l'enveloppe positive de ses rayons*

$$C = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = \sum_i \lambda_i \mathbf{l}_i + \sum_j \mu_j \mathbf{r}_j,\ \mathbf{l}_i \in \mathcal{L},\ \mathbf{r}_i \in \mathcal{R},\ \mu_j, \geq 0 \right\}, \tag{2.9}$$

où, $\mathcal{L}$ est l'ensemble de lignes du cône et $\mathcal{R}$ l'ensemble de ses rayons. Cette représentation de cônes est dite explicite *ou* paramétrée*. Lorsque l'ensemble $\mathcal{L}$ est vide, le cône est dit* pointé*, i.e., un polyèdre à un seul sommet (l'origine $\mathcal{O}$). La translation d'un cône pointé est aussi appelée* cône*.*

Théorème 1. (Théorème de décomposition de polyèdres [Motzkin 1936]) *Un ensemble P de vecteurs de l'espace euclidien est un polyèdre si et seulement si $P = Q + C$, pour un certain polytope Q et un certain cône polyédrique C, dit* cône caractéristique de P*.*

La preuve de ce théorème est donnée dans [126], page 88.

D'après la propriété et le théorème ci-dessus, il découle que tout polyèdre rationnel peut être décomposé en une enveloppe linéaire de ses lignes, une enveloppe positive de ses rayons et une enveloppe convexe de ses sommets

$$P = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = \sum_i \lambda_i \mathbf{l}_i + \sum_j \mu_j \mathbf{r}_j + \sum_k \nu_k \mathbf{v}_k,\ \mu_j, \nu_k \geq 0, \sum_k \nu_k = 1 \right\}, \tag{2.10}$$

avec $\mathbf{l}_i \in \mathcal{L}$, $\mathbf{r}_i \in \mathcal{R}$ et $\mathbf{v}_k \in \mathcal{V}$, où $\mathcal{L}$, $\mathcal{R}$ et $\mathcal{V}$ sont respectivement les ensembles de lignes, rayons et sommets du polyèdre. Cette représentation des polyèdres est dite *représentation de Minkowski* [105] ou *représentation paramétrée* [126]. Un polyèdre ne possédant pas de lignes est un polyèdre *pointé* et un polyèdre ne possédant ni ligne ni rayon est un *polytope*.

La représentation de Minkowski d'un polyèdre peut aussi être donnée sous la forme matricielle

$$P = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = L\boldsymbol{\lambda} + R\boldsymbol{\mu} + V\boldsymbol{\nu},\ \mu_j, \nu_k \geq 0, \sum_k \nu_k = 1 \right\}, \tag{2.11}$$

où L, R et V sont les matrices dont les colonnes sont, respectivement, les lignes, les rayons et les sommets du polyèdre.

Les deux représentations de polyèdres (2.6 et 2.11) sont *duales*. Le passage d'une représentation à l'autre peut se faire par la méthode de double description de Motzkin [108], raffinée par Chernikova [34], Rubin [124], Fernandez et Quinton [58] puis Le Verge [86]. Cette méthode est implémentée dans la librairie polyédrique `Polylib` [151, 95, 94][1]. L'algorithme de Chernikova ne peut être appliqué qu'à des cônes. Afin de l'appliquer à un polyèdre quelconque, il est nécessaire de pouvoir associer à tout polyèdre un cône unique. Cela est possible en utilisant la représentation *homogène* des polyèdres dont l'idée est l'ajout d'une dimension artificielle [105, 106].

[1] La `Polylib` est la librairie que nous utilisons pour effectuer les opérations sur les polyèdres.

2.4.3 Représentation homogène de polyèdres

La représentation *homogène* [65, 151, 98] permet de représenter un polyèdre par un système d'inégalités *homogènes*. Cette forme est mieux adaptée aux opérations sur les polyèdres implémentées dans la `Polylib`. La représentation homogène est obtenue en appliquant la transformation $\mathbf{x} \rightarrow \begin{pmatrix} \xi\mathbf{x} \\ \xi \end{pmatrix}, \xi \geq 0$ au système non homogène représentant le polyèdre. Ainsi, la représentation homogène du polyèdre (2.6) est :

$$\widehat{P} = \left\{ \begin{pmatrix} \xi\mathbf{x} \\ \xi \end{pmatrix} \in \mathbb{Q}^{d+1} \middle| \widehat{A} \begin{pmatrix} \xi\mathbf{x} \\ \xi \end{pmatrix} \geq 0 \right\}, \tag{2.12}$$

avec $\widehat{A} = \left[\begin{array}{c|c} A & -\mathbf{c} \\ \hline \mathbf{0} & 1 \end{array}\right]$. Noter que le polyèdre original P (2.6) peut être obtenu en faisant l'intersection de $\widehat{P}$ avec l'hyperplan d'équation $\xi = 1$.

La représentation homogène de la forme de Minkowski (2.17) est :

$$\widehat{P} = \left\{ \begin{pmatrix} \xi\mathbf{x} \\ \xi \end{pmatrix} \in \mathbb{Q}^{d+1} \middle| \begin{pmatrix} \xi\mathbf{x} \\ \xi \end{pmatrix} = \widehat{L}\boldsymbol{\lambda}' + \widehat{R}\boldsymbol{\mu}',\ \mu'_i \geq 0 \right\}, \tag{2.13}$$

avec $\widehat{L} = \left[\begin{array}{c} L \\ \hline \mathbf{0} \end{array}\right]$ et $\widehat{R} = \left[\begin{array}{c|c} R & V \\ \hline \mathbf{0} & \mathbf{1} \end{array}\right]$. Remarquer que cette représentation transforme les sommets en rayons. Cela simplifie considérablement la manipulation des polyèdres qui sont maintenant des cônes [94].

2.5 Polyèdres convexes paramétrés

Les *polyèdres paramétrés* sont, par définition, des polyèdres qui dépendent d'un certain nombre de paramètres, quelle que soit la nature de cette dépendance. Cependant, cette appellation est, le plus souvent, utilisée pour désigner les polyèdres *linéairement* paramétrés. Ce sont des polyèdres dont la partie constante dépend linéairement des paramètres. Noter que dans certains travaux récents [67, 119], le nom *polyèdre non-linéairement paramétré* est utilisé pour désigner un polyèdre dont les coefficients des variables peuvent dépendre des paramètres.

Dans le cadre de cette thèse, nous nous intéressons aux polyèdres linéairement paramétrés, que nous appelons simplement polyèdres paramétrés. La classe de polyèdres étudiés est de la forme :

$$P_{\mathbf{p}} = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq B\mathbf{p} + \mathbf{c} \right\}, \tag{2.14}$$

avec $A \in \mathbb{Z}^{m \times d}, B \in \mathbb{Z}^{m \times n}$, $\mathbf{c} \in \mathbb{Z}^m$ et $\mathbf{p} \in \mathbb{Z}^n$ ($\mathbf{p}$ est un vecteur de n paramètres). Nous rappelons que le système d'inégalités ci-dessus peut comporter des équations implicites, i.e., des couples d'inégalités équivalents à des équations.

Un polyèdre paramétré peut aussi être donné dans l'espace *combiné* des variables et des paramètres sous la forme :

$$P = \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} \in \mathbb{Q}^{d+n} \middle| \ [A \mid -B] \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} \geq \mathbf{c} \right\}. \tag{2.15}$$

Enfin, la forme combinée homogène d'un polyèdre paramétrée est donnée par :

$$P = \left\{ \begin{pmatrix} \xi\mathbf{x} \\ \xi\mathbf{p} \\ \xi \end{pmatrix} \in \mathbb{Q}^{d+n+1} \;\middle|\; \left[\begin{array}{c|c|c} A & -B & -\mathbf{c} \\ \hline \mathbf{0} & \mathbf{0} & 1 \end{array}\right] \begin{pmatrix} \xi\mathbf{x} \\ \xi\mathbf{p} \\ \xi \end{pmatrix} \geq 0 \right\}. \tag{2.16}$$

C'est cette dernière forme qui est implémentée dans la `Polylib`.

Théorème 2. (Loechner-Wilde [98, 94]) *La représentation de Minkowski, sous forme de lignes, rayons et sommets d'un polyèdre paramétré est de la forme :*

$$P_{\mathbf{p}} = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = L\boldsymbol{\lambda} + R\boldsymbol{\mu} + V(\mathbf{p})\boldsymbol{\nu},\ \mu_j, \nu_k \geq 0, \sum_k \nu_k = 1 \right\}, \tag{2.17}$$

où L et R, les matrices contenant les lignes et les rayons du polyèdre, sont des matrices entières constantes (ne dépendant pas des paramètres $\mathbf{p}$), et $V(\mathbf{p})$, la matrice contenant les sommets du polyèdre, est une matrice rationnelle dont chaque coefficient est une fonction affine des paramètres $\mathbf{p}$, définie sur un certain domaine de validité de $\mathbf{p}$.

La preuve de ce théorème est donnée dans [98, 94].

Dans ce qui suit, nous présentons certains concepts de base pour le comptage de points entiers dans des polyèdres paramétrés. Bien évidemment, seuls les *polyèdres bornés* (polytopes) sont pris en compte (ceux qui ne le sont pas, contiennent une infinité de points entiers).

2.5.1 Les sommets d'un polytope paramétré

Nous allons voir, un peu plus loin, que les coordonnées des sommets d'un polytope jouent un rôle fondamental pour les méthodes de dénombrement des points entiers. Dans le cas d'un polytope *non paramétré*, les coordonnées des sommets sont simplement des constantes. Celles-là peuvent être directement calculées par l'algorithme de Chernikova. Lorsque le polytope est paramétré, les coordonnées des sommets peuvent être des fonctions affines des paramètres (théorème 2). Dans ce cas, un sommet (paramétré) peut être valide (actif) pour *seulement* un sous-ensemble des valeurs de paramètres, dit *domaine de validité* du sommet.

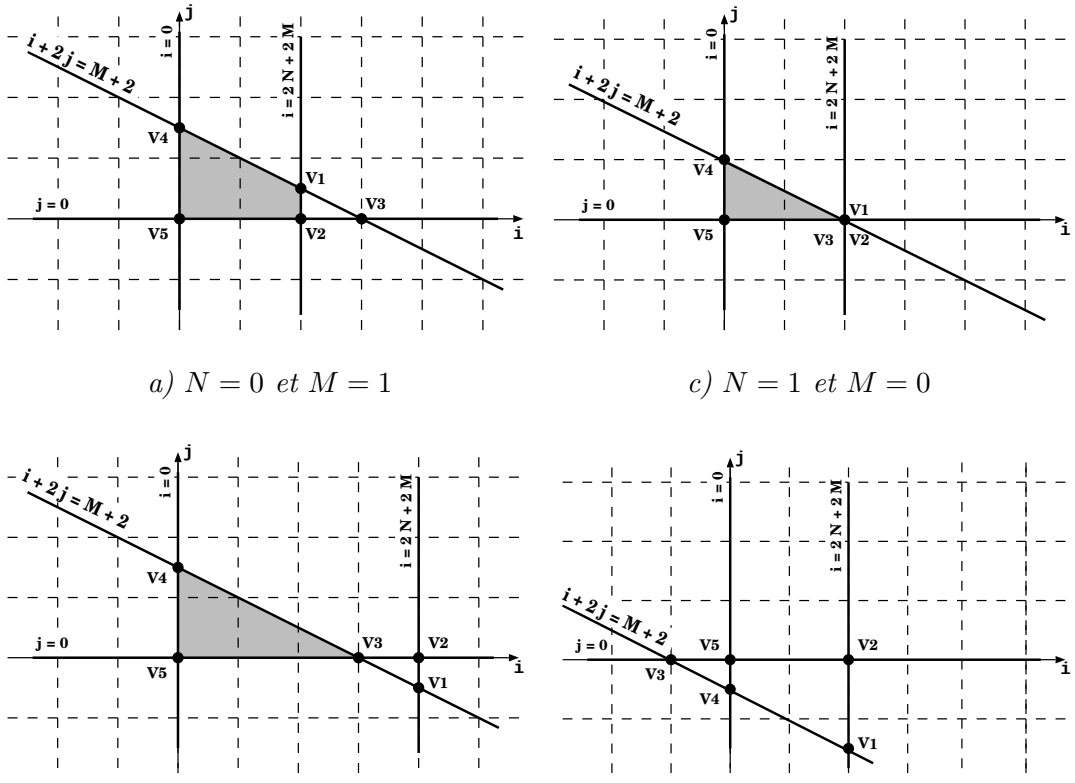
Exemple 2. Le polytope paramétré de la figure 2.2 est donné par le système d'inégalités linéaires paramétrées suivant :

$$P_{N,M} = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid i \geq 0 \wedge j \geq 0 \wedge i \leq 2N + 2M \wedge i + 2j \leq M + 2 \right\}. \tag{2.18}$$

La forme matricielle (2.14) correspondante est :

$$P_{N,M} = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ -1 & -2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \geq \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -2 & -2 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} N \\ M \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ -2 \end{pmatrix} \right\}.$$

La figure 2.2 montre que pour certaines valeurs de paramètres, seulement un sous-ensemble de sommets sont actifs (valides). En fonction des sommets valides, le polytope change de forme. Les sommets qui changent de position ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ et $\mathbf{v}_4$) sont paramétrés, i.e., définis par des fonctions affines des paramètres.



a) $N = 0$ *et* $M = 1$

c) $N = 1$ *et* $M = 0$

b) $N = 1$ *et* $M = 1$

d) $N = 4$ *et* $M = -3$

FIG. 2.2 – Exemples de formes (trapèze, triangle ou vide) d'un polytope paramétré, en fonction des valeurs de ses paramètres.

Calcul des sommets paramétrés et leurs domaines de validité

Loechner et Wilde [98, 94] ont proposé un algorithme de calcul des sommets d'un polytope paramétré et leurs domaines de validité. Le principe de cet algorithme, tel qu'il est implémenté dans la *PolyLib*, est le suivant :

1. Calculer toutes les *n-faces* du polyèdre paramétré (sous sa forme combinée homogène (2.16)), où n est la dimension de l'espace de paramètres. Les n-faces sont elles mêmes des polyèdres (de dimension n) représentés par un ensemble de lignes, rayons et sommets. Chaque n-face est donc donnée par une matrice M_i dont les colonnes sont ses lignes, rayons et sommets ;
2. Pour toute *n-face* F_i^n, représentée par la matrice M_i :
   - calculer B_i la matrice base de M_i (B_i contient les colonnes de M_i qui sont linéairement indépendantes),
   - calculer B_{id} (resp. B_{ip}) la projection de B_i sur l'espace de données (resp. de paramètres),
   - si B_{ip} est inversible, le sommet est valide et vaut $B_{id} \times B_{ip}^{-1}$ sur un domaine de validité égal à la projection de la *n-face* F_i^n sur l'espace de paramètres.

Pour des raisons de simplicité, nous utilisons une terminologie différente de celle utilisée par l'algorithme original pour expliquer, plus en détail, comment les sommets paramétrés et leurs domaines de validité sont calculés.

Considérons, sans perte de généralité[2], un d-polytope paramétré de dimension pleine :

$$P_{\mathbf{p}} = \left\{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq B\mathbf{p} + \mathbf{c} \right\}.$$

Les sommets du polytope $P_{\mathbf{p}}$ correspondent à ses 0-faces. Cela veut dire que chaque sommet paramétré $\mathbf{v}_i(\mathbf{p})$ de $P_{\mathbf{p}}$ est donné par l'intersection de $P_{\mathbf{p}}$ et un certain ensemble $\{ \mathbf{x} \in \mathbb{Q}^d \mid A'_i\mathbf{x} = B'_i\mathbf{p} + \mathbf{c}'_i \}$, où $A'_i\mathbf{x} \geq B'_i\mathbf{p} + \mathbf{c}'_i$ est un sous-système de $A\mathbf{x} \geq B\mathbf{p} + \mathbf{c}$, tel que la dimension de l'intersection est 0. Nous pouvons écrire :

$$\mathbf{v}_i(\mathbf{p}) = \{ \mathbf{x} \in \mathbb{Q}^d \mid A'_i\mathbf{x} = B'_i\mathbf{p} + \mathbf{c}'_i \wedge A''_i\mathbf{x} \geq B''_i\mathbf{p} + \mathbf{c}''_i \}, \tag{2.19}$$

où $A''_i\mathbf{x} \geq B''_i\mathbf{p} + \mathbf{c}''_i$ est le sous-système constitué par le reste des inégalités de $P_{\mathbf{p}}$. Puisque $P_{\mathbf{p}}$ est de dimension d, au moins d égalités sont requises dans le système $A'_i\mathbf{x} \geq B'_i\mathbf{p}+\mathbf{c}'_i$. En effet, tous les sommets peuvent être obtenus en faisant l'intersection de $P_{\mathbf{p}}$ avec toutes les combinaisons possibles de d égalités linéairement indépendantes, c'est-à-dire tel que la matrice A'_i est inversible. Le sommet $\mathbf{v}_i(\mathbf{p})$ correspond à la solution du système $A'_i\mathbf{x} = B'_i\mathbf{p} + \mathbf{c}'_i$. Il en découle que $\mathbf{v}_i(\mathbf{p})$ est un vecteur, de dimension d, de fonctions affines de paramètres. $\mathbf{v}_i(\mathbf{p})$ est un sommet *valide* de $P_{\mathbf{p}}$ seulement pour les valeurs de paramètres pour lesquelles l'ensemble (2.19) n'est pas vide. Autrement dit, pour les valeurs de paramètres qui appartiennent au polyèdre suivant :

$$R_i = \left\{ \mathbf{p} \in \mathbb{Q}^n \mid A''_i\mathbf{v}_i(\mathbf{p}) \geq B''_i\mathbf{p} + \mathbf{c}''_i \right\}.$$

Soit m le nombre de contraintes du système original. Puisque le nombre d'ensembles de d contraintes, linéairement indépendantes, du système original est au plus égal à la combinaison $\binom{m}{d}$, le nombre total de sommets paramétrés est polynomial en nombre de contraintes (pour une dimension fixée) et, par conséquent, polynomial en taille d'entrée (en nombre de bits).

Exemple 3. Considérons le polytope de l'exemple 2 :

$$P_{N,M} = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid i \geq 0 \wedge j \geq 0 \wedge i \leq 2N + 2M \wedge i + 2j \leq M + 2 \right\}.$$

$P_{N,M}$ est un polytope paramétré de dimension 2. Par conséquent, chacun de ses sommets est donné par une intersection, de dimension 0, de deux de ces facettes. Considérons, par exemple, le calcul du sommet $\mathbf{v}_1$ correspondant aux deux facettes $i = 2N + 2M$ et $i + 2j = M + 2$. $\mathbf{v}_1$ est la solution du système constitué par ces deux équations. Ce qui donne

$$\mathbf{v}_1 = \begin{pmatrix} 2N + 2M \\ -N - \frac{M}{2} + 1 \end{pmatrix}.$$

Le sommet $\mathbf{v}_1$ est valide s'il satisfait le reste des contraintes du polytope, i.e., $\{ i \geq 0 \wedge j \geq 0 \}$. En substituant $i = 2N + 2M$ et $j = -N - \frac{M}{2} + 1$ dans ces contraintes, on

[2]Tout polytope peut être transformé en un polytope de dimension pleine ayant le même nombre de points entiers.

obtient le polyèdre $R_{\mathbf{v}_1}$, contenant les valeurs de paramètres pour lesquelles le sommet $\mathbf{v}_1$ est valide :

$$R_{\mathbf{v}_1} = \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid N + M \geq 0 \wedge -2N - M \geq -2 \right\}.$$

En procédant d'une manière similaire sur les autres paires de facettes, on obtient le reste des sommets :

$$\mathbf{v}_2 = \begin{pmatrix} 2N + 2M \\ 0 \end{pmatrix},\ \mathbf{v}_3 = \begin{pmatrix} M + 2 \\ 0 \end{pmatrix},\ \mathbf{v}_4 = \begin{pmatrix} 0 \\ \frac{M}{2} + 1 \end{pmatrix} \text{ et } \mathbf{v}_5 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Ces sommets sont valides pour les valeurs de paramètres qui appartiennent respectivement aux polyèdres $R_{\mathbf{v}_2}, R_{\mathbf{v}_3}, R_{\mathbf{v}_4}$ et $R_{\mathbf{v}_5}$, avec

$$\begin{aligned}
R_{\mathbf{v}_2} &= \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid N + M \geq 0 \wedge -2N - M \geq -2 \right\}, \\
R_{\mathbf{v}_3} &= \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid M \geq -2 \wedge 2N + M \geq 2 \right\}, \\
R_{\mathbf{v}_4} &= \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid N + M \geq 0 \wedge M \geq -2 \right\}, \\
R_{\mathbf{v}_5} &= \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid N + M \geq 0 \wedge M \geq -2 \right\}.
\end{aligned}$$

La figure 2.2a montre le polytope $P_{N,M}$ et ses sommets lorsque $N = 0$ et $M = 1$. Le sommet $\mathbf{v}_3$ n'est pas valide pour ces valeurs de paramètres, puisque $(0, 1) \notin R_{\mathbf{v}_3}$. Pour ces valeurs de paramètres, le polytope a la forme d'un trapèze $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4, \mathbf{v}_5\}$. De même, la figure 2.2b montre le polytope lorsque $N = 1$ et $M = 1$. Les deux sommets $\mathbf{v}_1$ et $\mathbf{v}_2$ ne sont pas actifs, puisque $(1, 1) \notin R_{\mathbf{v}_1} \cup R_{\mathbf{v}_2}$. Dans ce cas le polytope a la forme d'un triangle $\{\mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5\}$. La figure 2.2c montre que pour $N = 1$ et $M = 0$, les trois sommets $\mathbf{v}_1$, $\mathbf{v}_2$, et $\mathbf{v}_3$ ont les mêmes coordonnées. Enfin, la figure 2.2d montre que lorsque $N = 4$ et $M = -3$, aucun des sommets n'est valide, i.e., le polytope est vide.

2.5.2 Décomposition des domaines de validité de sommets

Définition 14. *Soit $P_{\mathbf{p}}$ un polytope paramétré, avec $\mathbf{p} = (p_1, p_2, \cdots, p_n)$. $P_{\mathbf{p}}$ est dit* **homothétique-bordé** *s'il est tel que $\forall p_i,\ (1 \leq i \leq n)$, $\exists p_{i,min}, p_{i,max} \in \mathbb{Z}$ tel que $\forall p_i$: $p_{i,min} < p_i \leq p_{i,max}$, $P_{p_{1,min},\cdots,p_{i-1,min},\mathbf{p_i},p_{i+1,min},\cdots,p_{n,min}}$ et le résultat de l'homothétie de $P_{p_{1,min},\cdots,p_{i-1,min},p_{i,min},p_{i+1,min},\cdots,p_{n,min}}$ de centre l'origine du repère O et de rapport p_i. Pour un paramètre unique p, ceci correspond à la dilatation pP d'un polytope non paramétré P.*

Clauss et Loechner [39, 94] ont montré que tout polyèdre paramétré est homothétique-bordé par domaine de validité.

Pour pouvoir compter les points entiers d'un polytope paramétré, il faut qu'il soit homothétique-bordé. Si ce n'est pas le cas, il doit être décomposé en un ensemble de polytopes homothétiques-bordés. Chacun d'entre eux est défini sur un sous-ensemble

de valeurs des paramètres, dit *domaine de validité* du polytope. À titre d'exemple, pour le polytope de la figure 2.2, il s'agit de séparer les valeurs des paramètres pour lesquelles le polytope est un triangle homothétique-bordé de celles pour lesquelles il est un trapèze homothétique-bordé. Bien évidement, le reste des valeurs correspond au cas où le polytope est vide.

Clauss et Loechner [39] ont proposé un algorithme de calcul de domaines de validité *disjoints*[3] de polytopes paramétrés. Cet algorithme maintient une liste (R_i) de régions de dimension n (n est la dimension de l'espace de paramètres). Les intersections, deux à deux, de ces régions sont au plus de dimension $n-1$. À chaque région R_i est associée la liste des sommets paramétrés $\mathcal{V}_{R_i}$ qui y sont valides. Initialement, la liste contient une paire unique $(R_{\mathbf{v}_1}, \{\, \mathbf{v}_1 \,\})$ (un sommet quelconque et son domaine de validité). À chaque étape, un nouveau sommet $\mathbf{v}_j$ et son domaine de validité $R_{\mathbf{v}_j}$ sont considérés. Pour toute région R_i dans la liste courante, si $\dim(R_i \cap R_{\mathbf{v}_j}) = n$, la paire $(R_i, \mathcal{V}_{R_i})$ est remplacée par deux nouvelles paires $(R_i \cap R_{\mathbf{v}_j}, \mathcal{V}_{R_i} \cup \{\, \mathbf{v}_j \,\})$ et $(\overline{R_i \setminus R_{\mathbf{v}_j}}, \mathcal{V}_{R_i})$, où $\overline{S}$ est la clôture de S. Après parcours de toute la liste, la paire $(\overline{R_{\mathbf{v}_j} \setminus \bigcup_i R_i}, \{\, \mathbf{v}_j \,\})$ est ajoutée. Bien évidemment, si une des différences est vide, la paire correspondante ne sera pas ajoutée.

Noter que certaines régions intermédiaires peuvent être non pas des polyèdres, mais des unions de polyèdres. Les régions finales sont des polyèdres.

Exemple 4. Nous allons maintenant calculer les domaines de validité du polytope paramétré illustré par la figure 2.2.

Remarquer que $R_{\mathbf{v}_1}$ et $R_{\mathbf{v}_2}$ correspondent au même domaine de validité (voir exemple 3), nous notons ce domaine R_1. $R_{\mathbf{v}_4}$ et $R_{\mathbf{v}_5}$ correspondent également à un même domaine que nous notons R_3. Enfin, $R_{\mathbf{v}_3}$ est noté par R_2. À noter aussi que $R_3 = R_1 \cup R_2$. Nous introduisons R_1, R_2 et R_3 pour faciliter la présentation de cet exemple.

D'abord, la liste de couples (domaine de validité, ensemble de sommets valides) est initialisée

$$((R_1, \{\, \mathbf{v}_1 \,\}))\,.$$

Puisque $R_{\mathbf{v}_2} = R_1$ et $R_1 \setminus R_1$ est évidemment vide, la première itération résulte en

$$((R_1, \{\, \mathbf{v}_1, \mathbf{v}_2 \,\}))\,.$$

L'itération suivante rajoute la paire $(R_2, \{\, \mathbf{v}_3 \,\})$, puisque $R_{\mathbf{v}_3} = R_2$ et $\dim(R_1 \cap R_2) < 2$.

$$((R_1, \{\, \mathbf{v}_1, \mathbf{v}_2 \,\}), (R_2, \{\, \mathbf{v}_3 \,\}))\,.$$

La troisième itération rajoute simplement le sommet $\mathbf{v}_4$ aux deux domaines R_1 et R_2, puisque $R_{\mathbf{v}_4} = R_1 \cup R_2$.

$$((R_1, \{\, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4 \,\}), (R_2, \{\, \mathbf{v}_3, \mathbf{v}_4 \,\}))\,.$$

Enfin, la dernière étape étape est similaire à la troisième. Elle résulte en

$$((R_1, \{\, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4, \mathbf{v}_5 \,\}), (R_2, \{\, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5 \,\}))\,.$$

[3] Les domaines de validités sont disjoints au sens large : ils peuvent tout de même avoir des faces communes de dimensions inférieures.

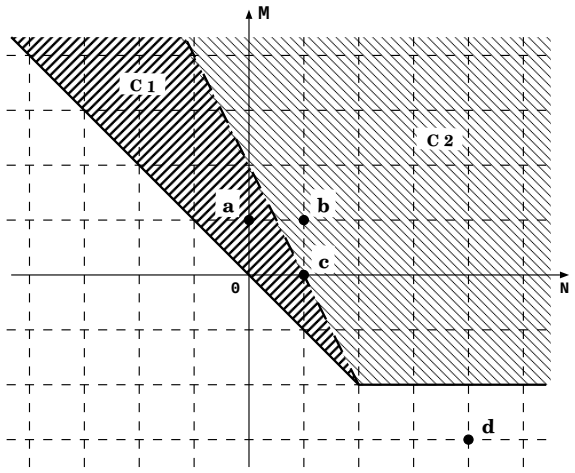


FIG. 2.3 – Les domaines de validité d'un polytope paramétré.

Le résultat final est donc donné par deux domaines de validité $C_1 = R_1$ et $C_2 = R_2$, avec leurs ensembles correspondants de sommets valides $\mathcal{V}_{C_1}$ et $\mathcal{V}_{C_2}$:

$$\left(C_1 = \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid N + M \geq 0 \wedge -2N - M \geq -2 \right\},\right.$$
$$\left. \mathcal{V}_{C_1} = \left\{ \begin{pmatrix} 2N + 2M \\ -N - \frac{M}{2} + 1 \end{pmatrix}, \begin{pmatrix} 2N + 2M \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \frac{M}{2} + 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\} \right),$$

$$\left(C_2 = \left\{ \begin{pmatrix} N \\ M \end{pmatrix} \in \mathbb{Q}^2 \mid M \geq -2 \wedge 2N + M \geq 2 \right\},\right.$$
$$\left. \mathcal{V}_{C_2} = \left\{ \begin{pmatrix} M + 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \frac{M}{2} + 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\} \right).$$

La figure 2.3 montre les régions dans l'espace de paramètres (domaines de validité) C_1 et C_2, où le polytope préserve une forme unique pour toutes les valeurs de paramètres à l'intérieur d'une même région. Dans cette figure, les points $a(0, 1)$, $b(1, 1)$, $c(1, 0)$ et $d(4, -3)$ correspondent respectivement aux configurations des figures 2.2a, 2.2b, 2.2c et 2.2d.

2.6 Quasi-polynômes d'Ehrhart

Le comptage de points entiers dans des polytopes paramétrés est souvent une étape clef pour résoudre un grand nombre de problèmes. Plusieurs disciplines ont montré leur besoin permanent de ce type de comptage. Nous reviendrons plus en détail dans les prochains chapitres, sur les applications du comptage de points entiers. Dans la présente section, nous nous contentons d'introduire les travaux fondamentaux qui ont servi de support aux travaux récents et, bien entendu, à nos contributions.

La première contribution fondamentale dans ce contexte est due au mathématicien *Eugène Ehrhart*. Son travail considère le cas particulier d'un polytope dont les contraintes linéaires dépendent d'un *seul* paramètre *positif* p[4]. Il a, notamment, montré que le nombre de points entiers dans la dilatation pP d'un polytope P ($pP = \{p\mathbf{x} \mid \mathbf{x} \in P,\ p \in \mathbb{N}\}$) peut être représenté par un polynôme en n dont les coefficients sont des nombres périodiques [52].

2.6.1 La théorie d'Ehrhart

Définition 15. *Un* **nombre périodique rationnel** *(unidimensionnel) $U(p)$ est une fonction $\mathbb{Z} \to \mathbb{Q}$, telle qu'il existe une période $q \in \mathbb{N}$, telle que $U(p) = U(p')$ quel que soit $p \equiv p' \bmod q$.*

Définition 16. *Un* **nombre n-périodique rationnel** *$U(\mathbf{p})$ est une fonction $\mathbb{Z}^n \to \mathbb{Q}$, telle qu'il existe une multi-période $\mathbf{q} = (q_1, \ldots, q_n) \in \mathbb{N}^n$, telle que $U(\mathbf{p}) = U(\mathbf{p}')$ quel que soit $p_i \equiv p'_i \bmod q$, pour $1 \leq i \leq n$. Le plus petit commun multiple* (ppcm) *des q_i est la période de $U(\mathbf{p})$.*

Ehrhart [53] représente un nombre périodique (unidimensionnel) $U(p)$ de période q par une liste (entre crochets) de toutes ses q valeurs possibles. Cette représentation s'applique aussi aux nombres périodiques multidimensionnels.

Exemple 5. $U(p_1, p_2) = \left(\cos\left(p_1.\frac{\pi}{2}\right)\right)^{p_2} = \left[\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{array}\right]_{p_1,p_2}$ est un nombre 2-périodique de multi-période $\mathbf{q} = (2, 4)$. Il est interprété comme suit :

$$U(p_1,p_2) = \begin{cases} [\ 1,\ 0,\ 1,\ 0\ ]_{p_2} \text{ si } p_1 \equiv 0 \bmod 2 \\ \quad = \begin{cases} 1 \text{ si } p_1 \equiv 0 \bmod 2 \text{ et } p_2 \equiv 0 \bmod 4 \\ 0 \text{ si } p_1 \equiv 0 \bmod 2 \text{ et } p_2 \equiv 1 \bmod 4 \\ 1 \text{ si } p_1 \equiv 0 \bmod 2 \text{ et } p_2 \equiv 2 \bmod 4 \\ 0 \text{ si } p_1 \equiv 0 \bmod 2 \text{ et } p_2 \equiv 3 \bmod 4 \end{cases} \\ [\ 1,\ 0,\ -1,\ 0\ ]_{p_2} \text{ si } p_1 \equiv 1 \bmod 2 \\ \quad = \begin{cases} 1 \text{ si } p_1 \equiv 1 \bmod 2 \text{ et } p_2 \equiv 0 \bmod 4 \\ 0 \text{ si } p_1 \equiv 1 \bmod 2 \text{ et } p_2 \equiv 1 \bmod 4 \\ -1 \text{ si } p_1 \equiv 1 \bmod 2 \text{ et } p_2 \equiv 2 \bmod 4 \\ 0 \text{ si } p_1 \equiv 1 \bmod 2 \text{ et } p_2 \equiv 3 \bmod 4 \end{cases} \end{cases}$$

Noter qu'un nombre périodique multidimensionnel peut être représenté par un ensemble de nombres périodiques unidimensionnels *imbriqués*. Par exemple :

$$U(p_1,p_2) = \left[\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{array}\right]_{p_1,p_2} = \left[\ [1,\,0,\,1,\,0]_{p_2},\ [1,\,0,\,-1,\,0]_{p_2}\ \right]_{p_1}.$$

Définition 17. *Un* **quasi-polynôme** *(univariable) de période q est une fonction $f : \mathbb{Z} \to \mathbb{Q}$, telle qu'il existe une liste de q polynômes $g_i (1 \leq i \leq q)$, telle que $f(p) = g_i(p)$ si $p \equiv i \bmod q$. Un quasi-polynôme (univariable) de degré d est de la forme :*

$$f(p) = c_d(p)p^d + \cdots + c_1(p)p + c_0,$$

[4] Ehrhart a aussi considéré certains problèmes avec deux paramètres [53].

où chaque $c_i(p)$ $(0 \leq i \leq d)$ *est un nombre périodique. Le* ppcm *des périodes des coefficients* $c_i(p)$ *est la* période *du quasi-polynôme.*

Définition 18. *Un* **quasi-polynôme** *(multivariable) en* $\mathbf{p} = (p_1, \ldots, p_n)$ *de degré* d *est une fonction* $f : \mathbb{Z}^n \to \mathbb{Q}$ *définie par :*

$$f(\mathbf{p}) = \sum_{i_1=0}^{d} \cdots \sum_{i_n=0}^{d} c_{i_1,\cdots,i_n} p_1^{i_1} \cdots p_n^{i_n}, \tag{2.20}$$

où, chaque $c_{i_1,\cdots,i_n}$ *est un nombre périodique multidimensionnel (au plus* n*-périodique). Le* ppcm *des périodes des coefficients* $c_{i_1,\cdots,i_n}$ *est la* période *du quasi-polynôme.*

Définition 19. *Le* **dénominateur d'un polytope** *(paramétré) est le plus petit commun multiple* (ppcm) *de ses* dénominateurs relatifs aux paramètres*, où le dénominateur relatif à un paramètre* p *d'un polytope est le* ppcm *des coefficients de* p *qui apparaissent dans les expressions affines des sommets (pour toute valeur entière des paramètres).*

Définition 20. *Un* **polytope (paramétré) entier** *est un polytope dont le dénominateur est égal à 1. Autrement dit, toutes les coordonnées de ses sommets sont des fonctions affines* entières *des paramètres.*

Théorème 3. (Théorème fondamental d'Ehrhart) *Le nombre de points entiers dans la dilatation* pP *d'un* d*-polytope rationnel* P $(pP = \{p\mathbf{x} \mid \mathbf{x} \in P,\ p \in \mathbb{N}\})$ *est donné par un quasi-polynôme en* p *de degré* d*. Le coefficient du terme* p^d *est égal au volume de* P *(indépendant de* p*). La période du quasi-polynôme est au plus égale au dénominateur du polytope* pP*. Lorsque* pP *est un polytope entier, le nombre de ses points entiers est donné par un simple polynôme.*

Ce théorème a été étendu pour un polytope paramétré quelconque par Clauss [37].

Théorème 4. (Théorème étendu de Clauss) *Le nombre de points entiers dans un* d*-polytope paramétré quelconque* $P_{\mathbf{p}}$ $(\mathbf{p} = (p_1, \ldots, p_n) \in \mathbb{Z}^n)$ *peut être donné par un ensemble fini de quasi-polynômes multivariable en* $\mathbf{p}$ *de degré* d*. Chaque quasi-polynôme est valide sur un sous ensemble des valeurs des paramètres, dit* domaine de validité*. La période relative à un paramètre* p *du quasi-polynôme sur un domaine de validité donné est égale au dénominateur relatif à* p *du polytope (sur le domaine considéré). Lorsque le polytope est entier, sur un domaine de validité donné, le nombre de points entiers correspondant est donné par un simple polynôme.*

Tout au long de ce mémoire, nous utiliserons la notation $\mathcal{E}(S)$ pour désigner le nombre de points entiers appartenant à un ensemble paramétré S.

Exemple 6. Le nombre de points entiers du polytope (2.18) est donné par le quasi-polynôme d'Ehrhart suivant :

$$\mathcal{E}(P_{\mathbf{p}}) = \begin{cases} -N^2 - NM + 3N + \frac{7}{2}M + \left[2, \frac{3}{2}\right]_M & si \quad N + M \geq 0 \wedge -2N - M \geq -2 \\ \frac{1}{4}M^2 + 2M + \left[4, \frac{15}{4}\right]_M & si \quad M \geq -2 \wedge 2N + M \geq 2 \end{cases}$$

Note 2. *Le quasi-polynôme représentant le nombre de points entiers dans un polytope paramétré est communément appelé* polynôme d'Ehrhart *par la communauté de compilation, quelle que soit sa forme. Les mathématiciens, quant à eux, désignent par* quasi-polynôme d'Ehrhart *le polynôme défini par le théorème fondamental d'Ehrhart, et par* polynôme d'Ehrhart *un quasi-polynôme d'Ehrhart dont la période est égale à 1.*

Dans le chapitre suivant, nous commencerons par présenter un aperçu de quelques méthodes de comptage de points entiers dans des polytopes, et nous décrirons ensuite une nouvelle méthode de calcul de quasi-polynômes d'Ehrhart, basée sur la décomposition unimodulaire de Barvinok.

Chapitre 3

Dénombrement des points entiers de polytopes paramétrés

Le comptage de points entiers dans des polyèdres convexes bornés (polytopes) constitue une étape cruciale dans plusieurs contextes. En plus de l'utilité de ce comptage en compilation, que nous avons décrit dans le chapitre 2, De Loera et al. [46] citent plusieurs utilisations en mathématique, telles que la combinatoire [100, 133], la théorie des représentations [80, 125], les statistiques [47, 60], la théorie des nombres [16, 110] et l'optimisation discrète [66, 126]. Le comptage de points entiers dans des polytopes trouve aussi des applications en économie [90].

Depuis l'apparition des premiers algorithmes de comptage de points entiers dans des polytopes, plusieurs extensions et nouveaux algorithmes ont été proposés, e.g., [52, 115, 14, 39, 26, 113, 46, 148, 17, 9]. Certains algorithmes [155, 120, 119] ne calculent pas le nombre exact de points entiers mais une approximation de ce nombre. Dans ce manuscrit nous ne nous intéressons qu'aux méthodes exactes dont on distingue deux catégories principales : (i) les méthodes non paramétrées ne permettant de compter les points entiers que dans des polytopes de formes fixes, et dont le nombre de points est donné par une simple constante ; (ii) les méthodes paramétrées permettant de compter les points entiers dans des polytopes *paramétrés*, dont la forme dépend des valeurs de paramètres. Le résultat de ces algorithmes est donné par un ou plusieurs quasi-polynômes, aussi connus sous le nom de *quasi-polynômes d'Ehrhart* en référence au mathématicien Eugène Ehrhart qui fut le premier à les découvrir.

Remarque 1. *La plupart des algorithmes de comptage de points entiers que nous décrivons dans ce mémoire prennent en entrée un ensemble de contraintes linéaires. Ces contraintes sont mémorisées dans un fichier binaire. Tout au long de ce mémoire, lorsque nous parlons de complexité, par* taille des contraintes *on considère la taille du fichier correspondant, et par* dimension fixée *une dimension bornée par une constante relativement petite.*

Avant de présenter notre algorithme de comptage de points entiers dans des polytopes paramétrés, nous donnons un aperçu des méthodes les plus connues dans ce contexte, et nous présentons plus en détail l'algorithme de Barvinok [14, 46] qui introduira notre algorithme.

3.1 Aperçu des méthodes de comptage de points entiers dans des polytopes

3.1.1 Méthodes des sommes

En 1992, Tawbi et Feautrier [137] ont proposé une méthode de calcul du nombre de points entiers dans un polytope appelée *méthode des sommes*. Cette méthode, améliorée par Pugh [115], peut être utilisée pour calculer le nombre de points entiers dans des polytopes issus de l'analyse de nids de boucles. La méthode des sommes est basée sur les formules standards de sommes de puissances de nombres entiers [18]. Par exemple :

$$(\sum i : 1 \leq i \leq n : i^2) = (1 \leq n : \frac{n(n+1)(2n+1)}{6}),$$

où $(\sum i : 1 \leq i \leq n : i^2)$ signifie la somme (sur i) de i^2, avec $1 \leq i \leq n$.

Considérons maintenant une formule de sommes plus générale :

$$(\sum i_1, \cdots, i_d :$$
$$l_1(\mathbf{p}) \leq i_1 \leq u_1(\mathbf{p}), \cdots, l_d(i_1, \cdots, i_{d-1}, \mathbf{p}) \leq i_d \leq u_d(i_1, \cdots, i_{d-1}, \mathbf{p}) : f(i_1, \cdots, i_d, \mathbf{p})),$$

où l_i, u_j sont des fonctions affines et $f(i_1, \cdots, i_d, \mathbf{p})$ est une fonction polynomiale des variables et des paramètres $\mathbf{p}$. Pour calculer des sommes de cette forme, on peut procéder à un ensemble de simplifications et de règles de réécriture pour se ramener aux fonctions de sommes de base décrites dans [18]. À chaque étape, la somme est calculée par rapport à une variable en considérant le reste des variables comme des constantes.

En remplaçant le terme $f(i_1, \cdots, i_d, \mathbf{p})$ par 1 dans la formule ci-dessus, nous obtenons la formule de sommes correspondant au nombre de points entiers dans le polytope défini par les bornes inférieures et supérieures sur les variables $i_1, \cdots, i_d$. Des expressions polynomiales apparaissent dans le terme $f(i_1, \cdots, i_d, \mathbf{p})$ au fur et à mesure que les variables sont éliminées.

Exemple 7. Considérons le nids de boucles suivant [136] :

```
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    for (k=j; k<=m; k++)
      S: ...
```

Le nombre de fois que l'instruction `S` est exécutée est donné par le nombre de points entiers dans un polytope $P = \{(i,j,k) \in \mathbb{Q}^3 \mid 1 \leq j \leq i \leq n \wedge j \leq k \leq m\}$ (la contrainte $i \geq 1$ est redondante). La formule de sommes correspondante à ce polytope est

$$(\sum i, j, k : 1 \leq j \leq i \leq n \wedge j \leq k \leq m : 1).$$

En faisant la somme sur k puis sur i, on trouve respectivement les formules de sommes suivantes :

$$(\sum i, j : 1 \leq j \leq i \leq n \wedge j \leq m : m - j + 1).$$

$$(\sum j : 1 \leq j \leq n, m : (n - j + 1)(m - j + 1)).$$

À ce stade, il faut d'abord séparer les deux bornes supérieures sur j, ce qui donne

$$(\sum j : 1 \leq j \leq n \leq m : (n-j+1)(m-j+1)) + \\ (\sum j : 1 \leq j \leq m < n : (n-j+1)(m-j+1)),$$

puis faire la somme sur j pour obtenir le résultat final

$$(1 \leq n \leq m : \frac{mn^2}{2} - \frac{n^3}{6} + \frac{nm}{2} + \frac{n}{6}) + (1 \leq m < n : \frac{m^2n}{2} - \frac{m^3}{6} + \frac{nm}{2} + \frac{m}{6}).$$

Même si cette méthode de sommes semble pratique dans des cas simples, elle reste très difficile à automatiser dans le cas général, d'autant plus qu'elle est exponentielle dans le pire cas. En effet, lorsque les bornes sur les variables sont rationnelles, cette méthode propose de générer un ensemble de réponses proportionnel aux dénominateurs des bornes. Afin de prévenir ce comportement exponentiel, Pugh [115] propose de calculer des approximations plutôt que les valeurs exactes. Ces approximations peuvent être intéressantes lorsque le calcul du nombre *exact* n'est pas nécessaire.

3.1.2 Méthode d'interpolation

La méthode d'interpolation introduite par Clauss et Loechner [39] partage avec notre algorithme (section 3.3) la particularité de manipuler des polytopes paramétrés quelconques (de la forme (2.14)), et d'être entièrement implémentée. C'est la raison pour laquelle nous la présentons un peu plus en détail. Une comparaison des performances des deux méthodes est présentée dans la section 3.3.6.

En 1998, Clauss et Loechner [39] ont utilisé le théorème 4 pour implémenter (dans la `Polylib` [95]) une méthode, dite *d'interpolation*, dédiée au calcul de quasi-polynômes d'Ehrhart représentant le nombre de points entiers dans un polytope paramétré quelconque. Le principe de cette méthode est le suivant :

- Calculer les sommets paramétrés et leurs domaines de validité (algorithme de Loechner-Wilde [98]).
- Calculer les domaines de validité du polytope et la liste des sommets qui sont valides sur chaque domaine, i.e., décomposition en polytopes homothétiques (algorithme de Clauss/Loechner [39]).
- Pour chaque domaine de validité (ou polytope homothétique), calculer par *interpolation* le quasi-polynôme d'Ehrhart correspondant.

L'interpolation du polynôme d'Ehrhart sur chaque domaine de validité se fait en fonction des dénominateurs des sommets du polytope sur le domaine considéré, et d'un certain nombre d'énumérations initiales de polytopes non paramétrés, obtenus par substitution de paramètres.

Considérons un d-polytope à un seul paramètre n dont le dénominateur (*ppcm* des dénominateurs des coordonnées de ses sommets) sur un domaine de validité donné est q. D'après le théorème fondamental d'Ehrhart, la forme générale du nombre de points entiers de ce polytope est :

$$\mathcal{E}(n) = u_d n^d + \cdots + [u_{1,q-1}, \ldots, u_{1,0}]_n n + [u_{0,q-1}, \ldots, u_{0,0}]_n. \tag{3.1}$$

Les valeurs des $u_{i,j}$ où $0 \leq i \leq d$ et $0 \leq j < q-1$ sont inconnues. Pour les calculer, nous devons générer un système de $(d+1) \times q$ équations en substituant n à différentes valeurs pour lesquelles le nombre de solutions (points entiers) est facilement calculable. La solution implémentée dans la `Polylib` consiste à résoudre q systèmes de $d+1$ équations chacun. Chaque système est obtenu de la manière suivante :

On sait que pour une valeur donnée n_0 de n avec $n_0 \equiv k \bmod q$, le pseudo-polynôme ci-dessus devient un simple polynôme (à coefficients constants) : $\mathcal{E}(n_0) = u_d n_0^d + u_{d-1,k} n_0^{d-1} + \cdots + u_{0,k}$. Les coefficients de ce polynôme restent les mêmes en remplaçant n dans l'équation (3.1) par tout entier de la forme $n_0 + iq$, car $(n_0 + iq) \equiv k \bmod q$. D'autre part, ce polynôme comporte $d+1$ constantes inconnues, donc il suffit de connaître la valeur du polynôme pour les $d+1$ valeurs de n de la forme $n_i = (n_0 + iq)$ avec $0 \leq i < d$, pour calculer tous ses coefficients. Cela revient à résoudre le système suivant :

$$\left\{ \begin{array}{rcl} u_d n_0^d + u_{d-1,k} n_0^{d-1} + \cdots + u_{0,k} & = & \mathcal{E}(n_0) \\ u_d n_1^d + u_{d-1,k} n_1^{d-1} + \cdots + u_{0,k} & = & \mathcal{E}(n_1) \\ & \vdots & \\ u_d n_d^d + u_{d-1,k} n_d^{d-1} + \cdots + u_{0,k} & = & \mathcal{E}(n_d) \end{array} \right. \tag{3.2}$$

Chaque constante $\mathcal{E}(n_i)$ correspond au nombre de points entiers du polytope (non paramétré) lorsque $n = n_i$. Ce nombre doit être préalablement calculé.

Le principe de cette interpolation reste le même pour le cas général de plusieurs paramètres. Le seul élément à prendre en compte est que maintenant, les quantités $\mathcal{E}(n_i)$ peuvent être non pas des simples constantes mais des quasi-polynômes en fonction du reste des paramètres. Ces derniers sont calculés en faisant des appels récursifs à la même procédure expliquée ci-dessus, jusqu'à ce que tous les paramètres soient substitués.

La méthode de calcul de quasi-polynômes d'Ehrhart par interpolation a deux inconvénients : l'énumération initiale de polytopes non paramétrés et la taille exponentielle de la solution (dans le pire des cas).

Énumération initiale de polytopes non paramétrés

Nous avons vu dans le paragraphe ci-dessus qu'en se basant sur la connaissance de la forme générale de la solution (théorème 4), Clauss et Loechner [39] interpolent un quasi-polynôme pour chaque domaine de validité, en calculant le nombre de points dans un ensemble de polytopes *non paramétrés*. Ces polytopes sont obtenus par instanciation des paramètres sur le domaine de validité concerné. Pour interpoler un quasi-polynôme de dimension d et de périodes q_i $(1 \leq i \leq n)$, il est nécessaire de faire $\prod_{i=1}^{n}(d+1)q_i$ énumérations initiales, où n est le nombre de paramètres. Cela veut dire qu'il faut trouver $\prod_{i=1}^{n}(d+1)q_i$ combinaisons différentes de valeurs des paramètres. Ces valeurs sont recherchées dans un hypercube situé à l'intérieur du domaine de validité. Or il est parfois impossible de trouver un hypercube de la bonne taille qui est entièrement inclus dans le domaine de validité. Dans ce cas, la méthode échoue.

Exemple 8. Considérons le nid de boucles présenté par la figure 3.1. Le nombre de fois que l'instruction `t += A[i+j]` est exécutée par ce nid est donné par le nombre de

```
void s(int N, int M, int *A)
{
  int i,j;
  for(i=max(0,N-M); i<=N-M+3; i++)
    for(j=0; j<=N-2*i; j++)
      t += A[i+j];
}
```

FIG. 3.1 – Nid de boucles

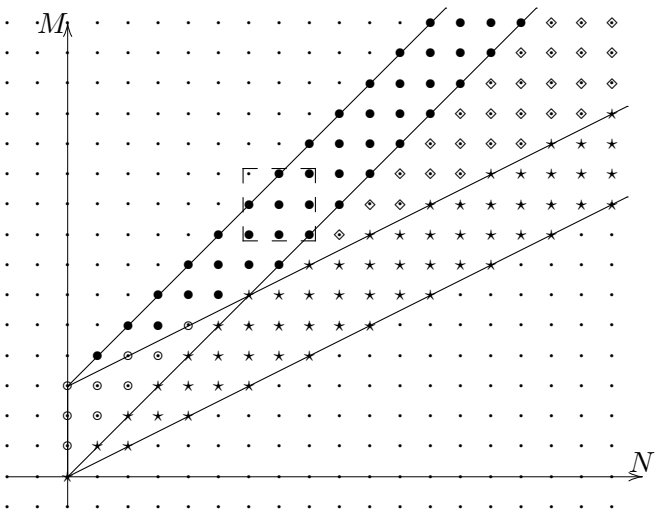


FIG. 3.2 – La représentation géométrique de domaines de validité du polytope (3.3). Le domaine marqué par • correspond au cas dégénéré.

points entiers dans le polytope

$$P_{\binom{N}{M}} = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Q}^2 \mid \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ -2 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} \geq \begin{pmatrix} 0 & 0 \\ 1 & -1 \\ -1 & 1 \\ 0 & 0 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} N \\ M \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -3 \\ 0 \\ 0 \end{pmatrix} \right\}. \quad (3.3)$$

En appliquant les algorithmes de calcul de sommet et de décomposition en domaines de validité décrits respectivement dans les sections 2.5.1 et 2.5.2, on trouve quatre domaines de validité auxquels sont associés des sous-ensembles de sommets du polytope. Les différents domaines de validité sont illustrés par la figure 3.2. Considérons le domaine marqué par $\bullet$ ($N \leq M \leq N+3 \wedge N \leq 2M-7$). Sur ce domaine, tous les sommets sont entiers, i.e., les périodes (en N et M) sont égales à 1. Le nombre d'énumérations initiales nécessaires pour l'interpolation du polynôme est donc $\prod_{i=1}^{2}(2+1) \times 1 = 9$. On a donc besoin de neuf combinaisons différentes de valeurs des paramètres. L'algorithme de Clauss/Loechner recherche ces valeurs dans un rectangle de dimension 3×3. Or, comme on peut le voir sur la figure 3.2, aucun rectangle de cette taille n'est entièrement inclus dans le domaine de validité en question. C'est ce qui est connu sous le nom de *domaine dégénéré*.

À noter que le problème des domaines dégénérés peut être résolu, par exemple, en rajoutant un paramètre supplémentaire [112]. Mais le problème de la complexité due au nombre d'énumérations initiales reste intrinsèque à la méthode.

Taille de la solution

Puisque les périodes q_i d'un quasi-polynôme sont bornées seulement par les *valeurs* des coefficients des contraintes définissant un polytope donné, elle peuvent être exponentielles en la taille de ces contraintes. Ainsi, le *temps de calcul* par interpolation d'un quasi-polynôme peut être *exponentiel* dans le pire des cas, et ce même pour une dimension fixée. Par ailleurs, puisque la méthode d'interpolation représente les nombres périodiques par des tableaux multidimensionnels dont la taille dépend des périodes, la *taille du résultat* (quasi-polynômes) peut aussi être exponentielle dans le pire des cas.

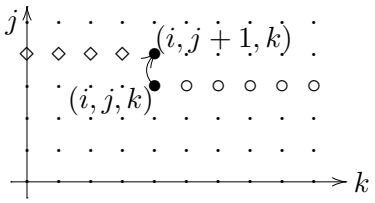
Exemple 9. Considérons le programme de la figure 3.3 (multiplication de matrices), et supposons que nous souhaitons compter le nombre de pages distinctes dont l'adresse est stockée dans le TLB (*Translation Lookaside Buffer*) qui sont accédées, entre deux accès consécutifs à la même page du TLB. Ce nombre, dit distance de réutilisation [24], peut servir pour détecter le nombre de défauts de TLB.

Pour des raisons de simplicité, nous supposons que `A[i][k]` et `B[k][j]` accèdent à des pages de TLB différentes, et nous nous concentrons sur `A[i][k]`. Nous supposons aussi que `A` est une matrice 200×200 stockée en mémoire par colonnes (*column major order*) à partir de l'adresse zéro. La taille d'un élément de A est supposée de 4 octets. Ainsi, un élément `A[i][k]` se situe à l'adresse $4 \times (200k + i)$.

Les itérations (i, j, k) et $(i, j+1, k)$ accèdent au même élément `A[i][k]`. La figure 3.3b montre les itérations qui sont exécutées entre ces deux itérations : les itérations $(i, j, k+1 \ldots 199)$ ($\circ$ sur la figure) et les itérations $(i, j+1, 0 \ldots k-1)$ ($\diamond$ sur la

```
do i = 0, 199
  do j = 0, 199
    s = 0
    do k = 0, 199
      s = s + A[i][k] * B[k][j]
    enddo
    C[i][k] = s
  enddo
enddo
```

(a) Code source



(b) accès intermédiaires

FIG. 3.3 – Multiplication de matrices

figure). L'ensemble de pages de TLB accédées par les itérations $\circ$ peut être décrit par :

$$S_1 = \left\{ t \mid \exists k' : t = \left\lfloor \frac{800k' + 4i}{L} \right\rfloor \wedge 0 \leq i, j, k \leq 199 \wedge \ k + 1 \leq k' \leq 199 \right\},$$

où i, j et k sont des paramètres. En supposant que la taille d'une page est $L = 4096$, ceci peut être écrit sous forme d'un ensemble de contraintes linéaires :

$$S_1 = \{\, t \mid \exists k' : 1024t \leq 200k' + i \leq 1024t + 1023 \\ \wedge\, 0 \leq i, j, k \leq 199 \wedge k + 1 \leq k' \leq 199 \,\}.$$

Ceci peut être simplifié (e.g., en utilisant la librairie Omega [76]) en :

$$S_1 = \{\, t \mid 0 \leq i \wedge 1024t - 39800 \leq i \leq 199 \wedge 0 \leq k \leq 198 \\ \wedge\, 0 \leq j \leq 199 \wedge i + 200k \leq 823 + 1024t \,\}.$$

En procédant d'une façon similaire, on obtient une expression S_2 pour les itérations $\diamond$. Le nombre total de pages de TLB est $\mathcal{E}(S_1 \cup S_2) = \mathcal{E}S_1 + \mathcal{E}(S_2 \setminus S_1)$. L'ensemble $S1$ correspond à un polytope dont les sommets sont calculés par la `Polylib` sous la forme :

$$\frac{i}{1024} + 25\frac{k}{128} - \frac{823}{1024} \quad \text{et} \quad \frac{i}{1024} + \frac{4975}{128}.$$

Puisque la dimension de ce polytope est $d = 1$ et ses périodes sont $q_i = 1024$, $q_j = 1$ et $q_k = 128$, la méthode d'interpolation procède à $2^3 \cdot 1024 \cdot 128$ énumérations initiales. En supposant qu'une valeur (coefficient du quasi-polynôme) est codée sur deux octets, on trouve que chaque nombre périodique nécessite plus de 256KiB pour être stocké en mémoire. En utilisant notre méthode (section 3.3), on trouve, en un temps polynomial, un résultat équivalent mais beaucoup plus petit :

$$\left\lfloor \frac{i + 888}{1024} \right\rfloor - \left\lfloor \frac{i + 200k + 199}{1024} \right\rfloor + 39.$$

Certaines méthodes d'optimisation de programmes [24, 61, 97, 138] incorporent les quasi-polynômes résultants dans les programmes qu'elles optimisent. Lorsque ces quasi-polynômes sont de grandes tailles, la taille des programmes optimisés augmente, ce qui

rend les optimisations moins intéressantes, en particulier dans le contexte des systèmes embarqués. Nous montrerons un peu plus loin qu'en utilisant notre méthode, la taille des quasi-polynômes résultants n'augmente pas exponentiellement, comme c'est le cas pour la méthode d'interpolation.

3.1.3 Méthode des fractions partielles

La méthode des fractions partielles s'intéresse à des fonctions de comptage particulières, dites *les fonctions de partition vectorielle*. Dans ce qui suit, nous expliquerons ce que sont ces fonctions et nous décrirons brièvement la méthode des fractions partielles.

Les fonctions de partition vectorielle (*vector partition functions*) [134] sont des généralisations aux vecteurs, des *fonctions de partition* correspondant au nombre de représentations différentes d'un entier positif (naturel) par une somme d'entiers positifs.

Soit $A = (\mathbf{a}_1, \cdots, \mathbf{a}_n)$ une $(d \times n)$-matrice, de rang d, de nombres naturels ($a_{ij} \in \mathbb{N}$). La *fonction de partition vectorielle* correspondante $\phi_A : \mathbb{N}^d \to \mathbb{N}$ est définie comme suit : pour tout $\mathbf{b} = (b_1, \cdots, b_d)$, $\phi_A(\mathbf{b})$ est le nombre de vecteurs naturels $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_n) \in \mathbb{N}^n$, tel que $A.\lambda = \lambda_1\mathbf{a}_1 + \cdots + \lambda_n\mathbf{a}_n = \mathbf{b}$. D'une manière équivalente, ϕ_A est définie par une série entière formelle

$$\prod_{i=1}^{n} \frac{1}{(1 - \mathbf{t}^{\mathbf{a}_i})} = \sum_{\mathbf{b} \in \mathbb{N}^d} \phi_A(\mathbf{b}).\mathbf{t}^{\mathbf{b}}, \tag{3.4}$$

avec $\mathbf{t}^{\mathbf{a}_i} = t_1^{a_{1i}} t_2^{a_{2i}} \cdots t_d^{a_{di}}$ et $\mathbf{t}^{\mathbf{b}} = t_1^{b_1} t_2^{b_2} \cdots t_d^{b_d}$, où $\mathbf{a}_i$ est la $i^{ème}$ colonne de la matrice A.

Théorème 5 (Sturmfels [134])**.** *La fonction de partition vectorielle $\phi_A(\mathbf{b})$ est donnée par un ensemble de* quasi-polynômes*, en $\mathbf{b}$, de degré $d - rang(A)$. Chaque quasi-polynôme est défini sur une région* (chamber) *de $\mathbb{R}^d$ correspondant à un polyèdre de dimension d.*

Note 3. *Les polyèdres (toujours des cônes) sur lesquels sont définis les quasi-polynômes des fonctions de partition vectorielle sont des cas particuliers des domaines de validité du théorème 4. En effet, toute fonction de partition vectorielle $\phi_A(\mathbf{b})$ est équivalente au nombre de points entiers dans un polytope paramétré particulier P, i.e., $\phi_A(\mathbf{b}) = \#(P)$, où les variables et les paramètres sont tous positifs.*

$$P = \{\boldsymbol{\lambda} \in \mathbb{Q}^n \mid A\boldsymbol{\lambda} = I_1\mathbf{b} + \mathbf{0} \ \wedge \ I_2\boldsymbol{\lambda} \geq \mathbf{0}\},$$

où I_1 et I_2 sont respectivement les matrices identités $(d \times d)$ et $(n \times n)$, et $\mathbf{0}$ est le vecteur nul de dimension d (resp. n).

En 2004, Matthias Beck [17] a proposé une méthode de calcul de la fonction de partition vectorielle $\phi_A(\mathbf{b})$, basée sur le lemme d'Euler suivant :

Lemme 1 (Euler)**.** *La fonction de partition vectorielle $\phi_A(\mathbf{b})$ est égale au coefficient de $\mathbf{z}^{\mathbf{b}} := z_1^{b_1} z_2^{b_2} \cdots z_n^{b_n}$ de la fonction*

$$f(\mathbf{z}) = \frac{1}{(1 - \mathbf{z}^{\mathbf{a}_1}) \cdots (1 - \mathbf{z}^{\mathbf{a}_d})}$$

développée en séries entières centrées à $\mathbf{z} = \mathbf{0}$, où $\mathbf{a}_1, \cdots, \mathbf{a}_d$ sont les vecteurs colonnes de la matrice A.

Le coefficient de $\mathbf{z}^{\mathbf{b}}$ de la fonction $f(\mathbf{z})$ est égal au terme constant (coefficient de $\mathbf{z}^{\mathbf{0}}$) de la fonction $\frac{f(\mathbf{z})}{\mathbf{z}^{\mathbf{b}}}$. Ceci nous mène à écrire

$$\phi_A(\mathbf{b}) = \text{ const } \frac{1}{(1-\mathbf{z}^{\mathbf{a}_1})\cdots(1-\mathbf{z}^{\mathbf{a}_d})\mathbf{z}^{\mathbf{b}}}.$$

Pour calculer cette constante, Beck [17] procède au développement de la fonction ci-dessus en fractions partielles. Ce qui permet d'éliminer, à chaque étape, une variable z_i. Le terme constant est calculé une fois que la fonction n'est définie qu'en fonction d'une seule variable z_n. Nous nous contentons ici de donner le résultat d'un exemple de calcul d'une fonction de partition vectorielle. Le calcul de cet exemple est détaillé dans [17].

Exemple 10. La fonction de partition vectorielle $\phi_A(\mathbf{b})$, avec $A = \begin{pmatrix} 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$ et $\mathbf{b} = (a, b)$ ou autrement dit, le nombre de solutions entières du système

$$\begin{cases} x_1 + 2x_2 + x_3 &= a \\ x_1 + x_2 + x_4 &= b \end{cases}, \; x_1, x_2, x_3, x_4 \geq 0$$

est donnée par la méthode des fractions partielles sous la forme :

$$\phi_A(a,b) = \begin{cases} \frac{a^2}{4} + a + \frac{7+(-1)^a}{8} & \text{si } a \leq b, \\ ab - \frac{a^2}{4} - \frac{b^2}{2} + \frac{a+b}{2} + \frac{7+(-1)^a}{8} & \text{si } \frac{a}{2} - 1 \leq b \leq a+1, \\ \frac{b^2}{2} + \frac{3b}{2} + 1 & \text{si } b \leq \frac{a}{2}. \end{cases}$$

Noter que le terme $\frac{7+(-1)^a}{8}$ est équivalent à un nombre périodique $\left[1, \frac{3}{4}\right]_a$. Cela veut dire que la fonction $\phi_A(a,b)$ est donnée par des *quasi*-polynômes sur les deux premiers domaines de validité, et par un simple polynôme sur le dernier.

Plus récemment Baldoni et al. [9] ont proposé une implémentation des fractions partielles. Ils ont montré en particulier que leur implémentation est plus efficace que toutes les implémentations existantes pour une classe particulière de polytopes (*systèmes de racines*). Ils ont également mentionné que leurs programmes ont été conçus spécialement pour ce type de polytopes. Dans le cas général auquel nous nous sommes intéressés, les auteurs se sont contentés de monter que leur méthode est théoriquement générale, et pourrait être appliquée à tout polytope paramétré. Mais aucune analyse de complexité ni expériences n'ont été fournies. À noter que les techniques standard de calcul des fractions partielles [72] sont exponentielles.

Les méthodes de comptage vues jusque-là permettent de manipuler des polytopes *paramétrés*, où les résultats sont donnés par des (quasi-)polynômes en fonction de constantes symboliques (paramètres). Lorsqu'un polytope, ou d'une manière générale une formule de Presburger n'est pas paramétrée, le problème de comptage de points entiers est beaucoup moins compliqué, puisque le résultat est donné par une simple constante. Parmi les méthodes de comptage non paramétrées, il y a celles qui sont basées sur les automates à états finis [113, 26]. Dans ce qui suit, nous décrivons brièvement la méthode de Parker/Chatterjee [113] pour donner une idée de ce type de comptage.

3.1.4 Méthode de comptage par automates à états finis déterministes

En 2004, Parker et Chatterjee [113] ont proposé un algorithme de comptage basé sur le travail de Bartzis et Bultan [10]. L'idée est de représenter les entiers vérifiant chaque contrainte linéaire ($\sum_i a_i x_i + b$ $(=, <, >, \leq, \geq)$ 0) d'une formule de Presburger par un automate à états finis déterministe. Les différents automates sont ensuite combinés en utilisant les opérations sur les automates (intersection, union, complément et projection) offertes par la librairie `MONA` [82]. Le résultat est un automate dont le langage reconnu correspond aux solutions (codées en binaire) de la formule de Presburger. Chaque chemin accepté dans l'automate correspond à une solution. Par conséquent, le problème de comptage de solutions de la formule de Presburger se réduit au calcul du nombre de chemins (de même longueur[1]) acceptés par un automate à états finis déterministe $M = (S, \Sigma, \delta, q_0, F)$, où S est l'ensemble des états de l'automate, Σ est un ensemble fini de symboles dit *alphabet* (ici, des codes binaires de nombres entiers), δ est la fonction de transition $\delta : S \times \Sigma \to S$, q_0 est l'état initial, et F est l'ensemble des états finaux.

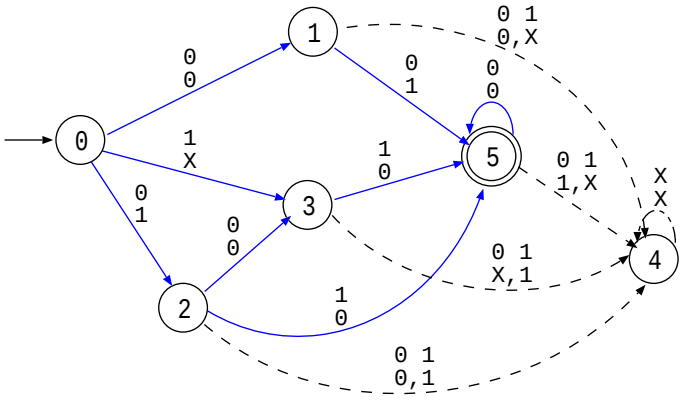


FIG. 3.4 – Automate à états finis représentant les solutions entières du système (3.5)

Exemple 11. La figure 3.4 montre un automate à états finis déterministe représentant les points entiers du polytope

$$P = \{(x, y) \in \mathbb{Q}^2 \mid 2x + 3y - 6 \geq 0 \wedge -x + y + 3 \geq 0, -x - 4y + 8 \geq 0\} \qquad (3.5)$$

Sur cette figure le caractère `X` indique un bit quelconque (0 ou 1), et les lignes en continu représentent les chemins qui donnent les mots (vecteurs entiers) acceptés par l'automate. Ces mots sont codés sur 3 bits (pour chaque variable) puisque la longueur du plus long chemin de l'état initial (0) jusqu'à l'état final (5) est 3. Le nombre de solutions est donné par le nombre de chemins de longueur 3 reliant l'état initial à l'état final. Dans notre exemple, ce nombre est égal à 5. Noter que l'arc reliant les états 0

[1] toutes les solutions sont représentées par le même nombre de bits

et 3 est compté deux fois à cause de la présence du caractère `X` dans sa pondération. Les mots acceptés par l'automate peuvent être explicitement obtenus, en concaténant, de droite à gauche, les bits reconnus. Ces mots (solutions du système (3.5)) sont : $\binom{000}{010} = \binom{0}{2}, \binom{011}{000} = \binom{3}{0}, \binom{011}{001} = \binom{3}{1}, \binom{010}{001} = \binom{2}{1}, \binom{100}{001} = \binom{4}{1}$.

Outre le fait qu'elle ne peut être généralisée au cas paramétré, la méthode de comptage par automates est de complexité proportionnelle au nombre de solutions de la formule de Presburger. Plus ce nombre est grand, plus les automates sont volumineux, ce qui influe considérablement sur le temps de calcul. Sur certains exemples (polytopes non-paramétrés) dont le nombre de points entiers est grand, notre algorithme (section 3.3) est jusqu'à 100 fois plus rapide. Ceci est dû au fait que notre méthode dépend des sommets du polytope et non pas du nombre de ses points entiers. La méthode de comptage par automates reste tout de même assez efficace lorsque le nombre de solutions est relativement petit.

3.2 Algorithme de Barvinok

Dans cette section, nous présentons l'algorithme de Barvinok pour le comptage de points entiers dans des polytopes non paramétrés. Cet algorithme constitue une introduction à notre méthode (section 3.3) qui est sa généralisation au cas paramétré.

En 1994, A. Barvinok [14] a proposé un algorithme qui calcule le nombre de points entiers dans un polytope en un temps polynomial (pour une dimension fixée). Cet algorithme a été récemment implémenté dans `LattE` [46, 154]. Il consiste en deux étapes principales. D'abord la *fonction génératrice* du polytope est calculée. Ensuite, cette fonction est évaluée pour obtenir le nombre de points du polytope. Dans le reste de ce chapitre, nous considérons sans perte de généralité que le polytope est de dimension pleine, c'est-à-dire qu'il ne comporte aucune égalité dans le système de contraintes linéaires le définissant. La suppression des égalités d'un polytope, i.e., sa transformation en un polytope de dimension pleine, sans affecter le nombre de points entiers qu'il comporte, est expliquée dans la section 3.3.5.

3.2.1 Fonction génératrice d'un polytope

L'idée de base de l'algorithme de Barvinok est de considérer la *fonction génératrice* des points entiers appartenant à un polytope P. Cette fonction est une série entière formelle avec un terme unique pour chaque point entier du polytope P, i.e.,

$$f(P;\mathbf{x}) = \sum_{\mathbf{m}\in\mathbb{Z}^d} [P](\mathbf{m})\mathbf{x}^{\mathbf{m}},$$

avec $\mathbf{x}^{\mathbf{m}} = x_1^{m_1} x_2^{m_2} \cdots x_d^{m_d}$, et où $[P]$ est la fonction indicatrice de P, tel que $[P](\mathbf{m}) = 1$ si $\mathbf{m} \in P$ et 0 sinon. La fonction génératrice de P peut aussi être écrite sous la forme

$$f(P;\mathbf{x}) = \sum_{\mathbf{m}\in P\cap\mathbb{Z}^d} \mathbf{x}^{\mathbf{m}},$$

En évaluant cette fonction à $\mathbf{x} = \mathbf{1}$, on obtient le nombre de termes de la série qui correspond au nombre de points entiers de P. La fonction génératrice n'est évidemment

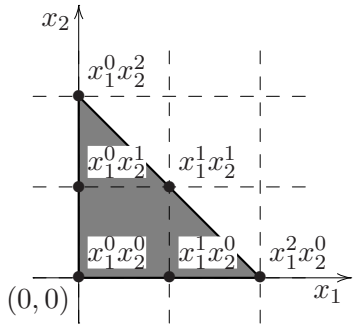


FIG. 3.5 – Exemple de Barvinok. Pour chaque point entier (i, j) dans le polytope P, il y a un terme $x_1^i x_2^j$ correspondant dans la fonction génératrice $f(P; \mathbf{x})$.

pas construite en parcourant tous les points entiers du polytope P, mais plutôt en calculant une somme signée d'un petit nombre de fonctions rationnelles (courtes) dérivées de la description de P.

Exemple 12. Considérons le polytope montré par la figure 3.5 :

$$P = \{\, \mathbf{x} \mid x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_1 + x_2 \leq 2 \,\}\,.$$

Les points entiers de P sont $(0,0)$, $(1,0)$, $(2,0)$, $(0,1)$, $(1,1)$, et $(0,2)$. La fonction génératrice de P correspondant à ces points est :

$$f(P; \mathbf{x}) = 1 + x_1 + x_1^2 + x_2 + x_1x_2 + x_2^2. \tag{3.6}$$

Cette fonction est donnée par l'algorithme de Barvinok sous forme d'une somme de fonctions rationnelles qui ne nécessitent pas de parcourir tous les points entiers du polytope :

$$f(P; \mathbf{x}) = \frac{x_1^2}{(1 - x_1^{-1})(1 - x_1^{-1}x_2)} + \frac{x_2^2}{(1 - x_2^{-1})(1 - x_1x_2^{-1})} + \frac{1}{(1 - x_1)(1 - x_2)}. \tag{3.7}$$

En substituant $(x_1, x_2) = (1, 1)$ dans la fonction (3.6) on trouve 6 (le nombre de points entiers de P). La substitution de $(x_1, x_2) = (1, 1)$ dans la fonction (3.7) devrait aussi donner 6, mais pour l'instant, on souligne juste que $(x_1, x_2) = (1, 1)$ est un pôle (dénominateur nul) des 3 fractions de la fonction génératrice. La section 3.2.2 explique comment est traité ce problème, en calculant la limite lorsque (x_1, x_2) tend vers $(1, 1)$.

Avant de décrire comment les fonctions génératrices sont construites par l'algorithme de Barvinok, nous introduisons quelques définitions et théorèmes.

Définition 21. *Un* **cône supportant** *d'un d-polyèdre P en un sommet $\mathbf{v}$ est la région de $\mathbb{Q}^d$ bornée par les contraintes de P qui sont saturées par $\mathbf{v}$. Autrement dit, c'est la région bornée par les contraintes $\langle \mathbf{a}, \mathbf{x} \rangle \geq \mathbf{b}$ avec $\langle \mathbf{a}, \mathbf{v} \rangle = \mathbf{b}$, où $\langle ., . \rangle$ est le produit scalaire de vecteurs.*

Théorème 6 (Brion [30])**.** *Soient P un polyèdre convexe entier*[2] *ne comportant aucune ligne, et $\mathcal{V}(P)$ l'ensemble de ses sommets. La fonction génératrice (caractéristique) de P*

[2]Un polyèdre est dit *entier* si toutes les coordonnées de ses sommets sont des entiers.

est donnée par la somme sur $\mathcal{V}(P)$ des fonctions génératrices de ses cônes supportants.

$$f(P;\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{V}(P)} f(K(P,\mathbf{v});\mathbf{x}). \tag{3.8}$$

En se basant sur ce théorème, Barvinok [12, 11] a démontré que le nombre de points entiers d'un polytope rationnel quelconque peut être calculé en un temps polynomial (pour une dimension fixé). Nous reviendrons sur ce point après présentation de quelques notions fondamentales.

Le cône supportant $K(P,\mathbf{v})$ du polyèdre P au sommet $\mathbf{v}$ n'est pas lui même un cône, mais la translation au sommet $\mathbf{v}$ d'un cône polyédrique $K_\mathbf{v}$ (ayant l'origine O comme sommet)[3], i.e., $K(P,\mathbf{v}) = K_\mathbf{v} + \mathbf{v}$. Lorsque tous les sommets du polyèdre sont entiers, la fonction génératrice (3.8) peut être écrite sous la forme : [12]

$$f(P;\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{V}(P)} f(K_\mathbf{v}+\mathbf{v};\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{V}(P)} \mathbf{x}^\mathbf{v} f(K_\mathbf{v};\mathbf{x}). \tag{3.9}$$

Nous nous focalisons maintenant sur le calcul de la fonction génératrice $f(K;\mathbf{x})$ d'un cône polyédrique K.

Définition 22. *Un* **cône polyédrique simplicial** *de dimension d est un cône polyédrique qui peut être engendré par d demi-droites, ou autrement dit, c'est un cône polyédrique ayant un nombre d'arêtes égal à sa dimension d.*

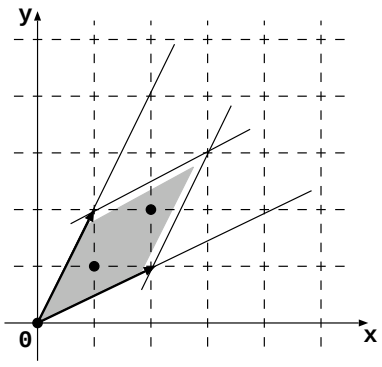


FIG. 3.6 – La région en gris est le parallélépipède fondamental du cône polyédrique $\text{pos}\{(2,1),(1,2)\}$.

Définition 23. *Soient K un cône polyédrique de dimension d ayant $\mathbf{u}_1,\mathbf{u}_2,\cdots,\mathbf{u}_d \in \mathbb{Z}^d$ comme vecteurs générateurs, i.e., $K = \{\sum_{i=1}^d \lambda_i\mathbf{u}_i \mid \lambda_i \geq 0\}$. Le* **parallélépipède fondamental** Π *du cône K est la région de K délimitée par les vecteurs générateurs $\{\mathbf{u}_1,\mathbf{u}_2,\cdots,\mathbf{u}_d\}$: $\Pi = \{\sum_{i=1}^d \lambda_i\mathbf{u}_i,\ 0 \leq \lambda_i < 1\}$ (voir la figure 3.6).*

Note 4. *Le nombre de points entiers du parallélépipède fondamental d'un cône K de dimension pleine est égal à la valeur absolue du déterminant de la matrice dont les colonnes sont les vecteurs générateurs de K.*

[3] Un cône polyédrique est par définition un polyèdre ayant l'origine comme sommet unique.

Il a été démontré [30, 133] qu'il est facile de calculer la fonction génératrice $f(K;\mathbf{x})$ d'un cône polyédrique K lorsqu'il est *simplicial*. En effet, tout point entier $\mathbf{m}$ du cône K ($\mathbf{m} \in K \cap \mathbb{Z}^d$) peut être représenté d'une façon unique sous la forme :

$$\mathbf{m} = \mathbf{n} + \sum_{i=1}^{d} a_i \mathbf{u}_i,$$

où, a_i ($1 \leq i \leq d$) sont des entiers non négatifs et $\mathbf{n}$ est un point entier appartenant au parallélépipède fondamental Π de K ($\mathbf{n} \in \Pi \cap \mathbb{Z}^d$). Autrement dit, il suffit de connaître les points entiers du parallélépipède fondamental de K pour pouvoir générer tous les autres points de K. La fonction génératrice du cône K peut donc être écrite en fonction des points entiers de son parallélépipède fondamental et de ses vecteurs générateurs comme suit :

$$f(K;\mathbf{x}) = \sum_{\mathbf{m} \in K \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{m}} = \left(\sum_{\mathbf{n} \in \Pi \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{n}} \right) \prod_{i=1}^{d} \frac{1}{1 - \mathbf{x}^{\mathbf{u}_i}}. \tag{3.10}$$

Exemple 13. Considérons un cône polyédrique K dont les vecteurs générateurs sont : $\mathbf{u}_1 = (1,2)$ et $\mathbf{u}_2 = (2,1)$ (cône de la figure 3.6). Le parallélépipède fondamental de ce cône comporte trois points entiers $\mathbf{m}_1 = (0,0), \mathbf{m}_2 = (1,1)$ et $\mathbf{m}_3 = (2,2)$. D'où, la fonction génératrice est :

$$f(K;\mathbf{x}) = (\mathbf{x}^{\mathbf{m}_1} + \mathbf{x}^{\mathbf{m}_2} + \mathbf{x}^{\mathbf{m}_3}) . \frac{1}{1 - \mathbf{x}^{\mathbf{u}_1}} . \frac{1}{1 - \mathbf{x}^{\mathbf{u}_2}} = \frac{1 + xy + x^2y^2}{(1 - xy^2)(1 - x^2y)}.$$

Puisqu'il est possible de décomposer tout cône polyédrique en un ensemble de cônes simpliciaux à travers une procédure de triangulation [5, 87], il devient possible de calculer la fonction génératrice de tout polyèdre entier (ne comportant aucune droite) en appliquant les équations (3.9) et (3.10). Cette méthode de calcul de fonctions génératrices est assez efficace dans plusieurs cas. Mais elle reste exponentielle dans le pire des cas. En effet, le nombre de points entiers du parallélépipède fondamental d'un cône peut être grand (puisqu'il dépend des coordonnées des vecteurs générateurs du cône), ce qui augmente la complexité du calcul de la fonction génératrice correspondante.

Définition 24. *Un* **cône simplicial unimodulaire** *de dimension d est un cône polyédrique dont les vecteurs générateurs* $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d \in \mathbb{Z}^d$ *forment une base de* $\mathbb{Z}^d$. *Autrement dit, la matrice formée par ses vecteurs générateurs est unimodulaire*[4]*, ou d'une manière équivalente, son parallélépipède fondamental comporte un point entier unique (l'origine O).*

Afin de réduire la complexité de calcul des fonctions génératrices, A. Barvinok [14, 12, 11] propose de décomposer chaque cône simplicial en un ensemble signé de cônes simpliciaux unimodulaires (ne comportant que l'origine O comme seul point entier dans leurs parallélépipèdes fondamentaux). Dans ce cas, il est clair que la fonction génératrice (3.10) devient plus courte. En effet, la somme $\left(\sum_{\mathbf{n} \in \Pi \cap \mathbb{Z}^d} \mathbf{x}^{\mathbf{n}} \right) = \mathbf{x}^{\mathbf{0}} = 1$. D'où,

$$f(K;\mathbf{x}) = \prod_{i=1}^{d} \frac{1}{(1 - \mathbf{x}^{\mathbf{u}_i})}. \tag{3.11}$$

[4]Une matrice carrée est dite unimodulaire, si son déterminant est égal à ± 1.

À ce stade, on peut déjà donner la fonction génératrice pour le cas le plus simple d'un polytope dont tous les sommets sont entiers et dont tous les cônes sont unimodulaires.

$$f(P;\mathbf{x}) = \sum_{\mathbf{v}\in\mathcal{V}} \frac{\mathbf{x}^{\mathbf{v}}}{\prod_{i=1}^{d}(1-\mathbf{x}^{\mathbf{u}_i})}.$$

Par exemple, le polytope de l'exemple 12 correspond à ce cas simple. En effet, les trois sommets $(0,0)$, $(2,0)$ et $(0,2)$ sont entiers, et les trois cônes $\text{pos}\{(1,0),(0,1)\}$, $\text{pos}\{(-1,0),(-1,1)\}$ et $\text{pos}\{(0,-1),(1,-1)\}$ sont unimodulaires. D'où la fonction génératrice est :

$$f(P;\mathbf{x}) = \frac{1}{(1-x_1)(1-x_2)} + \frac{x_1^2}{(1-x_1^{-1})(1-x_1^{-1}x_2)} + \frac{x_2^2}{(1-x_2^{-1})(1-x_1x_2^{-1})}.$$

Malheureusement, les cônes peuvent être non unimodulaires et les sommets peuvent être rationnels. Dans ce qui suit, nous présentons les idées de Barvinok traitant ce cas général.

Théorème 7 (Barvinok et Pommersheim [12])**.** *Pour une dimension fixée d, il existe un algorithme qui, étant donné un cône polyédrique $K \subset \mathbb{R}^d$, calcule en un* temps polynomial *des cônes unimodulaires K_i, $i \in I = \{1,2,\cdots,l\}$, et des nombres $\epsilon_i \in \{-1,1\}$, tels que*

$$[K] = \sum_{i\in I} \epsilon_i [K_i], \tag{3.12}$$

où $[K]$ est la fonction indicatrice de K, telle que $[K](\mathbf{m})$ égale 1 si $\mathbf{m} \in K$ et 0 sinon.

La décomposition simpliciale unimodulaire d'un cône polyédrique

Dans ce qui suit, nous suivons la description de De Loera et al. [46] pour expliquer comment sont obtenus les cônes simpliciaux unimodulaires à partir d'un cône polyédrique quelconque.

D'abord, la décomposition en cônes simpliciaux est calculée à travers la triangulation [5, 87]. La triangulation implémentée dans les librairies `LattE` [45] et `barvinok` [143] est celle de Delaunay. L'idée de cette triangulation est la suivante :

Étant donné un cône polyédrique K de dimension d avec les vecteurs générateurs $\mathbf{u}_1,\mathbf{u}_2,\cdots,\mathbf{u}_d$, un nouveau cône K' de dimension $d+1$ est calculé en ajoutant une nouvelle coordonnée w_i à chaque vecteur générateur $\mathbf{u}_i$ de K. La coordonnée w_i est égale à la somme des carrés des coordonnées du vecteur $\mathbf{u}_i$, i.e., $w_i = \sum_{j=1}^{d} u_{ij}^2$. Les vecteurs générateurs du nouveau cône K' sont donc : $(\mathbf{u}_1,w_1),(\mathbf{u}_2,w_2),\cdots,(\mathbf{u}_d,w_d)$. Ensuite, on calcule l'enveloppe convexe inférieure de K' et on la projette sur l'espace original (de dimension d) pour obtenir la triangulation de K. L'enveloppe convexe inférieure (par rapport à $\mathbf{w}$) de K' est donnée par ses faces inférieures, où par *face inférieure* on désigne un système $\{\mathbf{x} \in Q^{d+1} \mid A\mathbf{x} = \mathbf{0}\}$, tel que : (i) $\{\mathbf{x} \in Q^{d+1} \mid A\mathbf{x} \geq \mathbf{0}\}$ est un sous-ensemble des inégalités définissant le cône K'. (ii) tous les éléments de la $(d+1)^{ème}$ colonne de A sont positifs. Plus de détails sur cette triangulation sont donnés dans [63, 142].

Une fois que la décomposition simpliciale est calculée, chaque cône (simplicial) résultant est décomposé en un ensemble signé de cônes unimodulaires. L'algorithme de

Barvinok [14, 12, 46] permet de calculer la décomposition unimodulaire en un temps polynomial comme le cite le théorème 7. Le principe de cet algorithme est le suivant :

Soit K un cône simplicial de dimension d avec les vecteurs générateurs $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_d$, M la matrice $(d \times d)$ dont les colonnes sont les vecteurs générateurs de K, et $ind(K) = |det(M)|$ la fonction indice de K qui représente le volume de son parallélépipède fondamental. Considérons le parallélépipède fermé

$$\Gamma = \left\{ \sum_{i=1}^{d} \lambda_i \mathbf{u}_i : |\lambda_i| \leq (ind(K))^{-\frac{1}{d}} \right\}. \tag{3.13}$$

Γ correspond au parallélépipède fondamental (fermé) de K multiplié par $2(ind(K))^{-\frac{1}{d}} = 2(|det(M)|)^{-\frac{1}{d}}$ dans chaque direction et centré à l'origine O, i.e., tous les vecteurs générateurs de K sont multipliés par $2(|det(M)|)^{-\frac{1}{d}}$. Le volume de ce parallélépipède est donc donné par : $\left| det\left(2(|det(M)|)^{-\frac{1}{d}} M\right)\right|$. D'après les propriétés de calcul des déterminants on a : $\left| det\left(2(|det(M)|)^{-\frac{1}{d}} M\right)\right| = \left|\left(2(|det(M)|)^{-\frac{1}{d}}\right)^d det(M)\right| = 2^d$. Puisque le volume de Γ (2^d) est supérieur ou égal à 2^d, il existe, d'après le premier théorème de Minkowski [126], un point entier non nul $\mathbf{w}$ à l'intérieur de Γ. Le calcul de ce point est une étape clef de la décomposition. Afin de calculer le point $\mathbf{w}$, on a besoin d'une procédure qui calcule le plus court vecteur dans un lattice, De Loera et al. [46] proposent d'utiliser l'algorithme de réduction de base de Lenstra, Lenstra, et Lovász (LLL) [89, 126] pour calculer ce vecteur. Étant donné une matrice A $(d \times d)$ dont les vecteurs colonnes génèrent un lattice, l'algorithme LLL retourne (en un temps polynomial) une matrice A' qui génère le même lattice que A. Chaque vecteur de la matrice A' est une approximation du plus court vecteur recherché.

Soit A une matrice telle que $A = (\mathbf{u}_1|\mathbf{u}_2|\cdots|\mathbf{u}_n)$, où $\mathbf{u}_i$, $(1 \leq i \leq d)$ sont les vecteurs générateurs du cône K que nous souhaitons décomposer. Afin de calculer le point $\mathbf{w}$ du parallélépipède Γ, Dyer et Kannan [49] procèdent comme suit :

D'abord, une base réduite A' de A^{-1} est calculée. Ensuite le plus court vecteur[5] non nul $\boldsymbol{\lambda}$ du lattice généré par A' est cherché dans un ensemble de combinaisons linéaires entières des colonnes de A'. Noter qu'en pratique il est souvent suffisant et utile de prendre $\boldsymbol{\lambda}$ égal au minimum des vecteurs de la matrice A'. Il a été démontré [46, 126] que la norme infinie du vecteur $\boldsymbol{\lambda}$, i.e., la valeur absolue maximale des λ_i est inférieure ou égale à $|ind(K)|^{-\frac{1}{d}}$. Puisque $\boldsymbol{\lambda}$ appartient au lattice généré par A', et A^{-1} génère le même lattice que A', il doit exister un vecteur entier $\mathbf{w}$ non nul, tel que $\boldsymbol{\lambda} = A^{-1}\mathbf{w}$, i.e., $\mathbf{w} = A\lambda$. Cela veut dire que $\mathbf{w}$ est une combinaison linéaire entière des vecteurs générateurs du cône K avec des coefficients λ_i au maximum égaux à $|ind(K)|^{-\frac{1}{d}}$. Donc $\mathbf{w}$ est bien le point du parallélépipède Γ (3.13) recherché. Noter que les vecteurs $\mathbf{w}$ et $\mathbf{u}_i$, $1 \leq i \leq d$ doivent appartenir au même demi-espace ouvert. Si ce n'est pas le cas, il suffit de considérer $-\mathbf{w}$ pour satisfaire cette condition [14].

Une fois que le vecteur $\mathbf{w}$ est calculé, il est utilisé pour produire un ensemble de cônes K_i ayant tous des indices inférieurs à l'indice du cône original $(ind(K_i) \leq ind(K)^{\frac{d-1}{d}} < ind(K))$. C'est-à-dire qu'ils sont tous plus proches du cas unimodulaire que K. Chaque

[5] En respectant la norme infinie ($\|\mathbf{x}\|_\infty = \max_i |x_i|$).

nouveau cône K_i est obtenu en remplaçant le vecteur $\mathbf{u}_i$ de K par $\mathbf{w}$, i.e.,

$$K_i = \text{pos}\{\mathbf{u}_1, \cdots, \mathbf{u}_{i-1}, \mathbf{w}, \mathbf{u}_{i+1}, \cdots, \mathbf{u}_d\}, 1 \leq i \leq d,$$

avec $[K] = \sum_{i=1}^{d} \epsilon_i [K_i]$, où $\epsilon_i = 1$ si $det(\mathbf{u}_1|\cdots|\mathbf{u}_d) \times det(\mathbf{u}_1|\cdots|\mathbf{u}_{i-1}|w|\mathbf{u}_{i+1}|\cdots|\mathbf{u}_d) \geq 0$ et $\epsilon_i = -1$ sinon. Cette procédure est répétée récursivement sur chaque cône résultant jusqu'à ce que tous les cônes deviennent unimodulaires.

Noter que la décomposition unimodulaire peut résulter en certains cônes de dimension non pleine (inférieure à la dimension de K). Mais ceci est évité grâce à la polarisation de Brion (*Brion polarization trick*) [30, 12].

Soit K^* le cône dual (polaire) de K défini par : $K^* = \{\mathbf{y} \in \mathbb{Q}^d \mid \langle \mathbf{x}, \mathbf{y} \rangle \geq 0, \forall \mathbf{x} \in K\}$. Le cône dual peut être obtenu en remplaçant les coefficients de ses contraintes (inégalités) par ceux de ses rayons (vecteurs générateurs) et vice versa. Par exemple, le cône dual K^* du cône $K = \text{pos}\{(2,1),(1,2)\} = \{(x,y) \in \mathbb{Q}^2 \mid -x+2y \geq 0 \wedge 2x+-y \geq 0\}$ est donné par $K^* = \{(x,y) \in \mathbb{Q}^2 \mid 2x+y \geq 0 \wedge x+2y \geq 0\} = \text{pos}\{(-1,2),(2,-1)\}$.

L'idée de Brion consiste à ne pas décomposer le cône K lui même, mais plutôt de décomposer son cône dual K^*. Ensuite la décomposition de K est obtenue en polarisant uniquement les cônes de dimension pleine résultant de la décomposition de K^*. En effet, les cônes de dimension inférieure sont ignorés puisque lorsqu'ils sont polarisés, ils résultent en lignes droites dont la contribution est nulle (lors du calcul de la fonction génératrice). Puisque le cône dual d'un cône unimodulaire est aussi unimodulaire, on est sûr de n'obtenir dans la décomposition de K que des cônes unimodulaires de dimension pleine (égale à la dimension de K).

Nous pouvons maintenant donner la fonction génératrice de tout cône supportant $K(P, \mathbf{v})$ d'un polytope P en un sommet *entier* $\mathbf{v}$ sous la forme :

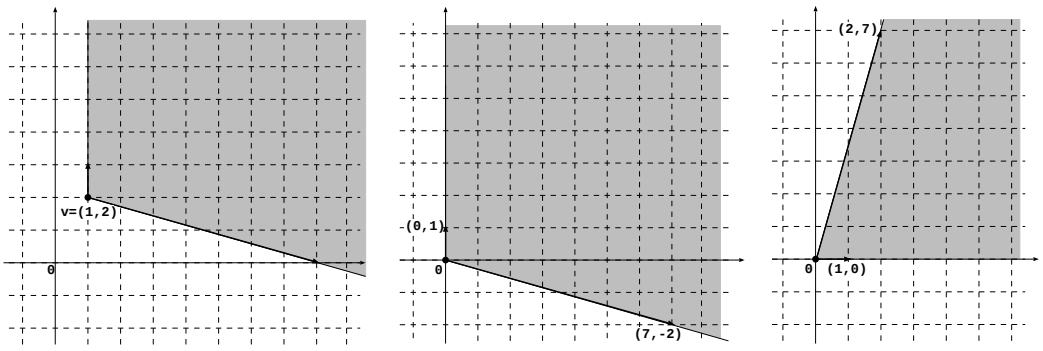
$$f(K(P,\mathbf{v}); \mathbf{x}) = \mathbf{x}^{\mathbf{v}} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}})|} \epsilon_i \frac{1}{\prod_{j=1}^{d}(1 - \mathbf{x}^{\mathbf{u}_j^i})}, \tag{3.14}$$

où, $K_{\mathbf{v}}$ est la translation du cône supportant $K(P, \mathbf{v})$ à l'origine O, $\mathcal{B}(K_{\mathbf{v}})$ est l'ensemble des cônes unimodulaires résultant de la décomposition de Barvinok, et $\mathbf{u}_j^i$ est le $j^{ème}$ vecteur générateur du $i^{ème}$ cône unimodulaire dans l'ensemble $\mathcal{B}(K_{\mathbf{v}})$.

Exemple 14. Considérons un cône supportant $K = \mathbf{v} + K_{\mathbf{v}}$, où $\mathbf{v} = (1,2)$ est un sommet entier, et $K_{\mathbf{v}} = \text{pos}\{(7,-2),(0,1)\}$ est un cône polyédrique (c'est la translation par $-\mathbf{v}$ de K). Le cône supportant K, le cône polyédrique $K_{\mathbf{v}}$ et le cône dual (polaire) $K_{\mathbf{v}}^* = \text{pos}\{(2,7),(1,0)\}$ de $K_{\mathbf{v}}$ sont montrés respectivement par les figures 3.7a, 3.7b, et 3.7c.

Pour calculer la fonction génératrice du cône supportant K, on commence par la décomposition en cônes unimodulaires du cône polaire $K_{\mathbf{v}}^*$. Cette décomposition est décrite dans [46] (exemple 6). Elle est calculée comme suit :

Le point de départ du l'algorithme est la matrice $A = \begin{pmatrix} 2 & 1 \\ 7 & 0 \end{pmatrix}$ formée par les vecteurs générateurs de $K_{\mathbf{v}}^*$. On a $det(A) = -7$, donc $K_{\mathbf{v}}^*$ n'est pas unimodulaire et doit être décomposé. Pour ce faire, on calcule $A^{-1} = \begin{pmatrix} 0 & \frac{1}{7} \\ 1 & \frac{-2}{7} \end{pmatrix}$ puis $A' = \begin{pmatrix} \frac{1}{7} & \frac{3}{7} \\ \frac{-2}{7} & \frac{1}{7} \end{pmatrix}$ la base réduite de A^{-1}. Le vecteur colonne le plus court (en respectant la norme infinie) de A'



(a) $K(P,\mathbf{v})$ (b) $K_\mathbf{v} = K(P,\mathbf{v}) - \mathbf{v}$ (c) $K^*_\mathbf{v} = dual(K_\mathbf{v})$

FIG. 3.7 – Un cône supportant (a), sa translation à l'origine (b) et le cône dual (polaire) de sa translation (c).

est $\boldsymbol{\lambda} = \begin{pmatrix} \frac{1}{7} \\ \frac{-2}{7} \end{pmatrix}$. D'où, $\mathbf{w} = A\boldsymbol{\lambda} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. En remplaçant la première colonne de A puis la deuxième par $\mathbf{w}$ on trouve les deux matrices : $B_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ et $B_2 = \begin{pmatrix} 2 & 0 \\ 7 & 1 \end{pmatrix}$

Le déterminant de B_1 est égal à -1. Le cône correspondant est unimodulaire, donc il est ajouté à l'ensemble de cônes unimodulaires $\mathcal{B}(K^*_\mathbf{v})$ avec un signe positif puisque $det(A) \times det(B_1) > 0$. On écrit : $\mathcal{B}(K^*_\mathbf{v}) = +\text{pos}\{(1,0),(0,1)\}$.

Le déterminant de B_2 est égal à 2. Le cône correspondant n'est donc pas unimodulaire. La matrice B_2 doit donc subir le même traitement que la matrice originale A, où tous les cônes qui en résultent sont multipliés par -1 puisque $det(B_2) \times det(A) < 0$. On calcule $B_2^{-1} = \begin{pmatrix} \frac{1}{2} & 0 \\ \frac{-7}{2} & 1 \end{pmatrix}$ puis sa base réduite $B'_2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{2} & \frac{1}{2} \end{pmatrix}$. Le plus court vecteur colonne de B'_2 est $\boldsymbol{\lambda} = \begin{pmatrix} \frac{1}{2} \\ \frac{-1}{2} \end{pmatrix}$, et le vecteur $\mathbf{w}$ est donné par $B_2\boldsymbol{\lambda} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$. En remplaçant la première colonne de B_2, puis la deuxième par $\mathbf{w}$, on trouve : $C_1 = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$ et $C_2 = \begin{pmatrix} 2 & 1 \\ 7 & 3 \end{pmatrix}$ $det(C_1) = 1$ et $det(C_2) = -1$, les deux cônes correspondants sont donc unimodulaires. Le premier est ajouté dans l'ensemble $\mathcal{B}$ avec un signe négatif (puisque $det(C_1) \times det(B_2) \times det(A) < 0$), et le deuxième est ajouté avec un signe positif (puisque $det(C_2) \times det(B_2) \times det(A) > 0$). La décomposition unimodulaire finale de $\mathcal{B}(K^*_\mathbf{v})$ est donc donnée par : $\mathcal{B}(K^*_\mathbf{v}) = \{+\text{pos}\{(1,0),(0,1)\}, +\text{pos}\{(2,7),(1,3)\}, -\text{pos}\{(1,3),(0,1)\}\}$. Cette décomposition est montrée par la figure 3.8.

Enfin la décomposition unimodulaire du cône $K_\mathbf{v}$ est donnée par les cônes duaux (polaires) de l'ensemble $\mathcal{B}(K^*_\mathbf{v})$. $\mathcal{B}(K_\mathbf{v}) = \{+\text{pos}\{(0,1),(1,0)\}, +\text{pos}\{(7,-2),(-3,1)\}, -\text{pos}\{(-3,1),(1,0)\}\}$. La fonction génératrice de $K(P,\mathbf{v})$ est finalement donnée en fonction du sommet $\mathbf{v}$ et des vecteurs générateurs des cônes $\mathcal{B}(K_\mathbf{v})$ sous la forme :

$$f(K(P,\mathbf{v});xy) = \frac{x^1y^2}{(1-y)(1-x)} + \frac{x^1y^2}{(1-x^7y^{-2})(1-x^{-3}y)} - \frac{x^1y^2}{(1-x^{-3}y)(1-x)}.$$

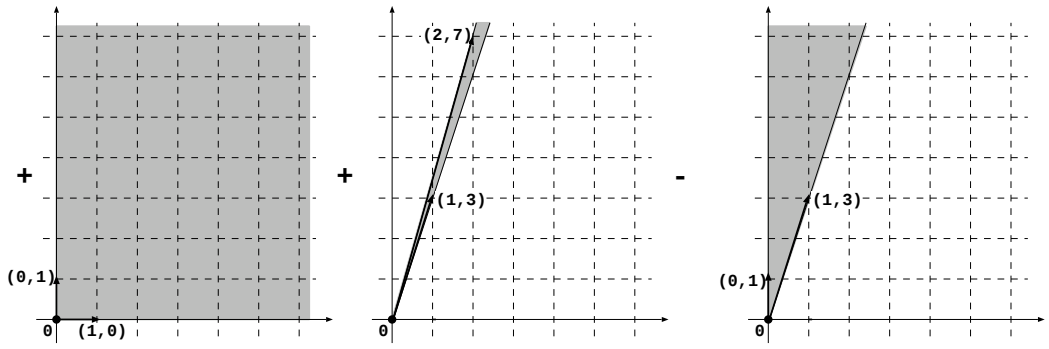


FIG. 3.8 – La décomposition unimodulaire du cône $K^*_{\mathbf{v}}$ de la figure 3.7c.

Les sommets rationnels

Nous avons vu jusque-là comment calculer la fonction génératrice d'un polytope dont tous les sommets sont entiers. La particularité de ce cas est que le point entier (unique) du parallélépipède fondamental de tout cône unimodulaire supportant en un sommet entier $\mathbf{v}$ est le sommet $\mathbf{v}$ lui même. Puisque le numérateur de chaque fonction génératrice d'un cône unimodulaire dépend du point entier de son parallélépipède fondamental, il suffit de multiplier toutes les fonctions génératrices des cônes unimodulaires par $\mathbf{x}^{\mathbf{v}}$. Lorsque le sommet $\mathbf{v}$ est *rationnel*, chaque parallélépipède fondamental d'un cône unimodulaire supportant possède un point entier différent à calculer.

Soit $K_i(P,\mathbf{v}) = K_i + \mathbf{v}$ un cône unimodulaire supportant du polytope P au sommet $\mathbf{v}$. C'est la translation au sommet $\mathbf{v}$ d'un cône polyédrique unimodulaire $K_i = \text{pos}\{\mathbf{u}^i_1, \mathbf{u}^i_2, \cdots, \mathbf{u}^i_d\}$ (K_i est l'un des cônes résultant de la décomposition unimodulaire de Barvinok). Le point entier du parallélépipède fondamental de $K_i(P,\mathbf{v})$, que nous notons $E(\mathbf{v}, K_i)$, est calculé en fonction du sommet $\mathbf{v}$ et des vecteurs générateurs de K_i comme suit :

D'abord le sommet $\mathbf{v}$ est réécrit comme une combinaison linéaire des vecteurs générateurs de K_i

$$\mathbf{v} = \sum_j \lambda_j \mathbf{u}^i_j. \tag{3.15}$$

Puisque les coordonnées du sommet $\mathbf{v}$ et les vecteurs générateurs $\mathbf{u}^i_j$ du cône K_i sont connus, il est possible de trouver les λ_j en résolvant le système (3.15). Le point $E(\mathbf{v}, K_i)$ est ensuite donné en fonction des λ_j et des vecteurs $\mathbf{u}^i_j$

$$E(\mathbf{v}, K_i) = \sum_j \lceil \lambda_j \rceil \mathbf{u}^i_j, \tag{3.16}$$

où, $\lceil . \rceil$ est la partie entière supérieure d'un nombre rationnel.

Exemple 15. Considérons la translation du cône unimodulaire $K = \text{pos}\{(1,0),(1,1)\}$ au sommet $\mathbf{v} = (\frac{1}{2}, \frac{4}{3})$ (figure 3.9). Nous avons

$$\mathbf{v} = \left(\frac{1}{2}, \frac{4}{3}\right) = \lambda_1(1,0) + \lambda_2(1,1) \Rightarrow (\lambda_1, \lambda_2) = \left(-\frac{1}{3}, \frac{4}{3}\right), \text{ donc}$$

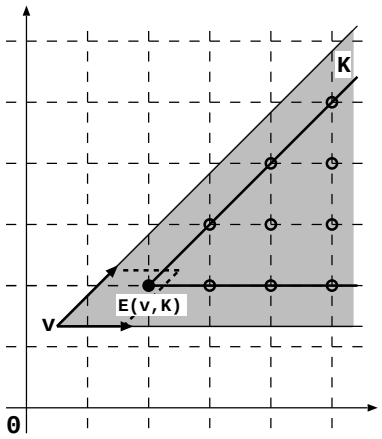


FIG. 3.9 – Le cône unimodulaire supportant au sommet rationnel $\mathbf{v} = (\frac{1}{2}, \frac{4}{3})$ (région en gris) possède la même fonction génératrice que sa translation au point entier $E(\mathbf{v}, K) = (2, 2)$ de son parallélépipède fondamental.

$$E(\mathbf{v}, K) = \left\lceil -\frac{1}{3} \right\rceil (1, 0) + \left\lceil \frac{4}{3} \right\rceil (1, 1) = (2, 2).$$

En se basant sur la décomposition de Barvinok et le calcul des points entiers dans les parallélépipèdes fondamentaux des cônes unimodulaires, on peut donner la fonction génératrice d'un polytope quelconque sous la forme :

$$f(P; \mathbf{x}) = \sum_{\mathbf{v} \in \mathcal{V}(P)} \sum_{i=1}^{|\mathcal{B}(K_\mathbf{v})|} \epsilon_i \frac{\mathbf{x}^{E(\mathbf{v}, K_i)}}{\prod_{j=1}^{d} (1 - \mathbf{x}^{\mathbf{u}_j^i})}, \tag{3.17}$$

où $\mathcal{V}(P)$ est l'ensemble de sommets du polytope P, $\mathcal{B}(K_\mathbf{v})$ est l'ensemble de cônes unimodulaires résultant de la décomposition de Barvinok du cône $K_\mathbf{v}$ ($K_\mathbf{v}$ est la translation à l'origine du cône supportant de P au sommet $\mathbf{v}$), $E(\mathbf{v}, K_i)$ est le point entier du parallélépipède fondamental de la translation du cône unimodulaire K_i au sommet $\mathbf{v}$, et $\mathbf{u}_j^i$ est le $j^{ème}$ vecteur générateur du cône K_i.

3.2.2 Évaluation des fonctions génératrices

Nous avons expliqué jusque-là, comment procéder pour calculer la fonction génératrice d'un polytope quelconque sous la forme (3.17). Nous avons également vu qu'afin de trouver le nombre de ses points entiers, il suffit d'évaluer la fonction génératrice à $\mathbf{x} = \mathbf{1}$. Or $\mathbf{x} = \mathbf{1}$ est un pôle de chaque terme de cette fonction. C'est-à-dire qu'elle ne peut être évaluée en substituant directement $\mathbf{x} = \mathbf{1}$ dans ses différents termes. Les termes de la fonction génératrice d'un polytope $f(P; \mathbf{x})$, i.e., les fonctions génératrices correspondant aux cônes unimodulaires (supportants) sont développables en séries de Laurent au voisinage de $\mathbf{x} = \mathbf{1}$, i.e., leur développement en séries entières comporte des termes ayant des puissances négatives ($f(K; \mathbf{x}) = \sum_{\mathbf{n}=-\infty}^{+\infty} a_n (\mathbf{x} - \mathbf{1})^\mathbf{n}$). Mais ces derniers termes s'éliminent entre eux dans la fonction génératrice globale. La fonction génératrice globale $f(P; \mathbf{x})$ est donc analytique au voisinage de $\mathbf{x} = \mathbf{1}$. C'est-à-dire

qu'elle est développable en séries de Taylor ($f(P;\mathbf{x}) = \sum_{\mathbf{n}=\mathbf{0}}^{\infty} a_n(\mathbf{x}-1)^{\mathbf{n}}$). Ainsi, il est clair que l'évaluation de $f(P;\mathbf{x})$ au voisinage de $\mathbf{x} = \mathbf{1}$ est simplement donnée par le terme constant (coefficient de $\mathbf{x^0}$) de son développement en série de Taylor. Tous les autres termes sont nuls lorsque $\mathbf{x}$ tend vers 1.

Plutôt que de calculer le terme constant du développement en série de Taylor de la fonction globale $f(P;\mathbf{x})$, Yoshida et al. [154, 46] proposent de calculer le terme constant du développement en série de Laurent de la fonction génératrice de chaque cône unimodulaire (supportant) et de faire leur somme (signée) pour trouver le terme constant de $f(P;\mathbf{x})$, i.e., le nombre de points entiers dans P.

Soit $f(K_i+\mathbf{v};\mathbf{x}) = \epsilon_i \frac{\mathbf{x}^{E(\mathbf{v},K_i)}}{\prod_{j=1}^{d}(1-\mathbf{x}^{\mathbf{u}_j^i})}$ la fonction génératrice de la translation d'un cône unimodulaire K_i à un point entier $\mathbf{v}$. Le calcul de $\lim_{\mathbf{x}\to\mathbf{1}} f(K_i+\mathbf{v};\mathbf{x})$ est d'abord simplifié en transformant la fonction rationnelle multivariable $f(K_i+\mathbf{v};\mathbf{x})$ en une fonction *univariable*. Pour ce faire, on a besoin d'un vecteur entier $\boldsymbol{\mu}$ non orthogonal à tous les vecteurs générateurs $\mathbf{u}_j$ ($1 \le j \le d$). Barvinok et Pommersheim [12] proposent un algorithme qui calcule un tel vecteur en un temps polynomial. Mais en pratique, il est souvent préférable de prendre un vecteur aléatoire [46] (le vecteur pris au hasard satisfait souvent la contrainte dés la première tentative). Une fois que le vecteur $\boldsymbol{\mu}$ est calculé, on applique la substitution $x_i = (s+1)^{\mu_i}$ à $f(K_i+\mathbf{v};\mathbf{x})$, on obtient

$$f(K_i+\mathbf{v};s) = \epsilon_i \frac{(s+1)^{\langle \boldsymbol{\mu}, E(\mathbf{v},K_i)\rangle}}{\prod_{j=1}^{d}(1-(s+1)^{\langle \boldsymbol{\mu}, \mathbf{u}_j^i\rangle})}.$$

Afin de n'obtenir que des exposants positifs dans le dénominateur, on doit mettre en facteur $(s+1)^c$ (dans le dénominateur), où c est la somme des exposants négatifs de $(s+1)$ dans le dénominateur. La fonction génératrice peut ensuite être réécrite sous la forme :

$$f(K_i+\mathbf{v};s) = \epsilon_i' \frac{N(s)}{D'(s)} = \epsilon_i' \frac{(s+1)^{\langle \boldsymbol{\mu}, E(\mathbf{v},K_i)\rangle - c}}{s^d D(s)}, \tag{3.18}$$

où $D(s)$ est un polynôme (non nul) avec des coefficients entiers indépendants de $\mathbf{v}$. Le terme constant de $f(K_i+\mathbf{v};\mathbf{x})$ lorsque $\mathbf{x} = \mathbf{1}$ est égal au terme constant de $f(K_i+\mathbf{v};s)$ lorsque $s = 0$. Cela revient à calculer le terme constant de la division polynomiale $\epsilon_i' \frac{N(s)}{D'(s)}$, ou d'une manière équivalente le coefficient de s^d dans la division polynomiale $\epsilon_i' \frac{N(s)}{D(s)}$, où $N(s) = (s+1)^{\langle \boldsymbol{\mu}, E(\mathbf{v},K_i)\rangle - c}$ et $D(s)$ est un polynôme en s. Pour faire ce calcul, on commence par développer le numérateur $(s+1)^{\langle \boldsymbol{\mu}, E(\mathbf{v},K_i)\rangle - c}$, jusqu'au terme s^d en utilisant la formule

$$(s+1)^n \equiv 1 + ns + \frac{n(n-1)}{2}s^2 + \cdots + \frac{n(n-1)\cdots(n-d+1)}{d!}s^d \mod s^{d+1}$$

Ensuite, la division polynomiale $\frac{N(s)}{D(s)}$ (jusqu'au terme s^d) est calculée comme suit :

Notons $N(s) \equiv a_0 + a_1 s + \cdots a_d s^d \mod s^{d+1}$, $D(s) \equiv b_0 + b_1 s + \cdots b_d s^d \mod s^{d+1}$, et $\frac{N(s)}{D(s)} \equiv c_0 + c_1 s + \cdots c_d s^d \mod s^{d+1}$. Les coefficients $c_0, c_1, \cdots, c_d$ sont donnés par la fonction de récurrence

$$\begin{aligned} c_0 &= \tfrac{a_0}{b_0}, \\ c_k &= \tfrac{1}{b_0}(a_k - b_1 c_{k-1} - b_2 c_{k-2} - \cdots - b_k c_0) \text{ pour } k = 1, 2, \cdots, d. \end{aligned} \tag{3.19}$$

Exemple 16. Dans cet exemple nous évaluons la fonction génératrice du polytope P de l'exemple 12.

$$f(P;\mathbf{x}) = \frac{x_1^2}{(1-x_1^{-1})(1-x_1^{-1}x_2)} + \frac{x_2^2}{(1-x_2^{-1})(1-x_1x_2^{-1})} + \frac{1}{(1-x_1)(1-x_2)}.$$

Afin d'évaluer $f(P;\mathbf{x})$ à $\mathbf{x} = \mathbf{1}$, on peut choisir $\boldsymbol{\mu} = (1,-1)$ puisqu'il n'est pas orthogonal à tous les vecteurs générateurs. En substituant $x_1 = (s+1)^{\mu_1}$ et $x_2 = (s+1)^{\mu_2}$ dans $f(P;\mathbf{x})$ (3.7), nous obtenons

$$f(P;s) = \frac{(s+1)^2}{(1-(s+1)^{-1})(1-(s+1)^{-2})} + \frac{(s+1)^{-2}}{(1-(s+1))(1-(s+1)^2)} + \frac{1}{(1-(s+1))(1-(s+1)^{-1})}.$$

Afin de n'obtenir que des puissances positives dans le dénominateur, nous multiplions le numérateur et le dénominateur de chaque terme par $(s+1)^{-c}$, où c est la somme des puissances négatives dans le dénominateur. Ici c est égal respectivement à -3, 0 et -1. La nouvelle fonction est

$$f(P;s) = \frac{(s+1)^5}{(1-(s+1))(1-(s+1)^2)} + \frac{(s+1)^{-2}}{(1-(s+1))(1-(s+1)^2)} - \frac{(s+1)}{(1-(s+1))(1-(s+1))} \tag{3.20}$$

ou

$$f(P;s) = \frac{(s+1)^5}{s^2(s+2)} + \frac{(s+1)^{-2}}{s^2(s+2)} - \frac{(s+1)}{s^2}. \tag{3.21}$$

Le développement de $\frac{1}{s+2}$ est donné par

$$\frac{1}{s+2} \equiv \frac{1}{2}\left(1 - \frac{1}{2}s + \frac{1}{4}s^2\right) \mod s^3 \tag{3.22}$$

Après le développement de $(s+1)^5$ et $(s+1)^{-2}$ et la simplification des différents termes nous obtenons

$$s^2 f(P;s) \equiv \left(\frac{1}{2} + \frac{9}{4}s + \frac{31}{8}s^2\right) + \left(\frac{1}{2} - \frac{5}{4}s + \frac{17}{8}s^2\right) - \left(1 + s + 0s^2\right) \mod s^3.$$

Enfin, le nombre de points entiers dans le polytope P est donné par la somme des coefficients de s^2 dans les trois polynômes, i.e., $\frac{31}{8} + \frac{17}{8} - 0 = 6$.

La méthode de comptage de Barvinok décrite jusque-là est implémentée dans `LattE` [46]. Elle permet de manipuler des polytopes non paramétrés quelconques. De Loera et al. l'ont également étendue pour traiter quelques formes de polytopes paramétrés à travers ce qu'ils appellent *l'algorithme de Barvinok homogénéisé* [44]. Cet algorithme supporte un seul paramètre et ne considère que des dilatations pP par un facteur p d'un polytope non paramétré P. Cela veut dire que tous les sommets sont de la forme $p\mathbf{v}$ et qu'aucune décomposition en domaines de validité n'est nécessaire. À noter également que la méthode de De Loera et al. ne calcule pas les quasi-polynômes d'Ehrhart mais plutôt des séries d'Ehrhart qui sont des séries entières formelles plus adaptées à leur domaine d'application.

3.3 Dénombrement des points entiers de polytopes paramétrés en utilisant les fonctions génératrices

Dans cette section, nous décrivons notre généralisation de l'algorithme de Barvinok Pour traiter des polytopes paramétrés quelconques (ayant plusieurs paramètres). Cet algorithme est le fruit d'un travail de collaboration avec Sven Verdoolaege de l'université catholique de Leuven (Belgique) et Kristof Beyls de l'université de Ghent (Belgique) [148, 147].

Considérons un polytope paramétré de dimension pleine (sans égalités)

$$P = \{\mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \geq B\mathbf{p} + \mathbf{b}\}, \tag{3.23}$$

où $\mathbf{p}$ est un vecteur de n paramètres. Nous considérons, sans perte de généralité, que le polytope P est de dimension pleine. En effet, tout polytope de dimension non pleine peut être transformé en un polytope de dimension pleine en éliminant ses égalités (voir la section 3.3.4).

Afin de calculer le nombre de points entiers dans un polytope paramétré de la forme (3.23), nous nous basons sur le calcul de ses fonctions génératrices paramétrées dont l'évaluation résulte en des *quasi-polynômes d'Ehrhart*. Puisque notre méthode est une version paramétrée de l'algorithme de Barvinok [14, 12], la fonction génératrice (paramétrée) du polytope (3.23) est donnée par la somme des fonctions génératrices des cônes supportants en ses différents sommets. Cela veut dire qu'avant tout, nous devons être capable de calculer (i) les sommets paramétrés du polytope (ii) les vecteurs générateurs de ses cônes supportants en ces sommets.

Comme nous l'avons présenté dans le chapitre 2 (sections 2.5.1 et 2.5.2), les sommets d'un polytope paramétré sont des expressions affines de paramètres, qui peuvent n'être valides que sur des sous-ensembles de valeurs des paramètres (domaines de validité). Clauss et Loechner [39] ont montré que tout polytope paramétré de la forme (3.23) peut être décomposé en un ensemble de polytopes homothétiques, chacun d'entre eux étant défini sur un domaine de validité, où tous ses sommet sont valides. Par conséquent, la fonction génératrice d'un polytope paramétré quelconque (3.23) est donnée par un ensemble de fonctions génératrices. Chacune d'entre elles correspond à un polytope homothétique-bordé, et donc n'est valide que sur un sous-ensemble de valeurs des paramètres (domaine de validité). Nous utilisons l'algorithme de Clauss/Loechner implémenté dans la `Polylib` [95] pour calculer les différents domaines de validité et les sommets qui sont valides sur chacun d'entre eux. Ensuite, nous calculons une fonction génératrice pour chaque domaine de validité.

Dans ce qui suit nous présentons comment nous calculons et évaluons la fonction génératrice pour un domaine de validité D, sur lequel un sous-ensemble $\mathcal{V}_D(P)$ des sommets de P sont valides. Lorsqu'il y a plusieurs domaines de validité, nous appliquons le même algorithme autant de fois qu'il y a de domaines.

3.3.1 Calcul de la fonction génératrice paramétrée

Comme c'est le cas pour l'algorithme original de Barvinok, la fonction génératrice d'un cône supportant $K(P, \mathbf{v}(\mathbf{p}))$ en un sommet paramétré $\mathbf{v}(\mathbf{p})$ dépend des vecteurs générateurs du cône $K(P, \mathbf{v}(\mathbf{p})) - \mathbf{v}(\mathbf{p})$ et des coordonnées du sommet $\mathbf{v}(\mathbf{p})$, puisque

les rayons d'un polyèdre paramétré sont toujours indépendants des paramètres dans un domaine de validité donné (voir le théorème 2 ou encore [12]). En effet, les rayons (les vecteurs générateurs en l'occurrence) sont définis par les directions des facettes d'un polyèdre. Ce sont des vecteurs orthogonaux aux vecteurs normaux des facettes. Or, ces derniers vecteurs sont simplement les coefficients des variables dans l'équation (3.23), i.e., les lignes de la matrice A. Il est donc clair que les vecteurs générateurs de la translation du cône supportant $K(P, \mathbf{v}(\mathbf{p}))$ à l'origine $(K(P, \mathbf{v}(\mathbf{p})) - \mathbf{v}(\mathbf{p}))$ sont indépendants des paramètres. Par conséquent, sa décomposition en cônes unimodulaires reste la même que celle de Barvinok décrite dans la section 3.2.1. Autrement dit, le dénominateur de la fonction génératrice d'un polytope paramétré est calculé de la même façon que pour un polytope non paramétré. Il ne reste donc qu'à définir son numérateur. Deux cas sont possibles :

Sommets entiers

Un sommet paramétré $\mathbf{v}(\mathbf{p})$ est dit *entier* si toutes ses coordonnées sont des fonctions affines à coefficients entiers, i.e., $\mathbf{v}(\mathbf{p}) = (g_1(\mathbf{p}), g_2(\mathbf{p}), \cdots, g_d(\mathbf{p}))$, où $g_i(\mathbf{p}) = a_0 + \sum_{j=1}^{n} a_j p_j$, avec $a_j \in \mathbb{Z}, \forall j \in \{0, 1, \cdots, n\}$.

Lorsqu'un sommet est entier, le calcul de sa fonction génératrice est facile : nous multiplions la fonction génératrice de tout cône unimodulaire (résultant de la décomposition de $K(P, \mathbf{v}(\mathbf{p})) - \mathbf{v}(\mathbf{p})$) par $\mathbf{x}^{\mathbf{v}(\mathbf{p})} = x_1^{g_1(\mathbf{p})} x_2^{g_2(\mathbf{p})} \cdots x_d^{g_d(\mathbf{p})}$. La fonction génératrice sur un domaine de validité D est donc donnée par :

$$f_D(P; \mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p}) \in \mathcal{V}_D(P)} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}(\mathbf{p})})|} \epsilon_i \frac{x_1^{g_1(\mathbf{p})} x_2^{g_2(\mathbf{p})} \cdots x_d^{g_d(\mathbf{p})}}{\prod_{j=1}^{d} (1 - \mathbf{x}^{\mathbf{u}_j^i})}, \tag{3.24}$$

où $\mathcal{V}_D(P)$ est l'ensemble de sommets de P qui sont valides sur le domaine D, $\mathcal{B}(K_{\mathbf{v}})$ est la décomposition unimodulaire du cône $K_{\mathbf{v}(\mathbf{p})} = K(P, \mathbf{v}(\mathbf{p})) - \mathbf{v}(\mathbf{p})$, et $\mathbf{u}_j^i$ est le $j^{ème}$ vecteur générateur du cône unimodulaire K_i.

Exemple 17. Considérons le polytope paramétré donné par le système d'inégalités $P = \{(x, y) \in \mathbb{Q}^d \mid 0 \leq x \leq -y + p \ \wedge \ 0 \leq y \leq 10\}$. La décomposition de ce polytope en polytopes homothétiques résulte en deux polytopes : P_1 défini sur le domaine de validité $0 \leq p \leq 10$ et ayant trois sommets $\mathbf{v}_1 = (0, 0)$, $\mathbf{v}_2 = (p, 0)$ et $\mathbf{v}_3 = (0, p)$, et P_2 défini sur le domaine de validité $p \geq 10$ et ayant quatre sommets $\mathbf{v}_1 = (0, 0)$, $\mathbf{v}_2 = (p, 0)$, $\mathbf{v}_4 = (0, 10)$ et $\mathbf{v}_5 = (p - 10, 10)$. Les sommets de P_1 et de P_2 sont entiers.

Les translations à l'origine des cônes supportants aux sommets du polytope P_1 sont : $K_{\mathbf{v}_1} = \text{pos}\{(1, 0), (0, 1)\}$, $K_{\mathbf{v}_2} = \text{pos}\{(-1, 0), (-1, 1)\}$ et $K_{\mathbf{v}_3} = \text{pos}\{(0, -1), (1, -1)\}$. Ces cônes sont déjà unimodulaires, donc aucune décomposition n'est nécessaire. Les fonctions génératrices correspondantes sont respectivement :

$$f(K_{\mathbf{v}_1}; xy) = \frac{1}{(1 - x)(1 - y)}, \ f(K_{\mathbf{v}_2}; xy) = \frac{x^p}{(1 - x^{-1})(1 - x^{-1}y)},$$

$$f(K_{\mathbf{v}_3}; xy) = \frac{y^p}{(1 - y^{-1})(1 - xy^{-1})}.$$

La fonction génératrice du polytope P_1 est donnée par : $f(P_1;xy) = f(K_{\mathbf{v}_1};xy) + f(K_{\mathbf{v}_2};xy)+f(K_{\mathbf{v}_3};xy)$. Pour P_2, nous avons les cônes $K_{\mathbf{v}_1} = \text{pos}\{(1,0),(0,1)\}$, $K_{\mathbf{v}_2} = \text{pos}\{(-1,0),(-1,1)\}$, $K_{\mathbf{v}_4} = \text{pos}\{(0,-1),(1,0)\}$, et $K_{\mathbf{v}_5} = \text{pos}\{(-1,0),(1,-1)\}$. Ces cônes sont de nouveau tous unimodulaires, leurs fonctions génératrices sont :

$$f(K_{\mathbf{v}_1};xy) = \frac{1}{(1-x)(1-y)},\ f(K_{\mathbf{v}_2};xy) = \frac{x^p}{(1-x^{-1})(1-x^{-1}y)},$$

$$f(K_{\mathbf{v}_4};xy) = \frac{y^{10}}{(1-y^{-1})(1-x)},\ f(K_{\mathbf{v}_5};xy) = \frac{x^{(p-10)}y^{10}}{(1-x^{-1})(1-xy^{-1})}.$$

La fonction génératrice du polytope P_2 est :

$$f(P_2;xy) = f(K_{\mathbf{v}_1};xy) + f(K_{\mathbf{v}_2};xy) + f(K_{\mathbf{v}_4};xy) + f(K_{\mathbf{v}_5};xy).$$

Enfin, la fonction génératrice globale du polytope P est donnée par :

$$f(P;xy) = \begin{cases} f(P_1;xy) \text{ si } 0 \leq N \leq 10, \\ f(P_2;xy) \text{ si } N \geq 10. \end{cases}$$

Sommets rationnels

Un sommet paramétré est dit *rationnel* si au moins une de ses coordonnées $g_i(\mathbf{p})$ est une fonction affine *rationnelle* des paramètres. Dans ce cas, le numérateur de la fonction génératrice est donné en fonction du point entier (paramétré) $E(\mathbf{v}(\mathbf{p}),K_i)$ du parallélépipède fondamental de la translation (au sommet $\mathbf{v}(\mathbf{p})$) de chaque cône unimodulaire K_i.

$$f_D(P;\mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p})\in\mathcal{V}_D(P)} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}(\mathbf{p})})|} \epsilon_i \frac{\mathbf{x}^{E(\mathbf{v}(\mathbf{p}),K_i)}}{\prod_{j=1}^d (1-\mathbf{x}^{\mathbf{u}_j^i})}, \tag{3.25}$$

où $E(\mathbf{v}(\mathbf{p}),K_i)$ est donné en fonction du sommet paramétré $\mathbf{v}(\mathbf{p}) = (g_1(\mathbf{p}),\cdots,g_d(\mathbf{p}))$ et des vecteurs générateurs $\mathbf{u}_j^i$ du cône K_i, comme suit :

$$E(\mathbf{v}(\mathbf{p}),K_i) = \sum_{j=1}^{d} \lceil \lambda_j(\mathbf{p}) \rceil \mathbf{u}_j^i, \tag{3.26}$$

où les $\lambda_j(\mathbf{p})$ sont des fonctions affines rationnelles des paramètres. Ce sont les solutions du système $\mathbf{v}(\mathbf{p}) = \sum_{j=1}^d \lambda_j(\mathbf{p})\mathbf{u}_j^i$. La fonction génératrice (3.25) est donc donnée par :

$$f_D(P;\mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p})\in\mathcal{V}_D(P)} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}(\mathbf{p})})|} \epsilon_i \frac{x_1^{\sum_{j=1}^d \lceil \lambda_j(\mathbf{p})\rceil u_{j1}^i} x_2^{\sum_{j=1}^d \lceil \lambda_j(\mathbf{p})\rceil u_{j2}^i} \cdots x_d^{\sum_{j=1}^d \lceil \lambda_j(\mathbf{p})\rceil u_{jd}^i}}{\prod_{j=1}^d (1-\mathbf{x}^{\mathbf{u}_j^i})}. \tag{3.27}$$

À noter que nous pouvons également écrire cette fonction génératrice en fonction de parties entières inférieures en remplaçant $\lceil \lambda_j(\mathbf{p}) \rceil$ par $-\lfloor -\lambda_j(\mathbf{p}) \rfloor$.

Les exposants $\lceil \lambda_j(\mathbf{p}) \rceil u_{jk}^i$ sont des nombres périodiques qui peuvent être représentés par des tableaux comme dans la méthode d'interpolation [39]. En effet, toute partie

supérieure d'une fonction affine rationnelle des paramètres ($\lceil \lambda_j(\mathbf{p}) \rceil$) peut être écrite comme une somme de la fonction affine $\lambda_j(\mathbf{p})$ et d'un nombre périodique $U^j_{\mathbf{p}}$ ayant la forme d'un tableau, comme suit :

$$\lceil \lambda_j(\mathbf{p}) \rceil = \lambda_j(\mathbf{p}) + \frac{(-\lambda'_j(\mathbf{p})) \bmod q_j}{q_j},$$

avec $\lambda'_j(\mathbf{p}) = \lambda_j(\mathbf{p}) \times q_j$, où q_j est le dénominateur commun des coefficients de $\lambda_j(\mathbf{p})$ (y compris la constante), i.e., $\lambda'_j(\mathbf{p})$ est une fonction entière. $U^j_{\mathbf{p}} = \frac{(-\lambda'_j(\mathbf{p})) \bmod q_j}{q_j}$ est un nombre p-périodique, où p est le nombre de paramètres qui apparaissent dans $\lambda'_j(\mathbf{p})$ et dont les coefficients ne sont pas des multiples de q_j. p est donc la dimension du tableau qui représente le nombre périodique $U^j_{\mathbf{p}}$. En substituant $\lceil \lambda_j(\mathbf{p}) \rceil = (\lambda_j(\mathbf{p}) + U^j_{\mathbf{p}})$ dans (3.27), on obtient

$$f_D(P;\mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p}) \in \mathcal{V}_D(P)} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}})|} \epsilon_i \frac{x_1^{\sum_{j=1}^d (\lambda_j(\mathbf{p})+U^j_{\mathbf{p}})u^i_{j1}} x_2^{\sum_{j=1}^d (\lambda_j(\mathbf{p})+U^j_{\mathbf{p}})u^i_{j2}} \cdots x_d^{\sum_{j=1}^d (\lambda_j(\mathbf{p})+U^j_{\mathbf{p}})u^i_{jd}}}{\prod_{j=1}^d (1-\mathbf{x}^{\mathbf{u}^i_j})}.$$

Sachant que $\sum_{j=1}^d (\lambda_j(\mathbf{p}) + U^j_{\mathbf{p}})u^i_{jk} = \sum_{j=1}^d \lambda_j(\mathbf{p})u^i_{k1} + \sum_{j=1}^d U^j_{\mathbf{p}}u^i_{jk} = g_k(\mathbf{p}) + U'^k_{\mathbf{p}}$, la fonction génératrice devient

$$f_D(P;\mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p}) \in \mathcal{V}_D(P)} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}})|} \epsilon_i \frac{x_1^{g_1(\mathbf{p})+U'^1_{\mathbf{p}}} x_2^{g_2(\mathbf{p})+U'^2_{\mathbf{p}}} \cdots x_d^{g_d(\mathbf{p})+U'^d_{\mathbf{p}}}}{\prod_{j=1}^d (1-\mathbf{x}^{\mathbf{u}^i_j})}, \tag{3.28}$$

où $g_k(\mathbf{p})$ est la $k^{ème}$ coordonnée du sommet rationnel, et $U'^k_{\mathbf{p}}$ est un nombre périodique.

Comme nous l'avons décrit dans la section 3.1.2, la représentation des nombres périodiques par des tableaux peut être exponentielle (lorsque les périodes sont grandes). Pour cette raison, nous proposons une alternative où nous gardons la fonction génératrice en fonction des parties entières (supérieures ou inférieures) des fonctions rationnelles de paramètres. Ceci permet de prévenir le comportement exponentiel des nombres périodiques, même si parfois (lorsque les périodes sont petites) il est préférable d'utiliser la représentation sous forme de tableaux, puisque les opérations d'addition et de multiplication sur les nombres périodiques sous forme de tableaux sont complètement simplifiables.

3.3.2 Évaluation des fonctions génératrices paramétrées

Pour calculer le quasi-polynôme d'Ehrhart à partir d'une fonction génératrice correspondant aux points entiers dans un polytope paramétré, il suffit de l'évaluer à $\mathbf{x} = \mathbf{1}$. Considérons de nouveau la fonction génératrice paramétrée (3.25)

$$f_D(P;\mathbf{x}) = \sum_{\mathbf{v}(\mathbf{p}) \in \mathcal{V}_D(P)} \sum_{i=1}^{|\mathcal{B}(K_{\mathbf{v}(\mathbf{p})})|} \epsilon_i \frac{\mathbf{x}^{E(\mathbf{v}(\mathbf{p}),K_i)}}{\prod_{j=1}^d (1-\mathbf{x}^{\mathbf{u}^i_j})},$$

où

$$E(\mathbf{v}(\mathbf{p}), K_i) = \begin{cases} \mathbf{v}(\mathbf{p}) \text{ si } \mathbf{v}(\mathbf{p}) \in \mathbb{Z}^d, \\ \sum_{j=1}^d \lceil \lambda_j(\mathbf{p}) \rceil \mathbf{u}^i_j = -\sum_{j=1}^d \lfloor -\lambda_j(\mathbf{p}) \rfloor \mathbf{u}^i_j \text{ si } \mathbf{v}(\mathbf{p}) \in \mathbb{Q}^d, \end{cases} \tag{3.29}$$

où les $\lambda_j(\mathbf{p})$s sont les solutions de $\mathbf{v}(\mathbf{p}) = \sum_{j=1}^{d} \lambda_j(\mathbf{p})\mathbf{u}_j^i$.

Comme dans le cas non paramétré, nous commençons par choisir un vecteur $\boldsymbol{\mu}$ non orthogonal à tous les vecteurs générateurs des cônes unimodulaires, et nous effectuons la substitution $x_i = (s+1)^{\mu_i}$. Nous obtenons le numérateur de la fonction génératrice (3.25) sous la forme

$$N_{\mathbf{p}}(s) = (s+1)^{\langle \boldsymbol{\mu}, E(\mathbf{v}(\mathbf{p}), K_i)\rangle - c} = (s+1)^{\Lambda(\mathbf{p})}, \tag{3.30}$$

où $\Lambda(\mathbf{p})$ est un polynôme multivariable de degré 1 si le sommet $\mathbf{v}(\mathbf{p})$ est entier, et un *quasi-polynôme* de degré 1 sinon, i.e., une combinaison linéaire rationnelle des parties entières inférieures de fonctions affines rationnelles des paramètres. Le dénominateur $D(S)$ quant à lui reste le même que celui du cas non paramétré.

Afin de calculer le coefficient de s^d dans $\frac{N_{\mathbf{p}}(s)}{D(s)}$ correspondant au nombre de points entiers du polytope (ici ce nombre est un quasi-polynôme d'Ehrhart), nous avons besoin de calculer les $d+1$ premiers coefficients de $N_{\mathbf{p}}(s)$ donnés par

$$n_i(\mathbf{p}) = \binom{\Lambda(\mathbf{p})}{i} = \frac{\prod_{j=0}^{i-1}(\Lambda(\mathbf{p}) - j)}{i!} \quad \text{pour } 0 \leq i \leq d. \tag{3.31}$$

Chaque coefficient $n_i(\mathbf{p})$ dans la formule ci-dessus est donné par le produit d'au maximum d quasi-polynômes de degrés 1 chacun. Ce qui veut dire que chaque coefficient $n_i(\mathbf{p})$ est un quasi-polynôme de degré inférieur ou égal à d. Le coefficient de s^d dans $\frac{N_{\mathbf{p}}(s)}{D(s)}$ n'est qu'une combinaison linéaire des coefficients $n_i(\mathbf{p})$ (selon l'équation (3.19)), donc son degré reste au maximum égal à d. Enfin la somme des différents coefficients de s^d provenant de tous les termes de la fonction génératrice est évidemment de degré inférieur ou égal à d. Donc, le résultat final (nombre de points entiers dans un polytope paramétré) calculé à partir d'une fonction génératrice est un quasi-polynôme de degré au maximum égal à d, ce qui vérifie le théorème fondamental d'Ehrhart/Clauss (théorème 4 du chapitre 2).

Note 5. *Afin d'éviter qu'une fonction génératrice d'un sommet ne soit calculée plusieurs fois (puisqu'un même sommet peut apparaître dans plusieurs domaines de validité), nous commençons par calculer les fonctions génératrices de tous les sommets. Ensuite, pour chaque domaine de validité nous faisons la somme des seules fonctions génératrices dont les sommets correspondants sont valides sur le domaine considéré.*

La méthode globale de comptage de points entiers dans un polytope paramétré quelconque à travers les fonctions génératrices de Barvinok est résumée par l'algorithme 1.

Exemple 18. Considérons une version paramétrée du polytope de l'exemple 12

$$P = \{\, \mathbf{x} \mid x_1 \geq 0 \wedge x_2 \geq 0 \wedge 2x_1 + 2x_2 \leq p \,\}.$$

Ce polytope possède un seul domaine de validité $p \geq 0$ sur lequel sont valides les sommets $\mathbf{v}_1 = (0,0)$, $\mathbf{v}_2 = (\frac{p}{2}, 0)$ et $\mathbf{v}_3 = (0, \frac{p}{2})$. Les vecteurs générateurs correspondant aux cônes supportants K_1, K_2 et K_3 de P aux sommets $\mathbf{v}_1, \mathbf{v}_2$ et $\mathbf{v}_3$ sont respectivement : $\{(1,0),(0,1)\}$, $\{(-1,0),(-1,1)\}$ et $\{(0,-1),(1,-1)\}$. Ces cônes supportants sont tous

Algorithme 1 Dénombrement des points entiers de polytopes paramétrés

1. Pour chaque sommet $\mathbf{v}_i(\mathbf{p}) \in \mathcal{V}(P)$
   (a) Calculer le cône supportant $\text{cone}(P, \mathbf{v}_i(\mathbf{p}))$
   (b) Soint $K = \text{cone}(P, \mathbf{v}_i(\mathbf{p})) - \mathbf{v}_i(\mathbf{p})$ la translation à l'origine de $\text{cone}(P, \mathbf{v}_i(\mathbf{p}))$
   (c) Soint $\{(\epsilon_j, K_j)\} = \mathcal{B}(K)$ la décomposition unimodulaire de K
   (d) Pour chaque cône unimodulaire K_j
      i. Calculer la fonction génératrice $f(K_j; \mathbf{x})$
   (e) $f(\text{cone}(P, \mathbf{v}_i(\mathbf{p})); \mathbf{x}) = \sum_j \epsilon_j \mathbf{x}^{E(\mathbf{v}_i(\mathbf{p}), K_j)} f(K_j; \mathbf{x})$
2. Pour chaque domaine de validité C_k de P
   (a) $f_{C_k}(P; \mathbf{x}) = \sum_{\mathbf{v}_i \in \mathcal{V}_{C_k}(P)} f(\text{cone}(P, \mathbf{v}_i(\mathbf{p})); \mathbf{x})$
   (b) Evaluer $f_{C_k}(P; \mathbf{1})$

unimodulaires, donc aucune décomposition n'est nécessaire. Puisque les sommets $\mathbf{v}_2$ et $\mathbf{v}_3$ sont rationnels, on doit calculer les points $E(K_2, \mathbf{v}_2)$ et $E(K_3, \mathbf{v}_3)$ en utilisant l'équation (3.29).

$$\mathbf{v}_2 = (\frac{p}{2}, 0) = -\frac{p}{2}(-1, 0) + 0(-1, 1) \Rightarrow E(K_2, \mathbf{v}_2) = \left\lceil -\frac{p}{2} \right\rceil (-1, 0) = \left(\left\lfloor \frac{p}{2} \right\rfloor, 0 \right).$$

$$\mathbf{v}_3 = (0, \frac{p}{2}) = -\frac{p}{2}(0, -1) + 0(1, -1) \Rightarrow E(K_3, \mathbf{v}_3) = \left\lceil -\frac{p}{2} \right\rceil (0, -1) = \left(0, \left\lfloor \frac{p}{2} \right\rfloor \right).$$

D'autre part on a $E(K_1, \mathbf{v}_1) = \mathbf{v}_1 = (0, 0)$. D'où la fonction génératrice de P est

$$f(P; x_1 x_2) = \frac{1}{(1 - x_1)(1 - x_2)} + \frac{x_1^{\lfloor \frac{p}{2} \rfloor}}{(1 - x_1^{-1})(1 - x_1^{-1} x_2)} + \frac{x_2^{\lfloor \frac{p}{2} \rfloor}}{(1 - x_2^{-1})(1 - x_1 x_2^{-1})}. \quad (3.32)$$

Puisque $\left\lfloor \frac{p}{2} \right\rfloor = \frac{p}{2} - \frac{p \bmod 2}{2} = \frac{p}{2} + \left[0, -\frac{1}{2}\right]_p$, la fonction génératrice ci-dessus peut être écrite sous la forme

$$f(P; x_1 x_2) = \frac{1}{(1 - x_1)(1 - x_2)} + \frac{x_1^{\frac{p}{2} + \left[0, -\frac{1}{2}\right]_p}}{(1 - x_1^{-1})(1 - x_1^{-1} x_2)} + \frac{x_2^{\frac{p}{2} + \left[0, -\frac{1}{2}\right]_p}}{(1 - x_2^{-1})(1 - x_1 x_2^{-1})}. \quad (3.33)$$

Nous évaluons maintenant ces fonctions à $\mathbf{x} = \mathbf{1}$ pour obtenir le quasi-polynôme d'Ehrhart correspondant au nombre de points entiers du polytope P.

En choisissant le vecteur $\boldsymbol{\mu} = (1, -1)$ et en effectuant la substitution $(x_1, x_1) = ((s+1)^{\mu_1}, (s+1)^{\mu_2}) = ((s+1), (s+1)^{-1})$ dans (3.33), on obtient

$$f(P; s) = -\frac{1}{s(1 - (s+1)^{-1})} + \frac{(s+1)^{\lfloor \frac{p}{2} \rfloor}}{(1 - (s+1)^{-1})(1 - (s+1)^{-2})} + \frac{(s+1)^{-\lfloor \frac{p}{2} \rfloor}}{s(1 - (s+1)^2)}. \quad (3.34)$$

En multipliant le numérateur et le dénominateur de chaque terme par $(s+1)^{-c}$, où c est la somme des exposants négatifs de $(s+1)$ dans le dénominateur, on obtient :

$$f(P; s) = -\frac{s+1}{s^2} + \frac{(s+1)^{\lfloor \frac{p}{2} \rfloor + 3}}{s^2(s+2)} + \frac{(s+1)^{-\lfloor \frac{p}{2} \rfloor}}{s^2(s+2)}.$$

Pour évaluer $f(P;0)$, il suffit de calculer de coefficient de s^2 dans :

$$\frac{N(s)}{D(s)} = \frac{(s+1)^{\lfloor \frac{p}{2} \rfloor + 3} + (s+1)^{-\lfloor \frac{p}{2} \rfloor}}{s+2} - (s+1). \tag{3.35}$$

Le deuxième terme ne contribue pas au coefficient de s^2. Le numérateur du premier terme est développé selon (3.31) en :

$$(s+1)^{\lfloor \frac{p}{2} \rfloor + 3} \equiv 1 + \left(\left\lfloor \frac{p}{2} \right\rfloor + 3 \right) s + \left(\frac{1}{2} \left\lfloor \frac{p}{2} \right\rfloor^2 + \frac{5}{2} \left\lfloor \frac{p}{2} \right\rfloor + 3 \right) s^2 \mod s^3$$

et

$$(s+1)^{-\lfloor \frac{p}{2} \rfloor} \equiv 1 - \left\lfloor \frac{p}{2} \right\rfloor s + \left(\frac{1}{2} \left\lfloor \frac{p}{2} \right\rfloor^2 + \frac{1}{2} \left\lfloor \frac{p}{2} \right\rfloor \right) s^2 \mod s^3.$$

Le dénominateur du premier terme est obtenu selon (3.19) sous la forme

$$\frac{1}{s+2} \equiv \frac{1}{2} \left(1 - \frac{1}{2} s + \frac{1}{4} s^2 \right) \mod s^3 \tag{3.36}$$

Enfin, le coefficient de s^2 est donné par :

$$\mathcal{E}(P;p) = f(P;0) = \frac{1}{2} \left\lfloor \frac{p}{2} \right\rfloor^2 + \frac{3}{2} \left\lfloor \frac{p}{2} \right\rfloor + 1. \tag{3.37}$$

À noter qu'il existe d'autres représentations de quasi-polynômes qui sont équivalentes à celle des parties entières inférieures, telles que les fonctions *modulos* ou *fractionnelles*. Ces représentations sont toutes polynomiales. Par exemple, la représentation fractionnelle ($\{x\} = x - \lfloor x \rfloor$) de la fonction génératrice ci-dessus est :

$$\mathcal{E}(P;p) = f(P;0) = \frac{1}{8} p^2 + \left(-\frac{1}{2} \left\{ \frac{1}{2} p \right\} + \frac{3}{4} \right) p - \frac{5}{4} \left\{ \frac{1}{2} p \right\} + 1.$$

La représentation fractionnelle est celle qui est effectivement implémentée dans la librairie `barvinok` (en plus de la représentation originale de Clauss/Loechner).

Contrairement aux représentations précédentes, la représentation des nombres périodiques par des tableaux est exponentielle dans le pire des cas. Pour notre exemple, cette représentation résulte en :

$$\mathcal{E}(P;p) = f(P;0) = \frac{1}{3} p^2 + \left[\frac{3}{4}, \frac{1}{2} \right]_p p + \left[1, \frac{3}{8} \right]_p. \tag{3.38}$$

Cette représentation peut être bénéfique dans certains cas, mais elle est exponentielle dans le pire des cas. Imaginons, par exemple, qu'au lieu de la contrainte $2x_1 + 2x_2 \leq p$, on avait $Nx_1 + Nx_2 \leq p$, où N est une constante relativement grande. Dans ce cas, on aurait obtenu (3.37) pratiquement de la même taille que pour 2 (on remplace simplement le dénominateur 2 par N). Alors que pour (3.38), on aurait obtenu les nombres périodiques sous forme de tableaux de taille N chacun. Il convient aussi de noter qu'il est difficile d'automatiser la simplification des expressions avec des parties entières inférieures/supérieures ou des expressions fractionnelles. Verdoolaege [142] propose quelques règles de simplifications mais elles demeurent limitées.

3.3.3 Complexité de l'algorithme

Lemme 2. *Lorsque la dimension est fixée, la taille de la décomposition en domaines de validité (section 2.5.2) est polynomiale en la taille du système de contraintes définissant le polytope paramétré, et elle peut être calculée en un temps polynomial.*

Démonstration. Considérons les k hyperplans dans l'espace des paramètres formés par les enveloppes affines des intersections de dimension $(n-1)$ de paires de domaines de validité des sommets, où n est la dimension de l'espace de paramètres. Ces hyperplans subdivisent l'espace des paramètres en un nombre de domaines borné par un polynôme en k (lorsque la dimension est fixée) [50]. Puisqu'une partie de ces domaines forme les domaines de validité que nous cherchons, et puisque les k hyperplans correspondent aux projections des faces de dimension $(n-1)$ du polytope paramétré sur l'espace des paramètres, qui sont elles-mêmes (les faces) bornées en nombre par un polynôme en la taille des contraintes en entrée (pour une dimension fixée), le nombre de domaines de validité résultants, ainsi que leur taille globale sont polynomiaux en la taille des contraintes en entrée (pour une dimension fixée). □

Proposition 1. *Lorsque la dimension est fixée, la taille des quasi-polynômes représentant le nombre de points entiers d'un polytope paramétré est polynomiale en la taille des contraintes linéaires le définissant, et ils sont calculés en un temps polynomial.*

Démonstration. Puisque le nombre de sommets paramétrés (cônes supportants en l'occurrence) est polynomial [98], et puisque la décomposition unimodulaire de Barvinok de chaque cône supportant est polynomiale en la taille des contraintes en entrée (pour une dimension fixée) [12], le nombre de termes dans la fonction génératrice de chaque domaine de validité est polynomial. Et comme le nombre de domaines de validité est aussi polynomial (lemme 2), il en découle que le nombre global des termes de toutes les fonctions génératrices est polynomial en la taille des contraintes (pour une dimension fixée), le temps de calcul est donc polynomial. Enfin la taille des quasi-polynômes (résultant de l'évaluation des fonctions génératrices), où les nombres périodiques sont représentés par les parties entières inférieures/supérieures (*step-polynomials*) est aussi polynomial, puisque leurs degrés sont au maximum égaux à la dimension d qui est supposée fixée, et puisque la taille de chaque domaine de validité est polynomiale selon le lemme 2. □

3.3.4 Polytopes de dimension non pleine

Au début de cette section, nous avons supposé que le polytope P est de dimension pleine. Si ce n'est pas le cas, nous le transformons en un autre polytope qui a le même nombre de points entiers, mais qui est porté par un espace de dimension inférieure. La méthode suivante est due à S. Verdoolaege [142].

Si $P \in \mathbb{Q}^d$ est de dimension (géométrique) $d-l$, avec $l \geq 1$, alors sa description contient au moins une égalité $\langle \mathbf{a}, \mathbf{x} \rangle = \langle \mathbf{b}, \mathbf{p} \rangle + c$. Soit $\mathbf{a}' = \mathbf{a}/g$, où g est le plus grand commun diviseur (*pgcd*) des éléments de $\mathbf{a}$. Le vecteur $\mathbf{a}' = (a'_1, \cdots, a'_d)$ peut

être étendu à une matrice unimodulaire U [25], de la forme : $\begin{pmatrix} a'_1 & \cdots & a'_d \\ u_{21} & \cdots & u_{2d} \\ \vdots & \ddots & \\ u_{d1} & & u_{dd} \end{pmatrix}$. La construction de la matrice U est basée sur le fait que, étant donnée une matrice entière $(k \times k)$ U_k, avec $|det(U_k)| = g_k$, ou $g_k = pgcd(\alpha_1, \cdots, \alpha_k)$, une autre matrice entière $(k+1)\times(k+1)$ U_{k+1} peut être construite à partir de U_k, telle que $|det(U_{k+1})| = g_{k+1} = pgcd(\alpha_1, \cdots, \alpha_k, \alpha_{k+1})$ (voir [25] pour plus de détails sur cette procédure).

Soit $P' = UP$. Puisque U et U^{-1} sont unimodulaires ($|det(U)| = pgcd(a'_1, \cdots, a'_d) = 1$), il y a une bijection entre les points entiers de P et ceux de P', i.e., le nombre de points entiers dans les deux polytopes est le même ($\mathcal{E}(P') = \mathcal{E}(P)$). De plus, La première coordonnée x'_1 de P' est indépendante des autres coordonnées, puisque $gx'_1 = g\langle \mathbf{a}', \mathbf{x}\rangle = \langle \mathbf{b}, \mathbf{p}\rangle + c$ (par construction de la matrice U). P' est donc le produit de $P'' = \{\, (\langle \mathbf{b}, \mathbf{p}\rangle + c)/g \,\}$ et d'un certain polytope $P_1 \in \mathbb{Q}^{d-1}$. Le nombre de points entiers dans P est donc égal au produit des nombres de points entiers dans P'' et P_1, i.e., $\mathcal{E}(P) = \mathcal{E}(P') = \mathcal{E}(P'') \cdot \mathcal{E}(P_1)$. Le nombre de points entiers de P'' est égal à zéro ou un, selon les valeurs des paramètres. Il peut donc être représenté par un nombre périodique. Cette décomposition, dite factorisation [142], permet de minimiser le temps de calcul puisque le problème se réduit au dénombrement des points entiers d'un polytope de dimension inférieure. En répétant cette procédure l fois, i.e., autant de fois qu'il y a d'égalités, on obtient un polytope $P_l \in \mathbb{Q}^{d-l}$ de dimension pleine.

3.3.5 Quelques détails d'implémentation

Notre algorithme de comptage de points entiers dans des polytopes paramétrés (algorithme 1) a été implémenté, essentiellement par Sven Verdoolaege avec notre collaboration, dans la librairie `barvinok` [143], disponible sur :

`http://freshmeat.net/projects/barvinok/.`

La procédure de calcul de la décomposition unimodulaire de Barvinok est une réimplémentation indépendante de la procédure correspondante dans `LattE` [45], telle que nous l'avons présentée dans la section 3.2.1. Nous utilisons également l'implémentation de Shoup [131] de l'algorithme de réduction de base LLL, et `GMP` [62] pour l'arithmétique exacte en entiers longs. Contrairement à `LattE`, nous utilisons la librairie `Polylib` [95] pour réaliser les opérations polyédriques, puisque cela nous permet de réutiliser les procédures de calcul des sommets paramétrés et de la subdivision de l'espace de paramètres en domaines de validité [98].

3.3.6 Expériences

Nous avons utilisé l'implémentation `barvinok` [143] pour comparer les performances de notre méthode à celles de Clauss, Loechner et Wilde (méthode d'interpolation) implémentée dans la `Polylib` [95] version `5.21.0`. Nous avons choisi de faire la comparaison avec cette méthode puisque, à notre connaissance, elle est la seule implémentation qui traite des polytopes paramétrés quelconques comme la notre.

Le tableau 3.1 présente les résultat. Ces expériences ont été réalisées sur une machine Athlon MP 1500 avec 512 MiB de mémoire interne. La première colonne in-

	#DV	#DD	interpolation	notre méthode	
				tableaux	fractions
e16	4	0	16.076s	0.844s	0.702s
isnm	2	0	5.153s	0.038s	0.027s
g14	6	2	0.688s*	0.040s	0.040s
RD1	2	1	151.524s*	0.224s	0.068s
RD2	1	0	26.920s	3.207s	0.026s
CME	5	?	∞	∞	0.333s

TAB. 3.1 – Comparaison entre la méthode d'interpolation et notre méthode. Les temps marqués par * sont partiels, puisqu'ils sont appliqués seulement aux domaines non dégénérés.

dique le nombre de domaines de validité, la deuxième montre le nombre de domaines dégénérés de la méthode d'interpolation (notre méthode calcule toujours la solution, quelque soit la forme et la taille du domaine de validité), les troisième et quatrième colonnes montrent respectivement les temps de calcul de la méthode d'interpolation et de notre méthode en représentant les nombres périodiques par des tableaux, et la dernière colonne montre le temps de calcul de notre méthode en utilisant la représentation fractionnelle.

Les trois premiers polytopes peuvent être trouvés dans le jeu de test de la `Polylib`. Les deux premiers contiennent des nombres périodiques de tailles moyennes. Notre méthode est clairement plus rapide que l'interpolation et la représentation fractionnelle est légèrement plus rapide. Le troisième polytope résulte en deux domaines dégénérés (pour l'interpolation), et notre méthode est plus rapide, même si elle produit plus de quasi-polynômes que l'interpolation (qui ne peut pas produire des quasi-polynômes pour les domaines dégénérés). Les deux polytopes suivants apparaissent dans le contexte de la *distance de réutilisation* [24, 19]. Le temps d'exécution est très remarquablement amélioré par notre méthode. Il s'agit ici de quasi-polynômes avec des périodes relativement grandes, c'est pourquoi la représentation fractionnelle est encore plus rapide. Enfin, le dernier polytope est basé sur les équations de défauts de cache (*Cache Miss Equations*) [64]. Les deux méthodes ne peuvent pas calculer le résultat (mémoire saturée) lorsqu'on représente les nombres périodiques par des tableaux, puisque les périodes sont trop grandes. En utilisant la représentation fractionnelle, le résultat est par contre rapidement calculé.

Les expériences qui suivent ont été réalisées sur une machine Pentium4 de 2.66GHz de fréquence.

Dans le tableau 3.2, nous montrons une comparaison entre la méthode d'interpolation et notre méthode. Cette comparaison est effectuée sur en ensemble important de polytopes issus du calcul des distances de réutilisation [24]. La *distance de réutilisation* d'un accès mémoire à un élément a d'un tableau est le nombre d'éléments (différents) recherchés depuis l'accès précédent à l'élément a. Les éléments de tableaux accédés entre utilisation et réutilisation d'un même élément est d'abord donné par une formule de Presburger qui est ensuite transformée en une union disjointe de polytopes paramétrés en utilisant Omega [77]. La distance de réutilisation est ensuite calculée en comptant le nombre de points entiers dans les polytopes résultants. À noter qu'aucun des polytopes

programme	nombre de polytopes	méthode d'interpolation		notre méthode
		#domaines dégénérés	temps d'exécution	temps d'exécution
vpenta	6496	0	269.50s	165.84s
mxm	66	0	7.92s	1.98s
liv18	5296	6	248.68s*	135.43s
cholesky	76	0	6.12s	1.94s
jacobi	246	6	11.58s*	6.32s
gauss-jordan	308	0	19.01s	8.08s
tomcatv	8786	66	731.31s*	247.46s
total	21274	78	1294.12s*	567.05s

TAB. 3.2 – Les colonnes sont : nombre de polytopes construits par le calcul de la distance de réutilisation, nombre de domaines dégénérés en utilisant l'interpolation, et les temps d'exécution de la méthode d'interpolation et de notre méthode. Les temps marqués par * sont partiels, puisqu'ils sont appliqués seulement aux domaines non dégénérés.

de ces expériences n'a un comportement périodique. La taille de la solution est donc polynomiale pour les deux méthodes. Nous remarquons néanmoins que notre méthode est à-peu-près deux fois plus rapide que la méthode d'interpolation. De plus, sur les 21274 polytopes, 78 domaines dégénérés ont été constatés. Pour tous ces domaines, aucun polynôme n'a pu être calculé par la méthode d'interpolation. À noter enfin, que 99% des polytopes ont un seul domaine de validité. Les 1% restants ont entre 2 et 4 domaines de validité. Les polytopes sont de dimension 1 ou 2 et le nombre de paramètres est compris entre 2 et 7.

Afin de montrer l'impact des périodes sur taille de la solution et le temps de calcul, nous nous sommes basés sur le calcul du nombre de lignes de cache et le nombre de pages mémoire accédés par une référence dans une exécution donnée d'un nid de boucles [59]. Contrairement aux expériences précédentes, les polytopes de ces expériences ont de grandes périodes. Dans la figure 3.10, nous montrons le temps de calcul en fonction de la taille des périodes. Sur cette figure, on peut constater que le temps de calcul de la méthode d'interpolation augmente brusquement avec la taille de la période. Alors que le temps d'exécution de notre méthode est toujours inférieur à une seconde. La figure 3.11 quant à elle montre la taille de la solution en fonction des périodes. On peut en retenir que la taille de la solution produite par la méthode d'interpolation peut augmenter jusqu'à 32MiB. Alors que la taille de la solution de notre méthode (utilisant la représentation fractionnelle) est toujours inférieure à 9 KiB. 99% des polytopes de ces expériences ont un seul domaine de validité. Les 1% polytopes restants en ont deux. La dimension des polytopes est 1 ou 2 et le nombre de paramètres varie de 0 à 3.

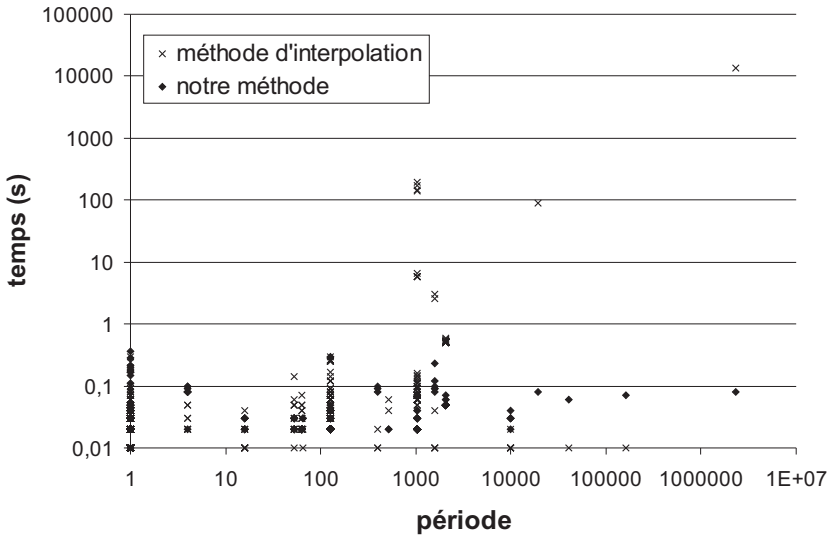


FIG. 3.10 – Temps d'exécution en fonction de la période maximale des quasi-polynômes calculés.

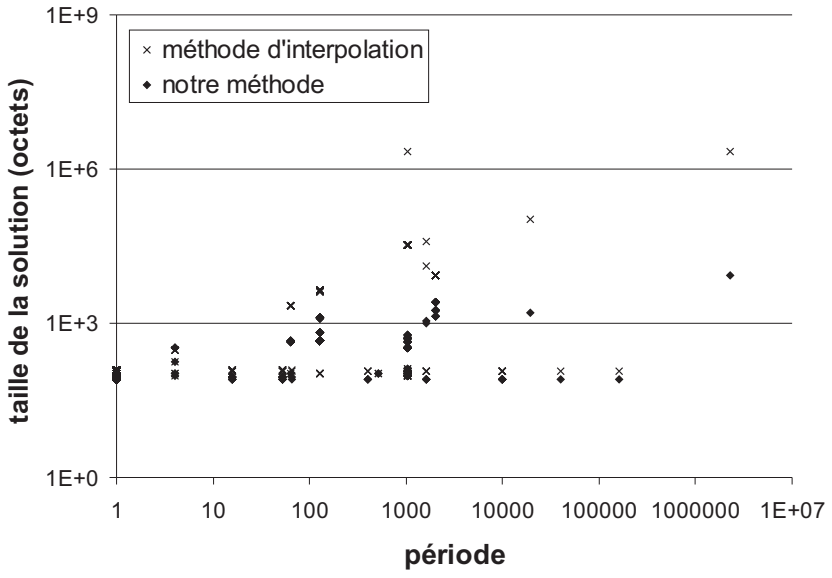


FIG. 3.11 – La taille de la solution en fonction de la période maximale des quasi-polynômes calculés.

Chapitre 4

Dénombrement des unions de $\mathbb{Z}$-polytopes paramétrés

Le dénombrement des unions de $\mathbb{Z}$-polytopes paramétrés est d'une grande importance. Il permet de répondre à de nombreux problèmes soulevés par le modèle polyédrique, tel que le dénombrement des ensembles issus de l'analyse de nids de boucles dont les pas sont supérieurs à un [70, 118, 111] ou l'analyse de nids de boucles multiples. Nous allons voir dans le chapitre 5 que l'image affine d'un $\mathbb{Z}$-polytope est donnée par une union de $\mathbb{Z}$-polytopes. Le calcul de ce type d'images est fondamentale, notamment pour modéliser les accès aux éléments de tableaux par des fonctions affines des indices de boucles. D'où l'importance de disposer d'un moyen permettant de calculer le nombre de points entiers dans de telles images. Parmi les applications ayant recours à ce genre de comptage, on peut citer le calcul de la quantité de mémoire touchée par les références à un tableau dans un morceau de programme. Lorsque les fonctions d'accès au tableau ne sont pas *surjectives*[1], une étape de calcul des *images affines* de $\mathbb{Z}$-polytopes est nécessaire pour pouvoir répondre à cette question. Le chapitre 5 est entièrement consacré à ce genre d'images. Par contre, lorsque les fonctions d'accès sont *surjectives*, la réponse à une telle question peut être donnée directement par l'algorithme de dénombrement des unions de $\mathbb{Z}$-polytopes, que nous proposons dans ce chapitre. Par opposition aux méthodes de calcul des unions disjointes, notre algorithme est basé sur le principe d'inclusion-exclusion. Cet algorithme a l'avantage de trouver la solution sans avoir besoin de calculer l'union disjointe de $\mathbb{Z}$-polytopes. Cette union peut en effet être exponentielle, et ce même pour un petit nombre de $\mathbb{Z}$-polytopes.

Dans ce qui suit, nous rappelons brièvement l'approche de dénombrement basée sur les unions disjointes. puis nous décrivons notre algorithme de dénombrement des unions quelconques de $\mathbb{Z}$-polytopes paramétrés. Mais avant cela, nous présentons d'abord quelques opérations sur les $\mathbb{Z}$-polytopes.

[1] Une fonction d'accès à un tableau est dite *surjective* si le nombre des éléments (différents) auxquels elle accède est égal au nombre des itérations du nid de boucles.

4.1 Opérations sur les $\mathbb{Z}$-polytopes

Plusieurs opérations peuvent être appliquées aux $\mathbb{Z}$-polytopes [118, 111]. Nous décrivons dans ce qui suit celles qui sont fondamentales pour les algorithmes de dénombrement des unions de $\mathbb{Z}$-polytopes

4.1.1 $\mathbb{Z}$-polytopes paramétrés

Un $\mathbb{Z}$-polytope paramétré est donné par l'intersection d'un polytope paramétré $P_{\mathbf{p}}$ et d'un lattice paramétré $L_{\mathbf{p}}$, avec

$$P_{\mathbf{p}} = \left\{ \mathbf{x} \in \mathbb{Q}^d, \mathbf{p} \in \mathbb{Z}^n \;\middle|\; \left(\begin{array}{c|c} A_{\mathbf{x}} & A_{\mathbf{p}} \end{array} \right) \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} + \mathbf{a} \geq 0 \right\},$$
$$L_{\mathbf{p}} = \left\{ \left(\begin{array}{c|c} B_{\mathbf{x}} & B_{\mathbf{p}} \end{array} \right) \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} + \mathbf{b} \;\middle|\; \mathbf{x} \in \mathbb{Z}^d,\ \mathbf{p} \in \mathbb{Z}^n \right\},$$

où $\mathbf{p} \in \mathbb{Z}^n$ est un vecteur de paramètres. En réalité, l'aspect paramétré d'un $\mathbb{Z}$-polytope n'intervient qu'au moment du comptage de ses points entiers (voir section 4.2.2). Toutes les opérations sur les $\mathbb{Z}$-polytopes que nous allons décrire ci-dessous sont effectuées en considérant les paramètres comme des variables régulières. Cela veut dire que l'on procède sur des $\mathbb{Z}$-polytopes homogènes, définis dans l'espace combiné des variables et des paramètres, de la forme $\mathcal{Z} = P \cap L$, avec

$$P = \left\{ \mathbf{x} \in \mathbb{Q}^{d+n} \mid A\mathbf{x} + \mathbf{a} \geq 0 \right\},$$
$$L = \left\{ B\mathbf{x} + \mathbf{b} \mid \mathbf{x} \in \mathbb{Z}^{d+n} \right\}.$$

Nous utilisons aussi la notation $L(B, \mathbf{b})$ pour désigner un lattice dont la matrice base est B et la pertie affine est $\mathbf{b}$.

4.1.2 Intersection de $\mathbb{Z}$-polytopes

L'intersection d'un ensemble de $\mathbb{Z}$-polytopes est polynomiale. Elles est calculée comme suit :

$$\bigcap^{i} \mathcal{Z}_i = \bigcap^{i} (P_i \cap L_i) = \left(\bigcap^{i} P_i \right) \cap \left(\bigcap^{i} L_i \right).$$

L'intersection d'un ensemble de $\mathbb{Z}$-polytopes se traduit donc par l'intersection d'un ensemble de polytopes et l'intersection d'un ensemble de lattices. L'intersection de polytopes est toujours un polytope *unique* pouvant être obtenu en concaténant simplement leurs contraintes respectives (et en supprimant les contraintes redondantes). L'intersection de lattices, quant à elle, est calculée comme suit :

Soit L_1 et L_2 deux lattices, avec $L_1 = \{A_1\mathbf{x}_1 + \mathbf{c}_1 \mid \mathbf{x}_1 \in \mathbb{Z}^d\}$ et $L_2 = \{A_2\mathbf{x}_2 + \mathbf{c}_2 \mid \mathbf{x}_2 \in \mathbb{Z}^d\}$. Les points entiers appartenant aux lattices L_1 et L_2 ci-dessus sont donnés respectivement par :

$$\forall \mathbf{y}_1 \in L_1, \exists \mathbf{x}_1 \in \mathbb{Z}^d, A_1 * \mathbf{x}_1 + \mathbf{c}_1 = \mathbf{y}_1,$$
$$\forall \mathbf{y}_2 \in L_2, \exists \mathbf{x}_2 \in \mathbb{Z}^d, A_2 * \mathbf{x}_2 + \mathbf{c}_2 = \mathbf{y}_2.$$

Si $\mathbf{z}$ est un point appartenant au lattice $L_3 = L_1 \cap L_2$, il doit exister $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{Z}^d$, tels que $\mathbf{z} = A_1 * \mathbf{x}_1 + \mathbf{c}_1 = A_2 * \mathbf{x}_2 + \mathbf{c}_2$. Ces deux égalités définissent un système d'équations diophantiennes

$$\begin{pmatrix} I_d & -A_1 & 0 \\ I_d & 0 & -A_2 \end{pmatrix} \begin{pmatrix} \mathbf{z} \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix}, \tag{4.1}$$

où I_d est la matrice identité $d \times d$.

La solution de ce système (si elle existe) est décrite par d variables libres, sous la forme :

$$\begin{pmatrix} \mathbf{z} \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} \mathbf{t} + \begin{pmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{pmatrix}, \ \mathbf{t} \in \mathbb{Z}^d,$$

où $\mathbf{t}$ est un d-vecteur de variables libres, $\mathbf{s}_1$ est un vecteur de d constantes et S_1 est une matrice $d \times d$. On obtient ainsi le lattice (unique) $L_3 = L_1 \cap L_2 = \{S_1\mathbf{t} + \mathbf{s}_1 \mid \mathbf{t} \in \mathbb{Z}^d\}$. Ceci reste valable pour un nombre quelconque de lattices (par associativité).

Le système d'équations (4.1) n'admet pas de solution entière si et seulement si l'intersection des deux lattices est vide. Dans ce cas, le $\mathbb{Z}$-polytope résultant est évidemment un ensemble vide. L'intersection d'un nombre quelconque de $\mathbb{Z}$-polytopes est donc toujours un $\mathbb{Z}$-polytope *unique*.

4.1.3 Union disjointe de $\mathbb{Z}$-polytopes

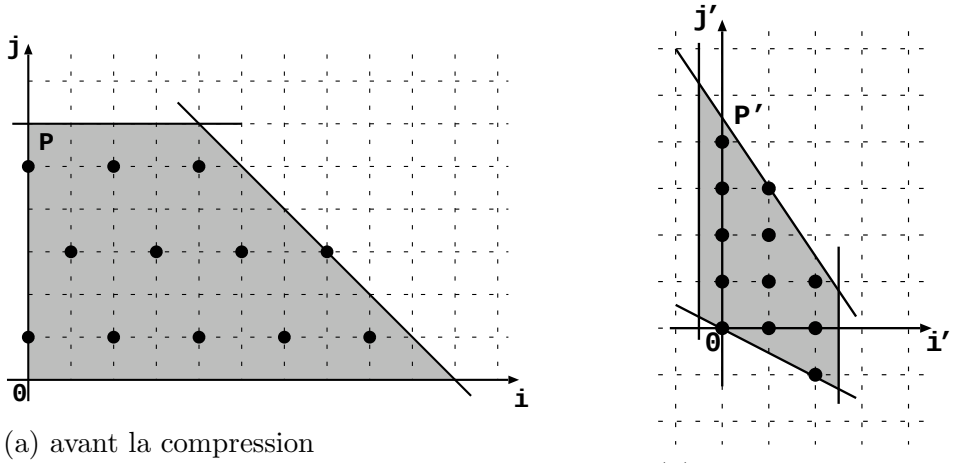
Contrairement à l'intersection, l'union de deux $\mathbb{Z}$-polytopes ne peut être calculée sous forme d'un $\mathbb{Z}$-polytope *unique*, sauf si un d'entre eux est un sous-ensemble de l'autre. Ce cas peut être facilement détecté et donc simplifié. Dans les autres cas, les $\mathbb{Z}$-polytopes sont simplement concaténés les uns aux autres [111]. Lorsque cette union n'est pas disjointe, i.e., au moins une paire de $\mathbb{Z}$-polytopes s'intersectent, il est souvent difficile de la manipuler et en particulier de compter les points entiers qu'elle comporte. D'où l'importance de calculer l'union disjointe de $\mathbb{Z}$-polytopes.

Le calcul de l'union disjointe de $\mathbb{Z}$-polytopes consiste à les séparer en un ensemble de $\mathbb{Z}$-polytopes dont les intersections deux à deux sont toutes vides. Ceci peut être effectué en deux étapes : d'abord l'union disjointe de polytopes est calculée, à travers des opérations d'intersection et de différence. Puis, les différents lattices à l'intérieur de chaque polytope résultant sont à leur tour séparés en lattices disjoints. C'est cette dernière étape qui est le plus souvent difficile à calculer. Nous reviendrons un peu plus en détail sur les unions disjointes dans la section 4.2.1.

4.1.4 Compression de $\mathbb{Z}$-polytopes

Définition 25. *Un* **$\mathbb{Z}$-polytope standard** *de dimension d est l'intersection d'un d-polytope avec le lattice standard $\mathbb{Z}^d$.*

La compression de $\mathbb{Z}$-polytopes est aussi très utile pour les algorithmes de comptage de points entiers dans des $\mathbb{Z}$-polytopes. Elles consiste à transformer un $\mathbb{Z}$-polytope



(a) avant la compression

(b) après la compression

FIG. 4.1 – Compression d'un $\mathbb{Z}$-polytope.

quelconque en un $\mathbb{Z}$-polytope standard, en supprimant les trous (à pas réguliers) qui existent entre ses points entiers. Le polytope résultant a la propriété de conserver le nombre de points entiers du $\mathbb{Z}$-polytope, mais non pas leurs coordonnées.

La compression d'un $\mathbb{Z}$-polytope $\mathcal{Z} = P \cap L$ est obtenue en calculant la préimage du polytope P par la matrice homogène définissant le lattice L. En pratique, ceci se traduit par le remplacement des variables $\mathbf{x}$ du polytope P par $M \begin{pmatrix} \mathbf{x'} \\ 1 \end{pmatrix}$, où M est la matrice homogène définissant le lattice L.

Exemple 19. Considérons le $\mathbb{Z}$-polytope $\mathcal{Z} = P \cap L$ de la figure 4.1a, avec

$$P = \{(i,j) \in \mathbb{Q}^2 \mid 0 \leq i \ \wedge \ 0 \leq j \leq 6 \ \wedge \ i + j \leq 10\},$$
$$L = \left\{ \begin{pmatrix} 1 & 2 \\ 2 & 8 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \;\middle|\; (i,j) \in \mathbb{Z}^2 \right\}.$$

Le lattice L peut être écrit sous la forme homogène : $L = \left\{ \begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \;\middle|\; (i,j) \in \mathbb{Z}^2 \right\}$. La compression du $\mathbb{Z}$-polytope peut ensuite être obtenue en remplaçant le vecteur $(i,j)^T$ par $\begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} i' \\ j' \\ 1 \end{pmatrix} = (i' + 2j', 2i' + 1)^T$ dans P. Ce qui donne le polytope de la figure 4.1b,

$$P' = \{(i',j') \in \mathbb{Q}^2 \mid 0 \leq i' + 2j' \ \wedge \ -1 \leq 2i' \leq 5 \ \wedge \ 3i' + 2j' \leq 9\},$$

ayant le même nombre de points entiers que le $\mathbb{Z}$-polytope original (figure 4.1a).

4.2 Dénombrement des unions de $\mathbb{Z}$-polytopes

Dans cette section, nous décrivons brièvement l'approche de dénombrement par l'union disjointe, et nous montrons ses inconvénients à travers un exemple. Puis nous

présentons notre algorithme basé sur le principe d'inclusion-exclusion et nous fournissons quelques résultats expérimentaux.

4.2.1 Dénombrement par l'union disjointe

Le nombre de points entiers d'un $\mathbb{Z}$-polytope (unique) peut être calculé directement en utilisant l'algorithme vu dans le chapitre 3 (algorithme 1). En effet, tout $\mathbb{Z}$-polytope peut être *compressé*, i.e., transformé en un $\mathbb{Z}$-polytope standard comportant le même nombre de points entiers. Malheureusement, cela ne peut être appliqué à des unions de $\mathbb{Z}$-polytopes, et ce pour la raison suivante :

Soient $\mathcal{Z}_1 = P_1 \cap L_1$ et $\mathcal{Z}_2 = P_2 \cap L_2$ deux $\mathbb{Z}$-polytopes. Le nombre de points entiers dans $\mathcal{Z}_1$ (resp. dans $\mathcal{Z}_2$) est égal au nombre de points entiers dans P_1' (resp. dans P_2'), où P_1' et P_2' sont respectivement les transformations de P_1 et P_2 par les matrices définissant les lattices L_1 et L_2. Cependant, le nombre de points entiers dans $\mathcal{Z}_1 \cup \mathcal{Z}_2$ n'est pas égal à celui de $P_1' \cup P_2'$, puisque la transformation appliquée à P_1 (resp. P_2) préserve seulement le nombre de points entiers de $\mathcal{Z}_1$ (resp. $\mathcal{Z}_2$) et non pas leurs coordonnées originales respectives. À noter que le résultat obtenu de cette façon est tout de même correct lorsque P_1 et P_2 sont disjoints ainsi que P_1' et P_2'.

```
for (i=1; i<=10; i+=2){
    for (j=3; j<=7; j+=2){
        ... = A[i][j] ...
    }
}
for (i=3; i<=12; i+=3){
    for (j=1; j<=6; j+=2){
        ... = A[i][j] ...
    }
}
```

FIG. 4.2 – Tableau accédé par deux nids de boucles

Exemple 20. Supposons que nous nous intéressons au nombre d'éléments (différents) du tableau A accédés par le morceau de programme de la figure 4.2. Dans ce morceau de programme, les fonctions de référence au tableau A sont simplement les indices de boucles. Ceci facilite le calcul du nombre d'éléments accédés. En effet, ce nombre est égal simplement au nombre des itérations (différentes) des deux nids de boucles. Des fonctions de référence plus générales seront étudiées dans le chapitre 5.

Les itérations des deux nids de boucles correspondent respectivement aux deux $\mathbb{Z}$-polytopes $\mathcal{Z}_1 = P_1 \cap L_1$ et $\mathcal{Z}_2 = P_2 \cap L_2$, avec

$$P_1 = \{(i,j) \in \mathbb{Q}^2 \mid 1 \leq i \leq 10 \ \wedge \ 3 \leq j \leq 7\},$$

$$L_1 = \left\{ \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \;\middle|\; (i,j) \in \mathbb{Z}^2 \right\},$$

$$P_2 = \{(i,j) \in \mathbb{Q}^2 \mid 3 \leq i \leq 12 \ \wedge \ 1 \leq j \leq 6\},$$

$$L_2 = \left\{ \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \;\middle|\; (i,j) \in \mathbb{Z}^2 \right\}.$$

Les deux $\mathbb{Z}$-polytopes sont montrés par la figure 4.3, où les cercles, les carrés et les diamants appartiennent respectivement à $\mathcal{Z}_1$, $\mathcal{Z}_2$ et $\mathcal{Z}_1 \cap \mathcal{Z}_2$.

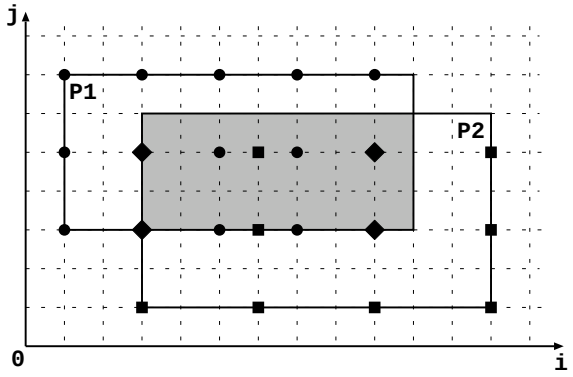


FIG. 4.3 – Union de deux $\mathbb{Z}$-polytopes.

Le nombre des itérations différentes des deux nids de boucles (nombre des éléments du tableau A qui sont accédés) est égal au nombre des points entiers dans *l'union* de $\mathcal{Z}_1$ et $\mathcal{Z}_2$. En substituant $i = 2i'_1 + 1$ et $j = 2j' + 1$ dans P_1 (resp. $i = 3i'$ et $j = 2j' + 1$ dans P_2), nous obtenons P'_1 (resp. P'_2) dans lesquels il y a respectivement 15 et 12 points entiers :

$$P'_1 = \{(i',j') \in \mathbb{Z}^2 \mid 0 \leq 2i' \leq 9 \ \wedge \ 1 \leq j' \leq 3\},$$

$$P'_2 = \{(i',j') \in \mathbb{Z}^2 \mid 1 \leq i' \leq 4 \ \wedge \ 0 \leq 2j' \leq 5\}.$$

On peut vérifier que le nombre de points entiers dans $P'_1 \cup P'_2$ est 19. Ce nombre ne correspond évidement pas au nombre de points dans $\mathcal{Z}_1 \cup \mathcal{Z}_2$. Sur la figure 4.3, on voit que l'union des deux $\mathbb{Z}$-polytopes comporte 23 points entiers.

Au meilleur de nos connaissances, toutes les méthodes antérieures (e.g., [104, 156]) s'intéressant au comptage de points entiers dans des unions de $\mathbb{Z}$-polytopes sont basées sur la décomposition de ces unions en des unions *disjointes* de $\mathbb{Z}$-polytopes. Ceci revient à séparer les lattices définissant les $\mathbb{Z}$-polytopes en une union disjointe de lattices. Or cette séparation est souvent très difficile à réaliser et peut être exponentielle au pire cas (en fonction des vecteurs générateurs des lattices), et ce même pour un nombre fixé de lattices. A titre d'exemple, l'union des deux lattices de la figure 4.3 (L_1 et L_2) résulte en quatre lattices disjoints :

$$L'_1 = \left\{ \begin{pmatrix} 6 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \;\middle|\; (i', j') \in \mathbb{Z}^2 \right\},$$

$$L'_2 = \left\{ \begin{pmatrix} 6 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \;\middle|\; (i', j') \in \mathbb{Z}^2 \right\},$$

$$L'_3 = \left\{ \begin{pmatrix} 6 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} 3 \\ 1 \end{pmatrix} \;\middle|\; (i', j') \in \mathbb{Z}^2 \right\},$$

$$L'_4 = \left\{ \begin{pmatrix} 6 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} 5 \\ 1 \end{pmatrix} \;\middle|\; (i', j') \in \mathbb{Z}^2 \right\}.$$

À noter que la composante 6 du premier vecteur générateur des lattices ci-dessus est égale au plus petit commun multiple (*ppcm*) de 2 et 3. D'une manière générale, le nombre de lattices dans l'union disjointe de deux lattices est proportionnel au *ppcm* des composantes de leurs vecteurs générateurs.

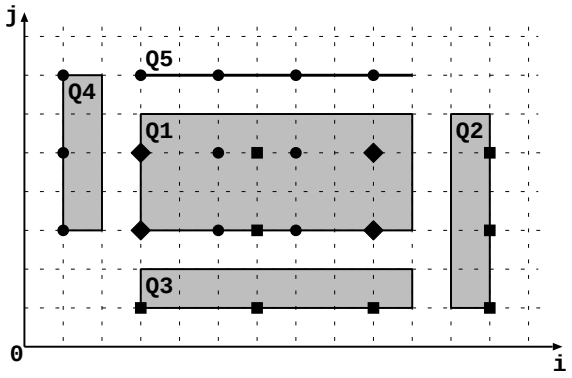


FIG. 4.4 – Union disjointe de $\mathbb{Z}$-polytopes.

En utilisant l'approche *union disjointe*, le nombre de points entiers dans $\mathcal{Z}_1 \cup \mathcal{Z}_2$ peut être calculé en trois étapes :

- calculer l'union disjointe des deux polytopes P_1 et P_2, ce qui résulte en cinq nouveaux polytopes $Q_1, \cdots, Q_5$ (voir la figure 4.4).
- calculer l'union disjointe des deux lattices L_1 et L_2 ($L'_1, \cdots, L'_4$ ci-dessus) à l'intérieur du polytope $Q_1 = P_1 \cap P_2$.
- transformer chaque $\mathbb{Z}$-polytope résultant ($\mathcal{Z}_1 = Q_1 \cap L'_1$, $\mathcal{Z}_2 = Q_1 \cap L'_2$, $\mathcal{Z}_3 = Q_1 \cap L'_3$, $\mathcal{Z}_4 = Q_1 \cap L'_4$, $\mathcal{Z}_5 = Q_2 \cap L_2$, $\mathcal{Z}_6 = Q_3 \cap L_2$, $\mathcal{Z}_7 = Q_4 \cap L_1$ et $\mathcal{Z}_8 = Q_5 \cap L_1$) en un $\mathbb{Z}$-polytope standard, compter ses points entiers (algorithme 1) et faire la somme des résultats.

Nous remarquons ici que le dénombrement de $\mathcal{Z}_1 \cup \mathcal{Z}_2$ se fait en comptant les points entiers dans *huit* polytopes. Dans la section suivante nous montrerons que notre algorithme le fait en dénombrant seulement *trois* polytopes.

4.2.2 Dénombrement par inclusion-exclusion

Dans cette section, nous décrivons notre algorithme de comptage de points entiers dans des unions quelconques de $\mathbb{Z}$-polytopes paramétrés de la forme : $\bigcup_i (Z_i = P_i \cap L_i)$, avec

$$P_i = \left\{ \mathbf{x} \in \mathbb{Q}^d, \mathbf{p} \in \mathbb{Z}^n \;\middle|\; \left(\begin{array}{c|c} A_{\mathbf{x}} & A_{\mathbf{p}} \end{array} \right) \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} + \mathbf{a} \geq 0 \right\},$$

$$L_i = \left\{ \left(\begin{array}{c|c} B_{\mathbf{x}} & B_{\mathbf{p}} \end{array} \right) \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} + \mathbf{b} \;\middle|\; \mathbf{x} \in \mathbb{Z}^d,\ \mathbf{p} \in \mathbb{Z}^n \right\},$$

où $\mathbf{p} \in \mathbb{Z}^n$ est un vecteur de paramètres.

Principe d'inclusion-exclusion

Contrairement aux algorithmes vus dans la section 4.2.1, notre algorithme est basé sur le principe d'inclusion-exclusion. Il s'agit de calculer le nombre de points dans des unions de $\mathbb{Z}$-polytopes dans lesquels les points d'intersection sont comptés plusieurs fois. Ensuite, le sur-comptage est corrigé en y retranchant le nombre de points dans des intersections de $\mathbb{Z}$-polytopes qui ont participé au sur-comptage. La formule générale du principe d'inclusion-exclusion est la suivante :

$$\left| \bigcup_{1 \leq i \leq p} S_j \right| = \sum_{1 \leq i_1 \leq p} |S_{i_1}| - \sum_{1 \leq i_1 \leq i_2 \leq p} |S_{i_1} \cap S_{i_2}| + \sum_{1 \leq i_1 \leq i_2 \leq i_3 \leq p} |S_{i_1} \cap S_{i_2} \cap S_{i_3}| - \cdots + (-1)^{p-1} |S_1 \cap S_2 \cdots \cap S_p|,$$

où $|S|$ est la cardinalité de l'ensemble S.

Comme le montre cette formule d'inclusion-exclusion, la seule opérations polyédrique dont on a besoin est *l'intersection*. Ceci constitue un avantage par rapport à l'approche *union disjointe* puisque l'intersection d'un nombre quelconque de $\mathbb{Z}$-polytopes résulte toujours en un *seul* $\mathbb{Z}$-polytope, quels que soient les vecteurs générateurs de leurs lattices (voir la section 4.1.2).

Dans notre algorithme (algorithme 2), nous commençons par appliquer le principe d'inclusion-exclusion ci-dessus à l'union de $\mathbb{Z}$-polytopes. Pour ce faire, nous procédons sur des listes de $\mathbb{Z}$-polytopes signés (S et S'). D'abord, la liste S est initialisée par une liste vide. Ensuite, les éléments de F (union de $\mathbb{Z}$-polytopes à dénombrer) sont retirés un par un et intersectés avec tous les éléments de S. Les intersections non vides sont insérées dans la liste S' avec des signes opposés à l'élément en cours de S. Enfin, la liste S' ainsi que l'élément retiré de F sont insérés dans la liste S.

Compression et normalisation de $\mathbb{Z}$-polytopes

Après application du principe d'inclusion-exclusion, le nombre de points entiers dans chaque $\mathbb{Z}$-polytope résultant $\mathcal{Y}_i = P_i \cap L_i$ est calculé comme suit :

Nous transformons d'abord la matrice $\left(\begin{array}{c|c} B_{\mathbf{x}} & B_{\mathbf{p}} \end{array} \right)$ générant le lattice $L_i = \left\{ \left(\begin{array}{c|c} B_{\mathbf{x}} & B_{\mathbf{p}} \end{array} \right) \begin{pmatrix} \mathbf{x} \\ \mathbf{p} \end{pmatrix} + \begin{pmatrix} \mathbf{b_x} \\ \mathbf{b_p} \end{pmatrix} \;\middle|\; \mathbf{x} \in \mathbb{Z}^d, \mathbf{p} \in \mathbb{Z}^n \right\}$ en une nouvelle matrice dans laquelle toutes les lignes qui transforment les paramètres sont *indépendantes* des variables.

Algorithme 2 Dénombrement d'une union de $\mathbb{Z}$-polytopes paramétrés

Entrée :
F : Union de $\mathbb{Z}$-polytopes paramétrés
Sortie :
L : Liste de (VD, EP) *// (Domaine de Validité, Polynôme d'Ehrhart)*
Variables :
S, S' : Liste de $(s, \mathcal{Y})$ *// (Signe, $\mathbb{Z}$-Polytope)*
E, L' : Liste de (VD, EP) ; D_1, D_2 : Domaines de Validité
P : Polytope ; $\mathcal{Z}, I$: $\mathbb{Z}$-Polytopes

// Principe d'inclusion-exclusion
S = Vide
Pour Tout $\mathcal{Z}$ dans F
  $S' = S$
  Pour Tout (s,$\mathcal{Y}$) dans S
    $I = \mathcal{Z} \cap \mathcal{Y}$
    Si Non Vide (I)
      $S' = S' + (-1 \times s, I)$
  Fin Pour
  $S = S' + (+1, \mathcal{Z})$
Fin Pour

// Dénombrement de S, seules les paires ayant des VD non vides sont ajoutées à L
L = Vide () ; D_1 = Vide
Pour Tout $(s, \mathcal{Y})$ dans S
  P = CompresserEtNormaliser $(\mathcal{Y})$
  $E = s\times$ Dénombrer (P) *// Algorithme 1*
  Si Non Vide (L)
    $D_1 = \bigcup_i VD_i$ dans L ; $D_2 = \bigcup_j VD_j$ dans E
  Fin Si
  L' = Vide ()
  Pour Tout (VD_i, EP_i) dans L
    Pour Tout (VD_j, EP_j) dans E
      $L' = L' + (VD_i(L) \cap VD_j(E), EP_i(L) + EP_j(E))$
    Fin Pour
    $L' = L' + (VD_i(L) - D_2, EP_i(L))$
  Fin Pour
  Pour Tout (VD_j, EP_j) dans E
    $L' = L' + (VD_j(E) - D_1, EP_j(E))$
  Fin Pour
  $L = L'$
Fin Pour

Ceci est nécessaire pour garder l'espace de données (variables) compressé lorsqu'on réécrit le polytope transformé en fonction de ses paramètres originaux. Noter que nous revenons aux paramètres originaux avant le dénombrement, puisqu'il est difficile de réécrire les quasi-polynômes résultants en fonction des paramètres initiaux.

La matrice recherchée est de la forme : $M = \left(\begin{array}{c|c} B_{\mathbf{xx}} & B_{\mathbf{xp}} \\ \hline 0 & B_{\mathbf{pp}} \end{array}\right)$. Les vecteurs colonnes de cette matrice génère les mêmes points entiers que ceux de la matrice originale $\left(\begin{array}{c|c} B_{\mathbf{x}} & B_{\mathbf{p}} \end{array}\right)$, puisqu'elle est calculée à partir de sa forme normale de Hermite, respectant le corollaire ci-dessous.

Définition 26. *Une matrice A de rang ligne plein* (full row rank) *est dite en* **forme normale de Hermite** *si elle est de la forme $(B\ 0)$, où B est une matrice inversible, triangulaire inférieure, non négative, et dont chaque ligne a une entrée maximale située sur sa diagonale principale.*

À noter que la forme normale de Hermite d'une matrice *carrée* inversible ne possède pas de colonnes de zéros, i.e., uniquement la partie B de la matrice $[B\ 0]$ ci-dessus. Les matrices définissant des lattices de points entiers rentrent dans cette catégorie de matrices.

Théorème 8 ([126])**.** *Toute matrice de rang ligne plein peut être ramenée à la forme normale de Hermite par une série d'opérations élémentaires sur ses colonnes. Les opérations élémentaires sont l'échange de deux colonnes, la multiplication d'une colonne par -1 et l'ajout d'un multiple entier d'une colonne à une autre.*

Corollaire 1 ([126])**.** *Soit A et A' deux matrices inversibles, les propriétés suivantes sont équivalentes :*

- *les colonnes de A et celles de A' génèrent le même lattice de points entiers.*
- *A' est obtenue à partir de A en appliquant une série d'opérations élémentaires sur ses colonnes.*

D'après le théorème et le corollaire précédent, il découle que la matrice représentant la partie linéaire d'un lattice (ses vecteurs générateurs) peut être remplacée par sa forme normale de Hermite (matrice triangulaire inférieure) sans affecter le lattice en question. Le fait qu'un lattice soit défini par une matrice triangulaire inférieure signifie que toute variable n'est définie qu'en fonction des variables qui la précèdent. Ainsi, si on met les paramètres en premier, on s'assure qu'aucun des paramètres ne soit défini en fonction des variables. Après le calcul de la nouvelle partie linéaire du lattice, on le réorganise de façon à avoir les paramètres en dernier (pour des considérations d'implémentation). C'est la raison pour laquelle la matrice $M = \left(\begin{array}{c|c} B_{\mathbf{xx}} & B_{\mathbf{xp}} \\ \hline 0 & B_{\mathbf{pp}} \end{array}\right)$ n'est pas tout à fait triangulaire inférieure.

Une fois que la matrice M est calculée comme expliqué ci-dessus, nous appliquons la transformation affine $\left(\begin{array}{c|c} M & \begin{pmatrix} \mathbf{b_x} \\ \mathbf{b_p} \end{pmatrix} \end{array}\right)$ à P_i pour obtenir un $\mathbb{Z}$-polytope standard P'_i (P'_i est la compression du $\mathbb{Z}$-polytope $\mathcal{Y}_i$). Ensuite, P'_i est réécrit en fonction des paramètres originaux en utilisant la sous-matrice $(B_{\mathbf{pp}}|\ \mathbf{b_p})$.

Note 6. *Dans le cas* non paramétré*, aucune transformation (normalisation) de lattice n'est nécessaire. Chaque $\mathbb{Z}$-polytope résultant de la procédure d'inclusion-exclusion $\mathcal{Y}_i = P_i \cap L_i$ est transformé directement en un $\mathbb{Z}$-polytope standard en calculant la préimage du polytope P_i par le lattice L_i.*

Dénombrement des unions de polytopes paramétrés

Après la normalisation des lattices et la compression des $\mathbb{Z}$-polytopes, nous utilisons l'algorithme (algorithme 1) pour calculer les quasi-polynômes d'Ehrhart représentant le nombre de points entiers dans chaque polytope résultant.

Noter que lorsque la sous-matrice $B_{\mathbf{pp}}$ n'est pas égale à la matrice identité, le polytope résultant de la transformation n'est valide que pour les valeurs de paramètres générées par le lattice $L_{\mathbf{p}}$ dont la base est $B_{\mathbf{pp}}$ et la partie affine est $\mathbf{b_p}$. Dans ce cas, le quasi-polynôme d'Ehrhart résultant est à multiplier par un nombre périodique égal à 1 lorsque les valeurs des paramètres sont valides (i.e., lorsqu'elle appartiennent au lattice $L_{\mathbf{p}}$) et 0 si non.

Enfin, le résultat global est obtenu en calculant les sommes/différences des différents polynômes d'Ehrhart. Comme nous l'avons présenté dans le chapitre 3, le nombre de points entiers dans un polytope paramétré (unique) peut être donné par un certain nombre de quasi-polynômes d'Ehrhart, chacun n'étant valide que sur un domaine de validité particulier. Donc, il est nécessaire de prendre en considération ces domaines de validité pour calculer le nombre total de points entiers dans une union de polytopes.

Considérons le cas simple de deux polytopes paramétrés disjoints P_1 et P_2 ayant respectivement pour nombre de points entiers $(D_1, \mathcal{E}_1)$ et $(D_2, \mathcal{E}_2)$, où $(D_i, \mathcal{E}_i)$ signifie que le nombre de points du polytope P_i est donné par un quasi-polynôme unique $\mathcal{E}_i$ lorsque les valeurs des paramètres appartiennent au domaine D_i. Dans ce cas, le nombre de points entiers dans l'union $P_1 \cup P_2$ est donné par trois couples :

$$\mathcal{E}(P_1 \cup P_2) = \{(D_1 \setminus D_2, \mathcal{E}_1),\ (D_2 \setminus D_1, \mathcal{E}_2),\ (D_1 \cap D_2, \mathcal{E}_1 + \mathcal{E}_2)\},$$

où seuls les couples dont les domaines de validité ne sont pas vides sont pris en compte.

La dernière partie de l'algorithme 2 résume le cas général de plusieurs polytopes ayant plusieurs domaines de validité. L'idée est la suivante :

Au début, la Liste L est initialisée par le résultat du dénombrement du premier polytope. Ensuite, à chaque fois qu'un polytope est dénombré, i.e., à chaque fois qu'une nouvelle liste E de couples (domaine de validité, polynôme d'Ehrhart) est calculée, la liste L est remplacée par une combinaison L' de la nouvelle liste E et L elle même.

La liste L' est remplie par trois sortes de couples, où seuls les couples ayant des domaines de validité non vides sont pris en compte :

- Les couples dont les domaines de validité correspondent aux intersections deux à deux des domaines de L et ceux de E. Les polynômes de ces couples sont donnés par les sommes deux à deux des polynômes de L et ceux de E.
- Les couples dont les domaines de validité sont les domaines de E moins ceux de L. Les polynômes correspondant à ces couples sont ceux de E.
- Les couples dont les domaines de validité sont les domaines de L moins ceux de E. Les polynômes correspondant à ces derniers couples sont ceux de L

Exemple 21. Considérons une version paramétrée des $\mathbb{Z}$-polytopes $\mathcal{Z}_1 = P_1 \cap L_1$ et $\mathcal{Z}_2 = P_2 \cap L_2$ de l'exemple 20, avec

$$P_1 = \{(i,j,N) \in \mathbb{Q}^3 \mid 1 \leq i \leq N+5 \ \wedge \ 3 \leq j \leq 7\},$$

$$L_1 = \left\{ \left(\begin{array}{cc|c} 2 & 0 & 3 \\ 0 & 2 & 0 \\ \hline 0 & 0 & 1 \end{array}\right) \left(\begin{array}{c} i \\ j \\ N \end{array}\right) + \left(\begin{array}{c} 1 \\ 1 \\ 0 \end{array}\right) \;\middle|\; (i,j,N) \in \mathbb{Z}^3 \right\},$$

$$P_2 = \{(i,j,N) \in \mathbb{Q}^3 \mid 3 \leq i \leq 2N+7 \ \wedge \ 1 \leq j \leq 6\},$$

$$L_2 = \left\{ \left(\begin{array}{cc|c} 3 & 0 & 0 \\ 0 & 2 & 0 \\ \hline 0 & 0 & 3 \end{array}\right) \left(\begin{array}{c} i \\ j \\ N \end{array}\right) + \left(\begin{array}{c} 2 \\ 1 \\ 0 \end{array}\right) \;\middle|\; (i,j,N) \in \mathbb{Z}^3 \right\},$$

où $N \in \mathbb{Z}$ est un paramètre. Le nombre de points entiers dans l'union $\mathcal{Z}_1 \cup \mathcal{Z}_2$ est donné par :

$$\mathcal{E}(\mathcal{Z}_1 \cup \mathcal{Z}_2) = \mathcal{E}(\mathcal{Z}_1) + \mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_1 \cap \mathcal{Z}_2).$$

Mettons $(\mathcal{Z}_1 \cap \mathcal{Z}_2) = \mathcal{Z}_3 = (P_3 \cap L_3)$, avec

$$P_3 = P_1 \cap P_2 = \{(i,j) \in \mathbb{Q}^2 \mid 3 \leq i \leq N+5 \ \wedge \ 3 \leq j \leq 6\},$$

$$L_3 = L_1 \cap L_2 = \left\{ \left(\begin{array}{cc|c} 3 & 0 & 0 \\ 0 & 2 & 0 \\ \hline 3 & 0 & 6 \end{array}\right) \left(\begin{array}{c} i \\ j \\ N \end{array}\right) + \left(\begin{array}{c} 2 \\ 1 \\ 3 \end{array}\right) \;\middle|\; (i,j,N) \in \mathbb{Z}^3 \right\}.$$

Le nombre de points entiers dans $\mathcal{Z}_3$ est calculé comme suit :

D'abord, la base $\left(\begin{array}{cc|c} 3 & 0 & 0 \\ 0 & 2 & 0 \\ \hline 3 & 0 & 6 \end{array}\right)$ du lattice L_3 est transformée en une nouvelle base $\left(\begin{array}{cc|c} 6 & 0 & 3 \\ 0 & 2 & 0 \\ \hline 0 & 0 & 3 \end{array}\right)$ dans laquelle les coefficients des variables dans la troisième ligne (la ligne transformant le paramètre N) sont tous égaux à zéro. La nouvelle base génère les même points entiers que le lattice original, puisqu'elle est calculée à partir de sa forme normale de Hermite comme expliqué ci-dessus. Le $\mathbb{Z}$-polytope $\mathcal{Z}_3$ est ensuite compressé, i.e., transformé en un $\mathbb{Z}$-polytope standard P, donné par la préimage de P_3 par la matrice homogène $\begin{pmatrix} 6 & 0 & 3 & 2 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 3 & 3 \end{pmatrix}$:

$$P = \{(i',j',N') \in \mathbb{Q}^3 \mid -3N'+1 \leq 6i' \leq 6 \ \wedge \ 2 \leq 2j' \leq 5\}.$$

Avant de compter les points entiers de P, nous avons besoin de l'écrire en fonction du paramètre original N. Pour ce faire, il suffit de calculer son image par la matrice $M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 3 \end{pmatrix}$, nous obtenons :

$$P' = \{(i',j',N) \in \mathbb{Q}^3 \mid -N+4 \leq 6i' \leq 6 \ \wedge \ 2 \leq 2j' \leq 5\}.$$

Le nombre de points entiers dans P' est donné par l'algorithme (algorithme 1) sous la forme :

$$\mathcal{E}(P') = \frac{1}{3}N + \left[2, \frac{5}{3}, \frac{4}{3}, 1, \frac{8}{3}, \frac{7}{3}\right]_N, \text{ avec } N \geq -2.$$

La troisième ligne de la matrice M dit que $N = 3N' + 3$, avec $N' \in \mathbb{Z}$. En d'autres termes, N doit être un multiple de 3 ($N \bmod 3 = 0$). C'est pourquoi le quasi-polynôme d'Ehrhart résultant est multiplié par le nombre périodique $[1, 0, 0]_N$ qui est égal à 1 lorsque $N \bmod 3 = 0$ et 0 sinon. Le résultat est

$$\mathcal{E}(Z_3) = \left[\frac{1}{3}, 0, 0\right]_N N + [2, 0, 0, 1, 0, 0]_N, \text{ avec } N \geq -2.$$

En procédant d'une façon similaire, les nombres de points entiers dans $\mathcal{Z}_1$ et $\mathcal{Z}_2$ peuvent être obtenus sous la forme :

$$\mathcal{E}(\mathcal{Z}_1) = \frac{3}{2}N + \left[9, \frac{15}{2}\right]_N, \text{ avec } N \geq -4,$$

$$\mathcal{E}(\mathcal{Z}_2) = [2, 0, 0]_N N + [3, 0, 0]_N, \text{ avec } N \geq -2.$$

Enfin, le nombre de points entiers dans $\mathcal{Z}_1 \cup \mathcal{Z}_2$ est donné par

$$\mathcal{E}(\mathcal{Z}_1 \cup \mathcal{Z}_2) = \mathcal{E}(\mathcal{Z}_1) + \mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_3).$$

Puisque $\mathcal{E}(\mathcal{Z}_2)$ et $\mathcal{E}(\mathcal{Z}_3)$ ont le même domaine de validité $N \geq -2$, on a directement

$$\mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_3) = \left[\frac{5}{3}, 0, 0\right]_N N + [1, 0, 0, 2, 0, 0]_N, \text{ avec } N \geq -2.$$

Le domaine de validité de $\mathcal{E}(\mathcal{Z}_1)$ ($N \geq -2$) et celui de $(\mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_3))$ ($N \geq -4$) ne sont pas les mêmes. Ce qui fait que pour calculer $\mathcal{E}(Z_1 \cup Z_2) = (\mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_3)) + \mathcal{E}(\mathcal{Z}_1)$, on doit séparer ces domaines comme suit :

$\mathcal{E}(Z_1 \cup Z_2)$ est égal à $(\mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_3)) + \mathcal{E}(\mathcal{Z}_1)$ lorsque $N \in (N \geq -2) \cap (N \geq -4)$, $\mathcal{E}(\mathcal{Z}_2) - \mathcal{E}(\mathcal{Z}_3)$ lorsque $N \in (N \geq -2) \setminus (N \geq -4)$, et $\mathcal{E}(\mathcal{Z}_1)$ lorsque $N \in (N \geq -4) \setminus (N \geq -2)$. Sachant que $(N \geq -2) \setminus (N \geq -4) = \emptyset$, le résultat final est :

$$\mathcal{E}(Z_1 \cup Z_2) = \begin{cases} \left[\frac{19}{6}, \frac{3}{2}, \frac{3}{2}\right]_N N + \left[10, \frac{15}{2}, 9, \frac{19}{2}, 9, \frac{15}{2}\right]_N & \text{si } N \geq -2, \\ \frac{3}{2}N + \left[9, \frac{15}{2}\right]_N & \text{si } -4 \leq N \leq -3, \\ 0 & \text{sinon.} \end{cases}$$

Noter que pour faire la somme de deux nombres périodiques, il faut qu'il soient de la même période. Si ce n'est pas le cas, on doit les transformer en deux nouveaux nombres périodiques dont la période est égale au plus commun multiple *ppcm* des périodes des nombres originaux. Par exemple

$$\begin{aligned} [1, 0, 0, 2, 0, 0]_N + \left[9, \frac{15}{2}\right]_N &= [1, 0, 0, 2, 0, 0]_N + \left[9, \frac{15}{2}, 9, \frac{15}{2}, 9, \frac{15}{2}\right]_N \\ &= \left[10, \frac{15}{2}, 9, \frac{19}{2}, 9, \frac{15}{2}\right]_N. \end{aligned}$$

Complexité de l'algorithme

La complexité de l'algorithme 2 dépend principalement de la complexité des opérations sur les $\mathbb{Z}$-polytopes, de la complexité de l'algorithme de comptage de points entiers dans un polytope paramétré et du nombre de $\mathbb{Z}$-polytopes de l'ensemble S, résultant de la procédure d'inclusion-exclusion. La seule opération significative sur les $\mathbb{Z}$-polytopes utilisée dans cet algorithme est l'intersection, qui est polynomiale puisque l'intersection de deux polytopes peut être obtenue en concaténant simplement leurs contraintes, et l'intersection de deux lattices se traduit par la résolution d'un système d'égalités linéaire [111], qui est polynomiale en la taille du système [126] (voir section 4.1.2). Le comptage de points entiers dans un polytope paramétré (algorithme 1) est aussi polynomial (pour une dimension fixée) en la taille des contraintes définissant le polytope (voir la section 3.3.3). Enfin, les $\mathbb{Z}$-polytopes résultants dans l'ensemble S sont donnés par le principe d'inclusion-exclusion. Ce principe est utilisé dans plusieurs autres contextes, comme par exemple pour le calcul des unions de fractions génératrices [13]. Lorsque le nombre d'ensembles en entrée est fixé, le principe d'inclusion-exclusion résulte en un nombre polynomial d'ensembles [13]. L'algorithme global est donc polynomial en la taille des contraintes définissant les $\mathbb{Z}$-polytopes en entrée (pour une dimension et un nombre de $\mathbb{Z}$-polytopes fixés).

Note 7. *Lorsque le nombre d'ensembles (disons n) est grand, l'application du principe d'inclusion-exclusion résulte théoriquement en un nombre exponentiel (2^n-1) d'ensembles. Ce nombre est atteint seulement si toutes les intersections de la formule d'inclusion-exclusion ne sont pas vides. En pratique, plus il y a d'ensembles, plus il y a d'intersections vides. Le comportement exponentiel n'apparaît donc que rarement.*

Expériences

Dans cette section nous présentons quelques résultats expérimentaux illustrant les temps de calcul de nombres de points entiers dans des unions de $\mathbb{Z}$-polytopes. En réalité, il ne s'agit que de donner une idée des temps de calcul pour différents ensembles de $\mathbb{Z}$-polytopes, puisqu'il est difficile de trouver un ensemble d'exemples représentatif. Ceci est dû au fait que plusieurs facteurs influent sur le temps d'exécution.

Le tableau 4.1 montre les temps de calcul des nombres de points entiers dans des unions de $\mathbb{Z}$-polytopes. Certaines unions sont les résultats des transformations par des applications affines de polytopes paramétrés (voir chapitre 5). Sur ce tableau, nous pouvons constater que parmi les facteurs influant sur le temps de calcul, il y a bien sur le nombre de $\mathbb{Z}$-polytopes dans l'union, avant et après l'application du principe d'inclusion-exclusion, mais aussi leur dimension et le nombre de paramètres. Un autre facteur (implicite) important est le dénominateur d'un $\mathbb{Z}$-polytope, i.e., l'ordre de grandeur des coefficients des variables dans le système d'inégalités le définissant. Plus ces facteurs sont grands, plus le temps de calcul augmente. Nous pouvons aussi constater que le nombre de $\mathbb{Z}$-polytopes après l'application du principe d'inclusion-exclusion est proche de $2^n - 1$ lorsque le nombre de $\mathbb{Z}$-polytopes dans l'union est petit (e.g., $3 \rightarrow 7$), où n est le nombre de $\mathbb{Z}$-polytopes de l'union en entrée. En pratique, plus ce nombre augmente, plus on s'éloigne de la borne supérieure $2^n - 1$.

Notons que, comme expliqué dans la section 4.2.2, le comportement polynomial de

Nb. $\mathbb{Z}$-poly.	Dimension	Nb. param.	Nb. $\mathbb{Z}$-poly. après incl./excl.	Temps (s)
2	2	2	3	0.02
3	2	3	5	0.01
3	2	3	7	0.40
3	4	1	7	0.09
3	4	2	7	1.30
7	2	4	11	0.05
7	2	1	41	0.18
9	3	1	9	0.04
13	1	1	30	0.02
16	2	1	186	0.70
32	2	1	32	0.02
35	2	1	42	0.14
38	1	2	48	0.12
82	1	1	96	0.48
191	2	1	250	0.54
471	2	1	1498	10.79

TAB. 4.1 – Temps d'exécution de l'algorithme de comptage de points entiers dans des unions de $\mathbb{Z}$-polytopes paramétrés

l'algorithme de dénombrement des unions de $\mathbb{Z}$-polytopes paramétrés ne concerne que des dimensions (nombre de variables et de paramètres) et des nombres de $\mathbb{Z}$-polytopes *fixés*. Lorsqu'un de ces deux paramètres est grand, l'algorithme peut tout de même avoir un comportement exponentiel.

Chapitre 5

Calcul des images affines de $\mathbb{Z}$-polytopes paramétrés

Plusieurs méthodes d'analyse et d'optimisation de nids de boucles affines soulèvent le problème de comptage des images affines de $\mathbb{Z}$-polytopes paramétrés. Ce problème est équivalent au comptage des solutions entières de formules de Presburger [115, 83][1]. Parmi les applications qui ont recours à ce type de comptage on peut citer : le calcul du nombre de défauts de cache à travers la résolution des équations CME (*Cache Miss Equations*) [64, 33], l'optimisation de la distribution de données d'un programme parallèle en calculant le volume des accès distants pour les machine de type NUMA [69], la compression de la mémoire en connaissant le nombre de localisations mémoire touchées par une boucle [155], la minimisation de la consommation d'énergie dans les systèmes embarqués à hiérarchie mémoire adaptative [41], l'amélioration des performances du cache à travers le calcul de la distance de réutilisation [22, 24] ou la linéarisation de tableaux [97]. Nous reviendrons plus en détail sur ces applications dans le chapitre 6.

Un certain nombre de méthodes de comptage des images affines de $\mathbb{Z}$-polytopes ont été proposées [115, 28, 13, 104, 113, 26]. Mais vu la complexité du problème traité, aucune de ces méthodes n'a pu satisfaire le besoin des différentes applications, le plus souvent parce qu'elles ne sont pas générales ou à cause de leur complexité, en particulier lorsque les $\mathbb{Z}$-polytopes sont paramétrés. Récemment Verdoolaege et al. [144, 142] ont proposé d'appliquer certaines règles de simplification (polynomiales) à des unions disjointes de $\mathbb{Z}$-polytopes, que l'on peut calculer par la librairie Omega [77]. Lorsque ces règles ne peuvent être appliquées, ils utilisent PIP (*Parametric Integer Programming*) [54] (comme dans [28]) pour calculer la solution. Même si cette méthode semble être assez efficace dans plusieurs cas [144], elle reste exponentielle dans le pire des cas, puisque PIP et Omega sont exponentiels dans le pire des cas. Verdoolaege et Woods [149] ont proposé une généralisation au cas paramétré de l'algorithme de Barvinok et Woods [13]. Ces algorithmes sont *théoriquement* polynomiaux, mais leur implémentation reste un vrai challenge. De plus, ils sont probablement moins avantageux par rapport à d'autres algorithmes, pour les problèmes non exponentiels (comme la projection d'un hypercube par exemple). La recherche d'une solution polynomiale reste un problème ouvert.

[1] Une formule de Presburger est un ensemble de contraintes linéaires combinées avec les connecteurs logiques $\neg, \wedge$ et $\vee$, et les quantificateurs $\forall$ et $\exists$.

Dans ce chapitre, nous présentons notre algorithme calculant l'image affine d'un $\mathbb{Z}$-polytope paramétré sous forme d'une union de $\mathbb{Z}$-polytopes [128, 129]. Comme les autres, cet algorithme est exponentiel dans le pire des cas, mais il est cependant assez efficace en pratique et lorsqu'il est comparé avec d'autres algorithmes. Un aperçu de certaines méthodes existantes est également donné vers la fin de ce chapitre.

5.1 De la transformation à la projection

Par image affine d'un $\mathbb{Z}$-polytope $\mathcal{Z} = P \cap L$ de dimension d, nous désignons son image par une transformation affine $T : \mathbb{Z}^d \to \mathbb{Z}^k$ ($k \leq d$), où P est un polytope rationnel et L est un lattice entier. Dans ce qui suit, nous considérons sans perte de généralité que le $\mathbb{Z}$-polytope est *standard*, i.e., $L = \mathbb{Z}^d$. Le cas d'un $\mathbb{Z}$-polytope quelconque est traité comme expliqué dans la section 5.2.3.

Il est bien connu que l'image affine d'un $\mathbb{Z}$-polytope standard peut être décrite par une formule de Presburger [115, 83] :

$$S = \{\mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{x}' \in \mathbb{Z}^{d'} : \ A\mathbf{x}' + B\mathbf{x} + C\mathbf{p} + \mathbf{c} = 0, \ A'\mathbf{x}' + B'\mathbf{x} + C'\mathbf{p} + \mathbf{c}' \geq 0\}, \quad (5.1)$$

où A, B, C, A', B', C' sont des matrices entières, $\mathbf{x}$ et $\mathbf{x}'$ sont des vecteurs des sous-espaces $\mathbb{Z}^d$ et $\mathbb{Z}^{d'}$ de l'espace de données $\mathbb{Z}^{d+d'}$, $\mathbf{c}$ et $\mathbf{c}'$ sont des vecteurs constants, et $\mathbf{p} \in \mathbb{Z}^n$ est un vecteur de paramètres.

```
do i=1, N
  do j=1, N
    do k=1, N
      A(3*i+6*k,  5*i+2*j+1)= ...
    enddo
  enddo
enddo
```

FIG. 5.1 – Nid de boucles paramétré accédant un tableau A.

Exemple 22. Considérons le morceau de programme de la figure 5.1, dans lequel les itérations d'un nid de boucles de profondeur 3 accèdent aux éléments d'un tableau de dimension 2. Contrairement au cas simple que nous avons évoqué dans le chapitre 4, le tableau A est de dimension inférieure à la profondeur du nid de boucles. Cela veut dire que le nombre de ses éléments accédés n'est pas forcément égal au nombre des itérations du nid de boucles, i.e., il peut y avoir une réutilisation des éléments de A. À titre d'exemple, les itérations $(1, 6, 2)$ et $(3, 1, 1)$ accèdent au même élément $A(15, 18)$. Les éléments du tableau A accédés par ce nid de boucles sont donnés par l'image du $\mathbb{Z}$-polytope $\mathcal{Z} = \{(i, j, k) \in \mathbb{Z}^3 \mid 1 \leq i \leq N \ \wedge 1 \leq j \leq N \ \wedge 1 \leq k \leq N\}$ ($N \in \mathbb{Z}$ est un paramètre) par la transformation affine $T(i, j, k) = (3i + 6k, 5i + 2j + 1)$. Ceci est équivalent à la formule de Presburger :

$$S = \{(y, z) \in \mathbb{Z}^2 \mid \exists (i, j, k) \in \mathbb{Z}^3 : \\ 1 \leq i \leq N \ \wedge \ 1 \leq j \leq N \ \wedge \ 1 \leq k \leq N \ \wedge \ y = 3i + 6k \ \wedge \ z = 5i + 2j + 1\},$$

Le problème du calcul l'image d'un $\mathbb{Z}$-polytope par une fonction affine se réduit à l'élimination des variables existentielles (liées par $\exists$) d'une formule de Presburger de la forme (5.1). Dans une telle formule, la conjonction des égalités et des inégalités forme un $\mathbb{Z}$-polytope de dimension non pleine $(d + d')$:

$$\mathcal{Z} = \{(\mathbf{x}', \mathbf{x}) \in \mathbb{Z}^{d'+d} \mid A\mathbf{x}' + B\mathbf{x} + C\mathbf{p} + \mathbf{c} = 0,\ A'\mathbf{x}' + B'\mathbf{x} + C'\mathbf{p} + \mathbf{c}' \geq 0\}. \quad (5.2)$$

Éliminer les variables existentielles de la formule de Presburger (5.1) revient à projeter le $\mathbb{Z}$-polytope (5.2) sur le sous-espace $\mathbb{Z}^d$. Pour ce faire, on commence par en éliminer toutes les égalités impliquant des variables existentielles dans la formule (5.1). Le résultat est un nouveau $\mathbb{Z}$-polytope $\mathcal{Z}'$, donné par l'intersection d'un $\mathbb{Z}$-polytope standard $\mathcal{Y}$ et d'un lattice entier L, dit *lattice de validité*, représentant les valeurs *valides* des variables et paramètres du $\mathbb{Z}$-polytope $\mathcal{Y}$. Ceci permet d'éliminer automatiquement m variables existentielles, où m est le nombre des égalités (non redondantes) éliminées. Dans ce qui suit, nous montrons comment de telles égalités sont éliminées.

5.1.1 Élimination des égalités impliquant des variables existentielles dans un $\mathbb{Z}$-polytope

Pour des raisons de simplicité, nous supposons, sans perte de généralité, que le $\mathbb{Z}$-polytope est non paramétré. En effet, la suppression des égalités d'un $\mathbb{Z}$-polytope *paramétré* se fait de la même façon que pour un $\mathbb{Z}$-polytope non paramétré. À noter cependant que les paramètres sont considérés comme des variables libres. Ils ne sont donc jamais éliminés, et ce quel que soit le nombre des égalités.

Considérons un $\mathbb{Z}$-polytope $\mathcal{Z}(\mathbf{x}', \mathbf{x})$ de dimension non pleine $d' + d$, défini par un système non redondant de contraintes, avec d' variables existentielles et d variables libres, et où le nombre des égalités impliquant des variables existentielles est m. Nous dénotons ces égalités par $E(\mathbf{y}', \mathbf{y})$, où $\mathbf{y}'$ est un vecteur de m variables choisies parmi les d' variables existentielles, et $\mathbf{y}$ est un vecteur des k variables restantes, avec $k = (d' - m) + d, m \leq d'$. Le reste des contraintes, i.e., celles qui correspondent soit à des inégalités, soit des égalités n'impliquant que des variables libres sont dénotées par $\overline{E}(\mathbf{y}', \mathbf{y})$.

Afin d'éliminer les m égalités, il suffit de résoudre le système $E(\mathbf{y}', \mathbf{y})$ pour trouver les valeurs des variables $\mathbf{y}'$ en fonction des variables $\mathbf{y}$, et de les substituer dans le reste des contraintes $\overline{E}(\mathbf{y}', \mathbf{y})$ (comme dans le cas rationnel). Le résultat est donc un nouveau $\mathbb{Z}$-polytope $\mathcal{Y}$ de dimension $k = d + d' - m$. Ce $\mathbb{Z}$-polytope (standard) doit ensuite être intersecté avec les valeurs de $\mathbf{y}$ pour lesquelles le système $E(\mathbf{y}', \mathbf{y})$ admet une solution entière. Les valeurs de $\mathbf{y}$ recherchées sont situées sur un lattice entier calculé à partir du système d'égalités $E(\mathbf{y}', \mathbf{y})$. Nous appelons ce lattice : le *lattice de validité* du $\mathbb{Z}$-polytope résultant $\mathcal{Y}$.

Dans ce qui suit, nous décrivons un algorithme de calcul du lattice de validité d'un ensemble d'égalités impliquant des variables existentielles. Cet algorithme est développé en collaboration avec Benoît Meister (Reservoir Labs, USA) [129].

Calcul du lattice de validité

Considérons un système de m égalités non-redondantes impliquant m variables existentielles et k variables libres :

$$\{\mathbf{y} \in \mathbb{Z}^k \mid \exists \mathbf{y}' \in \mathbb{Z}^m : A\mathbf{y}' + B\mathbf{y} + \mathbf{c} = \mathbf{0}\}, \tag{5.3}$$

où A est une matrice entière $(m \times m)$ de rang-ligne plein, B est une matrice entière $(m \times k)$, et $\mathbf{c}$ est un vecteur de dimension m. Notre objectif est de trouver une condition nécessaire et suffisante sur les variables $\mathbf{y}$ pour qu'il existe une solution entière en $\mathbf{y}'$ du système d'égalités (5.3). Nous allons montrer dans ce qui suit que la condition que nous cherchons est donnée par un lattice entier :

$$\{\mathbf{y} \in \mathbb{Z}^k \mid \forall \mathbf{z} \in \mathbb{Z}^k : \mathbf{y} = G\mathbf{z} + \mathbf{y}_0\} = L(G, \mathbf{y}_0),$$

où G est une matrice entière $(k \times k)$ et $\mathbf{y}_0$ est un vecteur entier de dimension k.

Considérons à nouveau l'équation de (5.3) :

$$A\mathbf{y}' = -B\mathbf{y} - \mathbf{c}. \tag{5.4}$$

Les valeurs entières générées par $A\mathbf{y}'$, pour des valeurs entières de $\mathbf{y}'$, sont celles appartenant au lattice $L(A, \mathbf{0})$. D'autre part, les valeurs entières de $\mathbf{y}$ définissent un lattice $L(-B, -\mathbf{c}) = L(B, -\mathbf{c})$. Le système d'égalités (5.3) définit donc une relation entre un point entier $\mathbf{y}$ et un point entier $\mathbf{y}'$ lorsque le point correspondant $-B\mathbf{y} - \mathbf{c}$ de $L(B, -\mathbf{c})$ est aussi un point $A\mathbf{y}'$ de $L(A, \mathbf{0})$. Ce cas se produit si et seulement si un tel point appartient au lattice $L(F, \mathbf{f}_0)$, donné par l'intersection des deux lattices $L(A, \mathbf{0})$ et $L(B, -\mathbf{c})$. Ceci est équivalent aux deux conditions suivantes :

- il existe au moins un point entier $\mathbf{f}_0$ de $L(B, -\mathbf{c})$ qui appartient à $L(A, \mathbf{0})$. Ce point correspond à la solution entière particulière $(\mathbf{y}'_0, \mathbf{y}_0)$ de (5.3).
- $L(F, \mathbf{0})$ est le plus petit sous-lattice commun de $L(A, \mathbf{0})$ et $L(B, \mathbf{0})$, i.e., il existe une matrice entière U_k (de k lignes) telle que $L(BU_k, \mathbf{0}) = L(F, \mathbf{0})$.

Les points $\mathbf{y}$ pour lesquels il existe une solution entière en $\mathbf{y}'$ pour (5.3) sont donc définis par une relation affine :

$$\mathbf{y} = U_k\mathbf{z}' + \mathbf{y}_0, \ \forall \mathbf{z}' \in \mathbb{Z}^k. \tag{5.5}$$

Démonstration. Des algorithmes connus (voir par exemple [111]) permettent de calculer une solution particulière $(\mathbf{y}'_0, \mathbf{y}_0)$ (lorsqu'elle existe) de (5.3). Nous avons donc

$$A\mathbf{y}'_0 = -B\mathbf{y}_0 - \mathbf{c} = \mathbf{f}_0.$$

Puisque $(\mathbf{y}'_0, \mathbf{y}_0)$ est une solution de (5.3), on a $A\mathbf{y}' + B\mathbf{y} + \mathbf{c} = 0$ et $A\mathbf{y}'_0 + B\mathbf{y}_0 + \mathbf{c} = 0$. La différence des deux équations résulte en :

$$A(\mathbf{y}' - \mathbf{y}'_0) = -B(\mathbf{y} - \mathbf{y}_0).$$

En substituant $\mathbf{y} = U_k\mathbf{z}' + \mathbf{y}_0$ dans l'équation ci-dessus, on obtient :

$$A(\mathbf{y}' - \mathbf{y}'_0) = -BU_k\mathbf{z}' \Rightarrow A\mathbf{y}' = -F\mathbf{z}' + \mathbf{f}_0.$$

Puisque $L(F, \mathbf{f}_0)$ est un sous lattice de $L(A, \mathbf{0})$, il existe un point entier $\mathbf{y}'$ pour toute valeur entière de $\mathbf{z}'$. L'équation (5.5) est donc une condition suffisante, et puisque le lattice $L(F, \mathbf{f}_0)$ est minimal, la condition (5.5) est aussi nécessaire. □

Il nous reste maintenant à calculer la matrice U_k. Pour ce faire, nous nous basons sur le calcul de l'intersection des lattices générés par les vecteurs colonnes de A et de B, en utilisant la forme normale de Hermite :

$$\left(\begin{array}{c|c|c} A & 0 & I \\ \hline 0 & B & I \end{array}\right).U = (H \mid 0),$$

où I est une matrice identité. Ceci peut être réécrit sous la forme :

$$\left(\begin{array}{c|c|c} A & 0 & I \\ \hline 0 & B & I \end{array}\right) \cdot \left(\begin{array}{c|c|c} U_{11} & U_{12} & U_{13} \\ \hline U_{21} & U_{22} & U_{23} \\ \hline U_{31} & U_{32} & U_{33} \end{array}\right) = \left(\begin{array}{c|c|c} I & 0 & 0 \\ \hline H_m & H_k & 0 \end{array}\right), \tag{5.6}$$

où le nombre de colonnes w de $U_{.3}$ est égal au nombre de colonnes zéro dans la matrice à droite de l'équation (5.6). Il est connu [118, 111] que la matrice U_{33} est une matrice qui génère l'intersection des deux lattices $L(A, \mathbf{0})$ et $L(B, \mathbf{0})$. En effet, d'après les w dernières colonnes de (5.6), on a :

$$A.U_{13} = -U_{33} \text{ et } B.U_{23} = -U_{33}.$$

U_{33} est donc un sous-lattice de A et de B. D'autres propriétés de U font que $L(U_{33}, \mathbf{0})$ est le plus petit commun sous-lattice de $L(A, \mathbf{0})$ et $L(B, \mathbf{0})$, i.e., leur intersection. Nous avons donc :

$$L(F, \mathbf{0}) = L(U_{33}, \mathbf{0}) = L(BU_{23}, \mathbf{0}) = L(BU_k, \mathbf{0}).$$

Ce qui montre que U_{23} est bien la matrice recherchée, on pose $U_k = U_{23}$. La matrice U_{23} est donc une matrice entière $(k \times w)$ qui définit le lattice $L(U_k, \mathbf{0})$ de dimension k, avec $w \geq k$. Lorsque $w > k$, le nombre de colonnes de U_{23}, générant le lattice $L(U_k, \mathbf{0})$, n'est pas minimal. Pour obtenir un nombre minimal de vecteurs générateurs, il suffit de calculer la forme normale de Hermite de U_{23} :

$$U_{23}.U' = (G \mid 0),$$

où G est une matrice entière triangulaire inférieure qui génère le lattice $L(U_k, \mathbf{0})$. La condition imposée par l'équation (5.3) peut finalement être donnée par :

$$\mathbf{y} = G\mathbf{z} + \mathbf{y}_0, \ \forall \mathbf{z} \in \mathbb{Z}^k.$$

Ceci correspond à un lattice entier $L(G, \mathbf{y}_0)$ que nous appelons *lattice de validité*.

Exemple 23. Considérons la formule de Presburger :

$$\begin{aligned} S = \{(x, y, z) \in \mathbb{Z}^3 \mid \exists (i, j) \in \mathbb{Z}^2 : \\ 1 \leq i \leq N \wedge 1 \leq j \leq N \wedge 1 \leq x \leq N \wedge 3i + 6j = y \wedge 5i = -2x + z - 1\}. \end{aligned} \tag{5.7}$$

Pour éliminer les égalités de cette formule, il suffit d'éliminer les variables i et j (comme dans le cas rationnel), et d'intersecter le résultat avec le lattice de validité correspondant aux valeurs de x, y, z, et N pour lesquelles le système des égalités admet une solution entière. L'élimination des égalités résulte en un $\mathbb{Z}$-polytope standard de dimension pleine :

$$\mathcal{Y} = \{(x, y, z) \in \mathbb{Z}^3 \mid 27 \leq 6x + 5y - 3z \leq 30N - 3 \wedge 6 \leq -2x + z \leq 5N + 1 \wedge 1 \leq x \leq N\}.$$

Pour calculer le lattice de validité de ce $\mathbb{Z}$-polytope, on commence par calculer la matrice U, telle que

$$\left(\begin{array}{cc|cccc|cc} 3 & 6 & 0 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right) .U = (H \mid 0).$$

La matrice U peut être calculée (en utilisant la `Polylib` par exemple) sous la forme :

$$U = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & -3 & -6 \\ 0 & 1 & 0 & -1 & 2 & 0 & -5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -3 & -6 \\ 0 & 1 & 0 & 0 & 0 & 0 & -5 & 0 \end{pmatrix}.$$

Ensuite, la matrice $G = U_{23}$ est donnée par une sous-matrice de U (lignes de 3 à 6 et colonnes de 5 à 8) :

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -3 & -6 \\ 2 & 0 & -5 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Enfin, la solution particulière $\mathbf{y}_0$ est $(0, 0, 1, 0)$. Le résultat de l'élimination des égalités de la formule (5.7) est donc donné par un nouveau $\mathbb{Z}$-polytope $\mathcal{Z}' = \mathcal{Y} \cap L(G, \mathbf{y}_0)$.

Deux cas sont possibles, une fois que toutes les égalités impliquant des variables existentielles dans une formule de Presburger de la forme (5.1) sont éliminées :

- Toutes les variables existentielles ont été éliminées. Ceci se produit lorsque le nombre des variables existentielles est égal au nombre des égalités les impliquant, i.e., lorsque $m = d'$ et $k = d$. Dans ce cas, la résultat de la transformation est donné simplement par le nouveau $\mathbb{Z}$-polytopes $\mathcal{Z}' = \mathcal{Y} \cap L(G, \mathbf{y}_0)$.
- Il reste encore des variables existentielles à éliminer. Ceci se produit lorsqu'il y a plus de variables existentielles que d'égalités les impliquant, i.e., lorsque $m < d'$ et $k > d$. Dans ce cas, les $d' - m$ variables existentielles restantes sont éliminées en projetant le $\mathbb{Z}$-polytope $\mathcal{Z}' = \mathcal{Y} \cap L(G, \mathbf{y}_0)$ sur l'espace $\mathbb{Z}^d$.

Dans la section suivante, nous présentons notre méthode de projection d'un $\mathbb{Z}$-polytope standard, i.e., lorsque le lattice de validité $L(G, \mathbf{y}_0)$ est égal au lattice standard $\mathbb{Z}^k$. La projection d'un $\mathbb{Z}$-polytope quelconque se réduit aussi à la projection d'un $\mathbb{Z}$-polytope standard. Nous reviendrons sur ce dernier cas dans la section 5.2.3.

5.2 Projection d'un $\mathbb{Z}$-polytope paramétré

La procédure d'élimination des variables existentielles de Fourier-Motzkin [42] permet d'éliminer successivement des variables dans un système d'inégalités linéaires, défini

sur l'ensemble des nombres rationnels ou réels. Elle consiste à réécrire le système original sous forme d'un ensemble de bornes inférieures et supérieures sur une variable z à éliminer. Ensuite, toute paire de bornes, inférieure et supérieure, de la forme : $\{l(\mathbf{x},\mathbf{p}) \leq \beta z,\ \alpha z \leq u(\mathbf{x},\mathbf{p})\}$ est remplacée par son équivalent $\alpha l(\mathbf{x},\mathbf{p}) \leq \beta u(\mathbf{x},\mathbf{p})$, où z est la variable existentielle éliminée, $l(\mathbf{x},\mathbf{p})$ et $u(\mathbf{x},\mathbf{p})$ sont des fonctions affines (indépendantes de z) des variables $\mathbf{x}$ et des paramètres $\mathbf{p}$, et α et β sont des constantes entières strictement positives. Pour pouvoir traiter des entiers, Pugh et al. [115, 116] ont proposé une généralisation de l'algorithme de Fourier-Motzkin. Le principe de cette généralisation, implémentée dans la librairie Omega [114], est le suivant :

Toute paire de bornes, inférieure et supérieure, sur une variable existentielle entière z, de la forme $\{l(\mathbf{x},\mathbf{p}) \leq \beta z,\ \alpha z \leq u(\mathbf{x},\mathbf{p})\}$ définit :

- une **ombre exacte** correspondant à la projection des points *rationnels* appartenant à cette paire de contraintes, donnée par : $\alpha l(\mathbf{x},\mathbf{p}) \leq \beta u(\mathbf{x},\mathbf{p})$.
- une **ombre noire** correspondant à la partie convexe de l'ombre exacte dans laquelle tout point *entier* possède au moins un antécédent *entier* appartenant à la paire de bornes considérée, donnée par $\alpha l(\mathbf{x},\mathbf{p}) + (\alpha-1)(\beta-1) \leq \beta u(\mathbf{x},\mathbf{p})$. Remarquons que si $\alpha = 1$ ou $\beta = 1$, l'ombre noire est égale à l'ombre exacte.

La partie de l'ombre exacte n'appartenant pas à l'ombre noire, comporte généralement des points ayant des antécédents entiers, et d'autres points entiers n'ayant que des antécédents rationnels. Nous appelons ces derniers points : les *trous* (voir la figure 5.2). L'objectif de Omega test [114] est de répondre à la question : existe-t-il un point entier dans la projection d'un polytope dont l'antécédent est entier ? ou autrement dit, une formule de Presburger possède-t-elle une solution entière ? Pour répondre à cette question, Omega test procède comme suit :

- si l'ombre exacte ne comporte aucun point entier, la réponse est : *non*,
- si l'ombre noire comporte au moins un point entier, la réponse est : *oui*,
- sinon, la réponse devient non évidente. Dans ce cas, il faudra savoir si la partie de l'ombre exacte n'appartenant pas à l'ombre noire comporte ou non un point dont l'antécédent est entier.

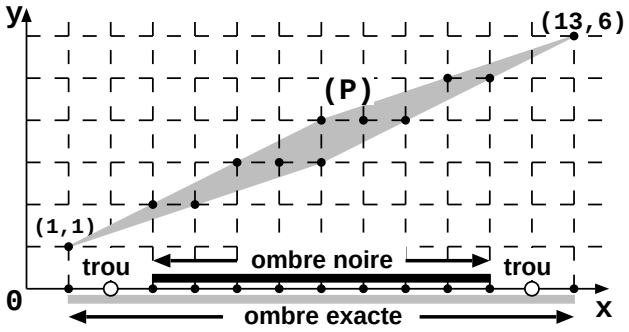


FIG. 5.2 – La projection entière d'un $\mathbb{Z}$-polytope.

Pour répondre à cette dernière question, Pugh et Wonnacott [116] vérifient si l'intersection d'un certain nombre (fonction de α et β) d'hyperplans avec les contraintes du système original comporte ou non un point entier. Cette solution entraîne l'introduction

de nouvelles contraintes (appelées *splinters*) avec éventuellement des variables existentielles supplémentaires. Ce qui complique la réponse aux deux questions suivantes :

– *combien* de points entiers existent-ils dans la projection du polytope ?

– comment *projeter successivement* à travers plusieurs dimensions ?

Noter que les *splinters* produits par la méthode de Pugh sont quelque part similaires à nos $\mathbb{Z}$-polytopes (lorsque les coefficients de la variable existentielle ne sont pas premiers entre eux, voir la section 5.2.1). Mais il n'est pas clair si ces *splinters* peuvent être projetés à travers une autre dimension sans parcourir un grand nombre de polytopes. De plus, lorsque les coefficients sont premiers entre eux, la méthode de Pugh ne propose pas de solution simple comme nous le faisons (voir la section 5.2.1). À noter également que dans [115, 116], aucune explication n'est fournie sur la projection à travers plusieurs dimensions, ni sur la façon dont on peut dénombrer les points entiers appartenant au résultat de ces projections.

Exemple 24. Considérons l'exemple (introduit dans [116]) suivant :

$$\mathcal{Z} = \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} :\ 0 \leq 3y - x \leq 7 \wedge 1 \leq x - 2y \leq 5\}.$$

L'ombre exacte définie par l'élimination de y est donnée par : $3 \leq x \leq 29$ et l'ombre noire est donnée par : $5 \leq x \leq 27$. L'algorithme de W. Pugh et D. Wonnacott [116] calcule un ensemble de contraintes qui contiennent les images n'appartenant pas à l'ombre noire comme suit :

$$\begin{array}{l}\{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} :\ x = 3y \wedge 1 \leq y \leq 5\ \} \cup \\ \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} :\ x = 3y - 1 \wedge 2 \leq y \leq 6\ \} \cup \\ \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} :\ x = 2y + 5 \wedge 5 \leq y \leq 12\}.\end{array}$$

Alors que notre méthode donne ces images directement sous la forme : $x = 3$ et $x = 29$.

5.2.1 Projection d'une paire de bornes sur une variable existentielle

Dans cette section, nous nous focalisons sur la projection d'une paire *unique* de bornes, inférieure et supérieure, sur une variable existentielle choisie pour être éliminée. La projection globale d'un $\mathbb{Z}$-polytope est simplement donnée par l'intersection des projections de toutes ses paires de bornes (comme c'est le cas pour l'algorithme original de Fourier-Motzkin)[2].

Considérons une paire de bornes, inférieure et supérieure, $\{l(\mathbf{x}, \mathbf{p}) \leq \beta z,\ \alpha z \leq u(\mathbf{x}, \mathbf{p})\}$. La projection de cette paire est donnée par l'union de son ombre noire ($\alpha l(\mathbf{x}, \mathbf{p}) - \beta u(\mathbf{x}, \mathbf{p}) + (\alpha - 1)(\beta - 1) \leq 0$) et un autre ensemble de points entiers qui ne peut être obtenu en appliquant des règles simples. La proposition suivante définit les hyperplans sur lesquels se situent ces derniers points.

Lemme 3. *Soient x et y deux nombres rationnels, et $\lceil x \rceil$, $\lceil y \rceil$ (resp. $\lfloor x \rfloor$, $\lfloor y \rfloor$) leurs parties entières supérieures (resp. inférieures). Par définition, les propriétés suivantes sont équivalentes.*

1. $\exists n \in \mathbb{Z}$ tel que $x \leq n \leq y$,

[2]Le résultat de la projection des paires de bornes sur une variable existentielle z est aussi intersecté avec les contraintes qui sont indépendantes de z.

2. $\lceil x \rceil \leq \lfloor y \rfloor$,
3. $\lceil x \rceil \leq y$,
4. $x \leq \lfloor y \rfloor$.

Proposition 2. *Considérons une paire de bornes, inférieure et supérieure, sur une variable existentielle z, $\{l(\mathbf{x},\mathbf{p}) \leq \beta z,\ \alpha z \leq u(\mathbf{x},\mathbf{p})\}$, et soit $l(\mathbf{x},\mathbf{p}) = l_l(\mathbf{x},\mathbf{p}) + c_l$ et $u(\mathbf{x},\mathbf{p}) = l_u(\mathbf{x},\mathbf{p}) + c_u$, où $l_l(\mathbf{x},\mathbf{p})$ et $l_u(\mathbf{x},\mathbf{p})$ sont des fonctions linéaires (avec des parties constantes nulles), c_l et c_u sont des entiers constants, et g est le plus grand commun diviseur* (pgcd) *des coefficients des variables* $\mathbf{x}$ *et des paramètres* $\mathbf{p}$ *dans la fonction linéaire $\alpha l_l(\mathbf{x},\mathbf{p}) - \beta l_u(\mathbf{x},\mathbf{p})$. Alors*

- *les points entiers appartenant à la projection entière de la paire d'inégalités, et n'appartenant pas à son ombre noire, se situent sur des hyperplans de la forme :*

$$\alpha l(\mathbf{x},\mathbf{p}) - \beta u(\mathbf{x},\mathbf{p}) + \gamma = 0, \tag{5.8}$$

  avec $\gamma \in \mathbb{Z}$, $0 \leq \gamma \leq \alpha\beta - \alpha - \beta$ et g divise $(\beta c_u - \alpha c_l - \gamma)$.
- *les valeurs de γ pour lesquelles l'hyperplan (5.8) contient des points qui nous intéressent sont celles vérifiant l'inégalité*

$$\alpha(-l(\mathbf{x},\mathbf{p}) \bmod \beta) \leq \gamma, \tag{5.9}$$

  qui est équivalente à

$$\beta(u(\mathbf{x},\mathbf{p}) \bmod \alpha) \leq \gamma. \tag{5.10}$$

Démonstration. Nous rappelons que les ombres exacte et noire sont respectivement données par : $\alpha l(\mathbf{x},\mathbf{p}) - \beta u(\mathbf{x},\mathbf{p}) \leq 0$ et $\alpha l(\mathbf{x},\mathbf{p}) - \beta u(\mathbf{x},\mathbf{p}) \leq -(\alpha-1)(\beta-1)$ [116]. Par définition, la partie de l'ombre exacte contenant les points de la projection qui sont à l'extérieur de l'ombre noire (voir la figure 5.2) est donnée par : $-(\alpha\beta - \alpha - \beta) \leq \alpha l(\mathbf{x},\mathbf{p}) - \beta u(\mathbf{x},\mathbf{p}) \leq 0$, ce qui est équivalent à $\alpha l(\mathbf{x},\mathbf{p}) - \beta u(\mathbf{x},\mathbf{p}) + \gamma = 0$, où γ est une constante entière telle que $0 \leq \gamma \leq \alpha\beta - \alpha - \beta$.

Soit $l(\mathbf{x},\mathbf{p}) = l_l(\mathbf{x},\mathbf{p}) + c_l$ et $u(\mathbf{x},\mathbf{p}) = l_u(\mathbf{x},\mathbf{p}) + c_u$, où $l_l(\mathbf{x},\mathbf{p})$ et $l_u(\mathbf{x},\mathbf{p})$ sont des fonctions linéaires des variables et des paramètres, et c_l et c_u sont des constantes entières. En substituant les valeurs de $l(\mathbf{x},\mathbf{p})$ et $u(\mathbf{x},\mathbf{p})$ dans (5.8), on obtient :

$$\alpha l_l(\mathbf{x},\mathbf{p}) - \beta l_u(\mathbf{x},\mathbf{p}) = \beta c_u - \alpha c_l - \gamma, \tag{5.11}$$

où $(\alpha l_l(\mathbf{x},\mathbf{p}) - \beta l_u(\mathbf{x},\mathbf{p}))$ est une fonction linéaire (avec une partie constante nulle), et $(\beta c_u - \alpha c_l - \gamma)$ est une constante entière. Une condition nécessaire et suffisante pour que l'hyperplan (5.11) contienne des points entiers est que le *pgcd* des coefficients de la fonction linéaire $(\alpha l_l(\mathbf{x},\mathbf{p}) - \beta l_u(\mathbf{x},\mathbf{p}))$ divise la constante $(\beta c_u - \alpha c_l - \gamma)$.

Par ailleurs, $\{l(\mathbf{x},\mathbf{p}) \leq \beta z,\ \alpha z \leq u(\mathbf{x},\mathbf{p})\}$ est équivalent à $\frac{l(\mathbf{x},\mathbf{p})}{\beta} \leq z \leq \frac{u(\mathbf{x},\mathbf{p})}{\alpha}$ (puisque $\alpha, \beta > 0$), où $\frac{l(\mathbf{x},\mathbf{p})}{\beta}$ et $\frac{u(\mathbf{x},\mathbf{p})}{\alpha}$ sont des fonctions rationnelles. D'après le lemme 3, il existe un entier z tel que $\frac{l(\mathbf{x},\mathbf{p})}{\beta} \leq z \leq \frac{u(\mathbf{x},\mathbf{p})}{\alpha}$ si et seulement si :

$$\left\lceil \frac{l(\mathbf{x},\mathbf{p})}{\beta} \right\rceil \leq \frac{u(\mathbf{x},\mathbf{p})}{\alpha}, \tag{5.12}$$

avec $\left\lceil \frac{l(\mathbf{x},\mathbf{p})}{\beta} \right\rceil = \frac{1}{\beta}(l(\mathbf{x},\mathbf{p}) + (-l(\mathbf{x},\mathbf{p})) \bmod \beta)$. En simplifiant l'inégalité (5.12), on obtient : $\alpha(-l(\mathbf{x},\mathbf{p}) \bmod \beta) \leq \beta u(\mathbf{x},\mathbf{p}) - \alpha l(\mathbf{x},\mathbf{p})$, avec $\beta u(\mathbf{x},\mathbf{p}) - \alpha l(\mathbf{x},\mathbf{p}) = \gamma$ (d'après l'équation (5.8)). L'inégalité (5.9) est donc vérifiée. En appliquant le lemme 3, on peut également prouver l'inégalité (5.10). □

Note 8. *Lorsqu'une des bornes $l(\mathbf{x},\mathbf{p})$ ou $u(\mathbf{x},\mathbf{p})$ est indépendante des variables et des paramètres, l'élimination de la variable existentielle z résulte en l'ombre noire seulement, car dans ce cas, $\left\lceil \frac{l(\mathbf{x},\mathbf{p})}{\beta} \right\rceil$ ou $\left\lfloor \frac{u(\mathbf{x},\mathbf{p})}{\alpha} \right\rfloor$ est remplacé simplement par une constante.*

Nous calculons les solutions de l'inégalité (5.9) ou (5.10) de deux façons différentes, selon que les coefficients α et β sont premiers entre eux ou non.

Cas de coefficients premiers entre eux

Proposition 3. *Considérons la paire de bornes $\{l(\mathbf{x},\mathbf{p}) \leq \beta z,\ \alpha z \leq u(\mathbf{x},\mathbf{p})\}$. Le calcul des valeurs de γ, pour lesquelles l'hyperplan (5.8) comporte des points entiers appartenant à la projection entière de cette paire de contraintes, ne dépend que des constantes α et β, i.e., il est indépendant des variables et des paramètres. Dans ce cas, les inégalités (5.9) et (5.10) sont respectivement équivalentes à (5.13) et (5.14) :*

$$\alpha((c_1\gamma) \bmod \beta) \leq \gamma, \tag{5.13}$$

$$\beta((c_2\gamma) \bmod \alpha) \leq \gamma, \tag{5.14}$$

où c_1 et c_2 sont des constantes entières telles que $c_1\alpha + c_2\beta = 1$.

Démonstration. Lorsque les coefficients α et β de la variable existentielle z sont premiers entre eux, il existe (d'après le théorème dit : *Identité de Bézout*) deux constantes entières c_1 et c_2 telles que

$$c_1\alpha + c_2\beta = 1. \tag{5.15}$$

En multipliant l'égalité (5.8) par c_1, on obtient $c_1\alpha l(\mathbf{x},\mathbf{p}) - c_1\beta u(\mathbf{x},\mathbf{p}) + c_1\gamma = 0$. Ceci est équivalent à $(1 - c_2\beta)l(\mathbf{x},\mathbf{p}) - c_1\beta u(\mathbf{x},\mathbf{p}) + c_1\gamma = 0$ (d'après (5.15)). D'où

$$l(\mathbf{x},\mathbf{p}) = \beta(c_2 l(\mathbf{x},\mathbf{p}) + c_1 u(\mathbf{x},\mathbf{p})) - c_1\gamma.$$

En substituant la valeur de $l(\mathbf{x},\mathbf{p})$ dans l'inégalité (5.9), on obtient

$$\alpha((-\beta(c_2 l(\mathbf{x},\mathbf{p}) + c_1 u(\mathbf{x},\mathbf{p})) + c_1\gamma) \bmod \beta) \leq \gamma \Leftrightarrow \alpha(c_1\gamma \bmod \beta) \leq \gamma,$$

puisque $-\beta(c_2 l(\mathbf{x},\mathbf{p}) + c_1 u(\mathbf{x},\mathbf{p}))$ est un multiple de β. En procédant d'une façon similaire sur l'inégalité (5.10), on obtient (5.14). □

Exemple 25. Considérons la formule de Presburger suivante :

$$\mathcal{S} = \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} :\ 2 \leq 3y - x \leq 5 \wedge -1 \leq x - 2y \leq p - 1\},$$

où p est un paramètre strictement positif ($p > 0$). Lorsque $p = 2$, l'ensemble $\mathcal{S}$ est équivalent à la projection, sur l'axe des x, du $\mathbb{Z}$-polytope de la figure 5.2.

D'après la paire de bornes $\{x - p + 1 \leq 2y,\ 3y \leq x + 5\}$, on a

$$l(x,p) = x - p + 1,\ u(x,p) = x + 5,\ \alpha = 3,\ \beta = 2.$$

Nous pouvons choisir $c_1 = 1$ et $c_2 = -1$ puisque $1 \times 3 + (-1) \times 2 = 1$.

La contrainte sur l'ombre noire correspondant à cette paire de bornes est $x \leq 3p+5$. Le reste des points de la projection de cette paire, i.e., ceux qui n'appartiennent pas à l'ombre noire sont donnés par :

$$\alpha l(x,p) - \beta u(x,p) + \gamma = 0,\ 0 \leq \gamma \leq \alpha\beta - \alpha - \beta \text{ et } \alpha((c_1\gamma) \bmod \beta) \leq \gamma$$

$$\Rightarrow x - 3p - 7 + \gamma = 0,\ 0 \leq \gamma \leq 1 \text{ et } 3(\gamma \bmod 2) \leq \gamma.$$

En parcourant les valeurs de γ, on trouve que la seule valeur qui satisfait ces contraintes est $\gamma = 0$. Le point correspondant (un hyperplan de le cas général) est $x = 3p + 7$. En procédant d'une manière similaire, on peut calculer le point $x = 1$ à partir de la deuxième paire de bornes $\{x + 2 \leq 3y,\ 2y \leq x + 1\}$ ayant une ombre noire égale à $x \geq 3$. Les autres paires de bornes sont redondantes, puisque p est strictement positif.

La projection du $\mathbb{Z}$-polytope peut ensuite être obtenue en intersectant les projections des deux paires de contraintes, i.e., $S = \{x = 3p + 7 \vee x \leq 3p + 5\} \cap \{x = 1 \vee x \geq 3\}$. Puisque le paramètre p est considéré strictement positif, le résultat final est

$$\mathcal{S} = \{x \in \mathbb{Z} \mid x = 1 \vee 3 \leq x \leq 3p + 5 \vee x = 3p + 7\}.$$

Cas de coefficients non premiers entre eux

Nous nous intéressons maintenant au calcul de la projection entière d'une paire de bornes, inférieure et supérieure, sur une variable existentielle z, lorsque les coefficients α et β de z ne sont pas premiers entre eux. Dans ce cas, le calcul des valeurs de γ, pour lesquelles l'hyperplan (5.8) comporte des points de la projection entière de la paire de contraintes considérée, dépend des coefficients α et β, mais aussi des variables et paramètres. Soit $g' = pgcd(\alpha, \beta)$, $\alpha' = \alpha/g'$, $\beta' = \beta/g'$, et soit g le *pgcd* des coefficients des variables et paramètres dans la fonction linéaire $\alpha' l_l(\mathbf{x}, \mathbf{p}) - \beta' l_u(\mathbf{x}, \mathbf{p})$, avec $l(\mathbf{x}, \mathbf{p}) = l_l(\mathbf{x}, \mathbf{p}) + c_l$ et $u(\mathbf{x}, \mathbf{p}) = l_u(\mathbf{x}, \mathbf{p}) + c_u$ (voir la proposition 2). On peut ainsi écrire l'équation de l'hyperplan (5.8) sous la forme :

$$\alpha' l(\mathbf{x}, \mathbf{p}) - \beta' u(\mathbf{x}, \mathbf{p}) + \gamma = 0, \tag{5.16}$$

avec $\gamma \in \mathbb{Z}$, $0 \leq \gamma \leq \alpha\beta' - \alpha' - \beta'$ et g divise $(\beta' c_u - \alpha' c_l - \gamma)$.

Les inégalités (5.9) et (5.10) peuvent être réécrites respectivement sous la forme (5.17) et (5.18) :

$$\alpha'(-l(\mathbf{x}, \mathbf{p}) \bmod \beta) \leq \gamma, \tag{5.17}$$

$$\beta'(u(\mathbf{x}, \mathbf{p}) \bmod \alpha) \leq \gamma. \tag{5.18}$$

Dans ce cas, il se peut que seulement un *sous-ensemble* des points entiers de l'hyperplan (5.16) appartiennent à la projection entière de la paire de bornes. Ces points

sont définis par l'intersection de cet hyperplan avec une union de lattices obtenus en résolvant une des équations *modulo* suivantes

$$-l(\mathbf{x},\mathbf{p}) \bmod \beta = \gamma',\ \text{avec } 0 \leq \gamma' \leq \min\left(\left\lfloor \frac{\gamma}{\alpha'} \right\rfloor, \beta\right), \tag{5.19}$$

$$u(\mathbf{x},\mathbf{p}) \bmod \alpha = \gamma',\ \text{avec } 0 \leq \gamma' \leq \min\left(\left\lfloor \frac{\gamma}{\beta'} \right\rfloor, \alpha\right). \tag{5.20}$$

En pratique, il est préférable de considérer l'équation (5.19) lorsque $\beta < \alpha$ et l'équation (5.20) sinon (pour minimiser les valeurs de γ' à parcourir).

La solution d'une équation modulo : $f(\mathbf{x},\mathbf{p}) \bmod a = b$ est un lattice de la forme :

$$L = \left\{ \left(\begin{array}{c|c} A_{\mathbf{x}} & A_{\mathbf{p}} \end{array} \right) \left(\begin{array}{c} \mathbf{x} \\ \mathbf{p} \end{array} \right) + \mathbf{c} \;\middle|\; \mathbf{x} \in \mathbb{Z}^d,\ \mathbf{p} \in \mathbb{Z}^n \right\}, \tag{5.21}$$

où $A_{\mathbf{x}}, A_{\mathbf{p}}$ sont des matrices entières, $\mathbf{c}$ est un vecteur entier, $\mathbf{x}$ est un vecteur de l'espace de données, et $\mathbf{p}$ est un vecteur de paramètres. Nous calculons cette solution en utilisant l'algorithme de calcul du lattice de validité décrit dans la section 5.1.1. En effet, $f(\mathbf{x},\mathbf{p}) \bmod a = b$ est équivalente à : $\exists z \in \mathbb{Z} : f(\mathbf{x},\mathbf{p}) = az + b$. Il suffit donc de calculer le lattice de validité de cette dernière équation en y supprimant la variable existentielle z, ou autrement dit, toutes les valeurs entières de $\mathbf{x}$ et $\mathbf{p}$ pour lesquelles la variable z est entière.

Exemple 26. Considérons une paire de bornes dans laquelle les coefficients de la variable existentielle y ne sont pas premiers entre eux $\{x - p - 2 \leq 2y,\ 4y \leq x + 5\}$, on a :

$$l(x,p) = x - p - 2,\ u(x,p) = x + 5,\ \alpha = 4,\ \beta = 2$$

$$\Rightarrow pgcd(\alpha,\beta) = 2,\ \alpha' = 2,\ \beta' = 1.$$

La contrainte sur l'ombre noire est $2x \leq 4p + 15 \Leftrightarrow x \leq 2p + 7$. Les points entiers qui sont à l'extérieur de cette ombre et qui appartiennent à la projection entière de la paire de contraintes sont portés par l'hyperplan

$$x - 2p - 9 + \gamma = 0,\ \text{tel que } 0 \leq \gamma \leq 1 \text{ et } 2((x - p - 2) \bmod 2) \leq \gamma.$$

Pour les deux valeurs de γ, la solution de l'inégalité modulo ci-dessus est le lattice

$$L = \left\{ \left(\begin{array}{cc} 2 & 1 \\ 0 & 1 \end{array} \right) \left(\begin{array}{c} x \\ p \end{array} \right) + \left(\begin{array}{c} -2 \\ 0 \end{array} \right) \;\middle|\; x \in \mathbb{Z},\ p \in \mathbb{Z} \right\}.$$

Les points $x = 2p + 9$ et $x = 2p + 8$ (obtenus en substituant les valeurs de γ dans l'égalité $x - 2p - 9 + \gamma = 0$) appartiennent donc à la projection seulement si x et p appartiennent au lattice L. Le résultat final de la projection de la paire de bornes est donné sous la forme :

$$\mathcal{S} = \{x \in \mathbb{Z} \mid (x = 2p + 8 \wedge (x,p) \in L) \vee (x = 2p + 9 \wedge (x,p) \in L) \vee x \leq 2p + 7\}.$$

5.2.2 Projection d'un $\mathbb{Z}$-polytope à travers plusieurs dimensions

Nous nous intéressons maintenant à un $\mathbb{Z}$-polytope avec plusieurs variables existentielles (au moins deux), et où les égalités, si elles existent, n'impliquent pas des variables existentielles. Comme nous l'avons montré dans la section précédente, l'élimination d'une première variable existentielle de ce $\mathbb{Z}$-polytope résulte en une ombre noire (un $\mathbb{Z}$-polytope standard) et un ensemble de $\mathbb{Z}$-polytopes de dimension non pleine, i.e., ayant au moins une égalité. Afin d'éliminer une deuxième variable existentielle, il suffit de projeter l'ombre noire de la façon décrite dans la section 5.2.1, et d'éliminer une des égalités de chaque $\mathbb{Z}$-polytope de dimension non pleine, comme suit.

Nous supposons, sans perte de généralité, que tous les $\mathbb{Z}$-polytopes (de dimension non pleine) sont *standards*, i.e., leurs lattices respectifs peuvent être ignorés sans affecter le nombre de points entiers qu'ils comportent ($\mathcal{E}(\mathcal{Y}_i \cap L_i) = \mathcal{E}(\mathcal{Y}_i)$). Si ce n'est pas le cas, le principe reste le même, mais nous devons traiter les différents lattices comme pour la projection d'un $\mathbb{Z}$-polytope quelconque (ce cas est décrit dans la section 5.2.3).

Soit z la deuxième variable existentielle choisie pour être éliminée. Deux cas sont possibles pour un $\mathbb{Z}$-polytope $\mathcal{Y}_i$ (de dimension non pleine), obtenu après élimination d'une première variable existentielle :

- aucune égalité du $\mathbb{Z}$-polytope $\mathcal{Y}_i$ n'implique la variable existentielle z, i.e., toutes ses égalités sont de la forme $0.z + \langle \mathbf{a}_j, \mathbf{x} \rangle + c_j = 0$. Dans ce cas, il suffit de projeter les *inégalités* de $\mathcal{Y}_i$, comme expliqué dans la section 5.2.1, et d'intersecter le résultat avec les égalités $\langle \mathbf{a}_j, \mathbf{x} \rangle + c_j = 0$.
- au moins une égalité de $\mathcal{Y}_i$ implique la variable existentielle z, i.e., elle est de la forme $bz + \langle \mathbf{a}, \mathbf{x} \rangle + c = 0$, avec $b \neq 0$. Dans ce cas, il suffit d'éliminer cette égalité, en appliquant l'algorithme de la section 5.1.1, pour éliminer la variable z.

Cette procédure est exécutée récursivement jusqu'à ce que toutes les variables existentielles soient éliminées.

Exemple 27. Soit S une formule de Presburger définie par :

$$S = \left\{ x \in \mathbb{Z} \;\middle|\; \exists (i,j) \in \mathbb{Z}^2 : \begin{array}{rcl} -3x - 2j & \leq 4i \leq & -3x - 2j + p + 3 \wedge \\ -2x - 3j - 6 & \leq 3i \leq & -2x - 3j - 2 \wedge \\ 3 & \leq 3j \leq & 2p \end{array} \right\}. \quad (5.22)$$

L'élimination de la variable i, comme expliqué dans la section 5.2.1, résulte en une union d'une ombre noire

$$S_1 = \left\{ x \in \mathbb{Z} \;\middle|\; \exists j \in \mathbb{Z} : \begin{array}{rcl} x - 3p - 27 & \leq 6j \leq & x - 14 \wedge \\ 3 & \leq 3j \leq & 2p \end{array} \right\}$$

et de 6 autres formules comportant chacune une égalité.

$$\begin{array}{l} S_2 = \{x \in \mathbb{Z} \mid \exists j \in \mathbb{Z} : \; x - 6j - 8 = 0 \wedge 3 \leq 3j \leq 2p\} \cup \\ S_3 = \{x \in \mathbb{Z} \mid \exists j \in \mathbb{Z} : \; x - 6j - 11 = 0 \wedge 3 \leq 3j \leq 2p\} \cup \\ S_4 = \{x \in \mathbb{Z} \mid \exists j \in \mathbb{Z} : \; x - 6j - 12 = 0 \wedge 3 \leq 3j \leq 2p\} \cup \\ S_5 = \{x \in \mathbb{Z} \mid \exists j \in \mathbb{Z} : \; x - 6j - 3p - 33 = 0 \wedge 3 \leq 3j \leq 2p\} \cup \\ S_6 = \{x \in \mathbb{Z} \mid \exists j \in \mathbb{Z} : \; x - 6j - 3p - 30 = 0 \wedge 3 \leq 3j \leq 2p\} \cup \\ S_7 = \{x \in \mathbb{Z} \mid \exists j \in \mathbb{Z} : \; x - 6j - 3p - 29 = 0 \wedge 3 \leq 3j \leq 2p\}. \end{array}$$

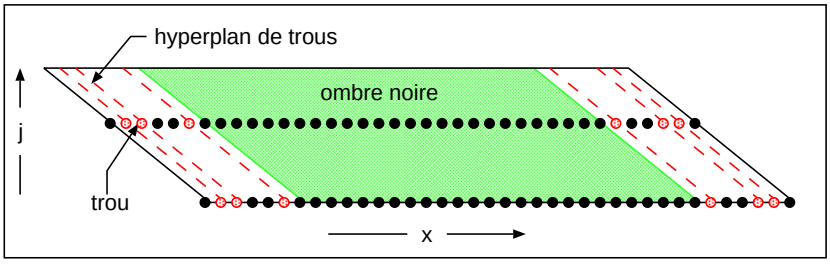


FIG. 5.3 – Illustration de l'élimination de la variable i de la formule (5.22) lorsque $p = 4$. Les hyperplans de trous (lignes en pointillé) sont des hyperplans dont les points entiers n'appartiennent pas à la transformation.

Ce résultat est illustré par la figure 5.3.

Le résultat final est obtenu en éliminant la variable j du résultat de la première projection ($\bigcup_{i=1}^{7} S_i$). Les contraintes de la formule S_1 forment un $\mathbb{Z}$-polytope standard de dimension pleine. Elle est donc traitée, comme la formule originale, par l'algorithme de la section 5.2.1. Le résultat est donné par une union d'une ombre noire :

$$\mathcal{Z}' = \{x \in \mathbb{Z} \mid 20 \leq x \leq 7p + 23 \ \wedge \ p \geq 2\},$$

et d'un ensemble de $\mathbb{Z}$-polytopes de dimension non pleine :

$$\begin{aligned}
\mathcal{Z}_1 &= \{x \in \mathbb{Z} \mid x = 7p + 27 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right), \\
\mathcal{Z}_2 &= \{x \in \mathbb{Z} \mid x = 7p + 26 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right), \\
\mathcal{Z}_3 &= \{x \in \mathbb{Z} \mid x = 7p + 25 \ \wedge \ p \geq 2\} \cap \left\{L\left(\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}\right) \cup L\left(\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right)\right\}, \\
\mathcal{Z}_4 &= \{x \in \mathbb{Z} \mid x = 7p + 24 \ \wedge \ p \geq 2\} \cap \left\{L\left(\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}\right) \cup L\left(\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right)\right\},
\end{aligned}$$

Notons que nous avons obtenu ces lattices puisque les coefficients de j (6 et 3) ne sont pas premiers entre eux.

Les contraintes des formules $S_2, \cdots, S_7$ forment des $\mathbb{Z}$-polytopes de dimension non pleine. Donc pour éliminer la variable j, il suffit d'éliminer l'égalité de chaque $\mathbb{Z}$-

polytope, comme décrit dans la section 5.1.1. Le résultat est :

$$\mathcal{Z}_5 = \{x \in \mathbb{Z} \mid 14 \leq x \leq 4p+8 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix}6 & 0\\ 0 & 1\end{pmatrix}, \begin{pmatrix}2\\0\end{pmatrix}\right),$$
$$\mathcal{Z}_6 = \{x \in \mathbb{Z} \mid 17 \leq x \leq 4p+11 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix}6 & 0\\ 0 & 1\end{pmatrix}, \begin{pmatrix}5\\0\end{pmatrix}\right),$$
$$\mathcal{Z}_7 = \{x \in \mathbb{Z} \mid 18 \leq x \leq 4p+12 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix}6 & 0\\ 0 & 1\end{pmatrix}, \begin{pmatrix}0\\0\end{pmatrix}\right),$$
$$\mathcal{Z}_8 = \{x \in \mathbb{Z} \mid 3p+39 \leq x \leq 7p+33 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix}3 & 0\\ 1 & 2\end{pmatrix}, \begin{pmatrix}33\\-66\end{pmatrix}\right),$$
$$\mathcal{Z}_9 = \{x \in \mathbb{Z} \mid 3p+36 \leq x \leq 7p+30 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix}3 & 0\\ 1 & 2\end{pmatrix}, \begin{pmatrix}30\\-60\end{pmatrix}\right),$$
$$\mathcal{Z}_{10} = \{x \in \mathbb{Z} \mid 3p+35 \leq x \leq 7p+29 \ \wedge \ p \geq 2\} \cap L\left(\begin{pmatrix}3 & 0\\ 1 & 2\end{pmatrix}, \begin{pmatrix}29\\-58\end{pmatrix}\right).$$

Enfin, le résultat global est donné par l'union : $\bigcup_{i=1}^{10} \mathcal{Z}_i \cup \mathcal{Z}'$. Le nombre de points entiers dans cette union peut être donné, en utilisant l'algorithme 2 (chapitre 4), sous la forme :

$$\mathcal{E}(S) = 7p + [14, 10, 12]_p \text{ si } p \geq 2 \text{ et } 0 \text{ sinon.}$$

Nous avons vu dans la section 5.2.1 que la projection d'un $\mathbb{Z}$-polytope standard peut résulter en une union de $\mathbb{Z}$-polytopes *non standards*. Afin de pouvoir projeter ce résultat à travers une autre variable existentielle, nous avons besoin d'un algorithme qui permet de calculer la projection d'un $\mathbb{Z}$-polytope quelconque.

5.2.3 Projection d'un $\mathbb{Z}$-polytope quelconque

Soit $\mathcal{Z} = P \cap L$ un $\mathbb{Z}$-polytope paramétré, avec

$$P = \left\{ (\mathbf{x}', \mathbf{x}) \in \mathbb{Q}^{d'+d}, \mathbf{p} \in \mathbb{Z}^n \;\middle|\; (A_{\mathbf{x}'} \mid A_{\mathbf{x}} \mid A_{\mathbf{p}}) \begin{pmatrix}\mathbf{x}'\\ \mathbf{x}\\ \mathbf{p}\end{pmatrix} + \mathbf{a} \geq 0 \right\},$$

$$L = \left\{ (B_{\mathbf{x}'} \mid B_{\mathbf{x}} \mid B_{\mathbf{p}}) \begin{pmatrix}\mathbf{x}'\\ \mathbf{x}\\ \mathbf{p}\end{pmatrix} + \mathbf{b} \;\middle|\; (\mathbf{x}', \mathbf{x}) \in \mathbb{Z}^{d'+d}, \mathbf{p} \in \mathbb{Z}^n \right\},$$

où, $\mathbf{x}'$ est un vecteur de variables existentielles, $\mathbf{x}$ est un vecteur de variables libres, et $\mathbf{p}$ est un vecteur de paramètres. Le polytope P peut aussi comporter des égalités (implicites). Afin d'éliminer les variables existentielles $\mathbf{x}'$ de ce $\mathbb{Z}$-polytope, ou autrement dit, le projeter sur l'espace $\mathbb{Z}^d$, nous procédons comme suit :

D'abord la matrice base $(B_{\mathbf{x}'} \mid B_{\mathbf{x}} \mid B_{\mathbf{p}})$ du lattice L est transformée en une matrice équivalente (qui génère le même lattice) de la forme :

$$M = \left(\begin{array}{c|c|c} B_{\mathbf{x}'\mathbf{x}'} & B_{\mathbf{x}'\mathbf{x}} & B_{\mathbf{x}'\mathbf{p}} \\ \hline 0 & B_{\mathbf{x}\mathbf{x}} & B_{\mathbf{x}\mathbf{p}} \\ \hline 0 & B_{\mathbf{p}\mathbf{x}} & B_{\mathbf{p}\mathbf{p}} \end{array}\right).$$

Il est toujours possible de calculer une matrice de cette forme en trois étapes :

– organiser le lattice L de façon à ce que les variables existentielles soient mises en dernier,
– remplacer la matrice base du nouveau lattice par sa forme normale de Hermite [126],
– réorganiser le nouveau lattice de façon à revenir à l'ordre initial des variables et paramètres.

La matrice M est ensuite utilisée pour transformer le $\mathbb{Z}$-polytope $\mathcal{Z}$ en un $\mathbb{Z}$-polytope standard en calculant la préimage des points entiers de P par la transformation $(M \mid \mathbf{b})$. Les contraintes du $\mathbb{Z}$-polytope résultant, disons $\mathcal{Z}'$, sont données par la multiplication de la matrice de contraintes du polytope P et celle du nouveau lattice L', i.e.,

$$\mathcal{Z}' = \left\{ (\mathbf{x}', \mathbf{x}) \in \mathbb{Z}^{d'+d}, \mathbf{p} \in \mathbb{Z}^n \;\middle|\; (A_{\mathbf{x}'} \mid A_{\mathbf{x}} \mid A_{\mathbf{p}}) \left(M \mid \mathbf{b}\right) \begin{pmatrix} \mathbf{x}' \\ \mathbf{x} \\ \mathbf{p} \end{pmatrix} + \mathbf{a} \geq 0 \right\}.$$

À ce stade, les variables existentielles du $\mathbb{Z}$-polytope standard $\mathcal{Z}'$ sont éliminées comme nous l'avons expliqué dans les sections précédentes. C'est-à-dire, d'abord en éliminant ses égalités (lorsqu'elles existent) en utilisant l'algorithme de la section 5.1.1, puis en appliquant l'algorithme de la section 5.2.1 pour calculer sa projection l'espace $\mathbb{Z}^d$. Le résultat de cette projection est donné par une union de $\mathbb{Z}$-polytopes de dimension d (éventuellement non pleine). Enfin, puisque $\mathcal{Z}'$ est le résultat de la compression d'un $\mathbb{Z}$-polytope qui préserve le nombre de points entiers mais pas leurs coordonnées (voir la section 4.1.4), nous devons réécrire les $\mathbb{Z}$-polytopes résultant de la projection de $\mathcal{Z}'$ en fonction de leurs coordonnées originales. Plus concrètement, le résultat final est donné par les images de tous les $\mathbb{Z}$-polytopes par la sous-matrice $\left(\begin{array}{c|c} B_{\mathbf{xx}} & B_{\mathbf{xp}} \\ \hline B_{\mathbf{px}} & B_{\mathbf{pp}} \end{array}\right)$ de M.

Exemple 28. Dans l'exemple 23, nous avons vu que l'élimination des égalités d'une formule de Presburger résulte en un $\mathbb{Z}$-polytope $\mathcal{Z} = \mathcal{Y} \cap L(G, \mathbf{y}_0)$, avec

$$\mathcal{Y} = \{(x,y,z) \in \mathbb{Z}^3 \mid 27 \leq 6x{+}5y{-}3z \leq 30N{-}3 \wedge 6 \leq -2x{+}z \leq 5N{+}1 \wedge 1 \leq x \leq N\}, \text{ et}$$

$$L(G, \mathbf{y}_0) = L\left(\left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 0 & -3 & -6 \\ 2 & 0 & -5 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right), \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \end{array}\right)\right).$$

Supposons que l'on veuille projeter ce $\mathbb{Z}$-polytope sur l'espace (y, z, N), i.e., éliminer la variable x. Pour ce faire, le lattice $L(G, \mathbf{y}_0)$ doit d'abord être normalisé sous la forme :

$$L(G', \mathbf{x}_0) = L\left(\left(\begin{array}{c|ccc} 5 & 3 & 1 & 0 \\ \hline 0 & 3 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right), \left(\begin{array}{c} 0 \\ \hline 0 \\ 1 \\ 0 \end{array}\right)\right).$$

La matrice $(G' \mid \mathbf{y}_0)$ est utilisée pour calculer la préimage du $\mathbb{Z}$-polytope $\mathcal{Y}$ sous la forme :

$$\mathcal{Z}' = \{(x,y,z) \in \mathbb{Z}^3 \mid 1 \leq -2x - y \leq N \wedge 1 \leq 5x + 3y + z \leq N \wedge 1 \leq -x - y \leq N\}.$$

Algorithme 3 Calcul de l'image affine d'un $\mathbb{Z}$-polytope

Entrée :
$\mathcal{Z}$: $\mathbb{Z}$-Polytope
T : Matrice de transformation
Sortie :
$\mathcal{Z}_u$: Union de $\mathbb{Z}$-polytopes
Variables :
F, U : Formules de Presburger

F = FormuleDePresburger $(\mathcal{Z}, T)$
F = EliminaterLesEgalities (F)

// Elimination des variables existentielles
Pour Toute variable existentielle v dans F
  F = NormaliserLesLattices (v, F)
  U = Univers $(\text{Dim}(F) - 1)$
  Pour Toute paire de bornes $\{\alpha u, \beta l\}$ sur v dans F
    D = OmbreNoire $(\alpha u, \beta l)$
    Si $\alpha = 1$ ou $\beta = 1$
      $U = U \cap D$
    Sinon
      E = OmbreExacte $(\alpha u, \beta l)$
      $U = U \cap (D \cup \text{EliminerLesTrous } (E - D, \alpha u, \beta l))$
    Fin Si
  Fin Pour
  $F = U$
Fin Pour

// F est mintenant une union de $\mathbb{Z}$-polytopes
$\mathcal{Z}_u = F$

Ce $\mathbb{Z}$-polytope peut ensuite être projeté comme expliqué dans la section 5.2.1. Enfin, le résultat global est donné par l'image de l'union des $\mathbb{Z}$-polytopes résultant de la projection de $\mathcal{Z}'$ par la matrice $\left(\begin{array}{ccc|c} 3 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right)$.

Le calcul de l'image affine d'un $\mathbb{Z}$-polytope est résumé par l'algorithme 3.

5.3 Dénombrement des images affines de $\mathbb{Z}$-polytopes

Nous avons vu jusque-là comment calculer l'image affine *exacte* d'un $\mathbb{Z}$-polytope paramétré. Le plus souvent, et en particulier dans le contexte de l'optimisation de programmes, nous avons besoin de connaître le nombre de points entiers que comporte une telle image. Nous calculons ce nombre en utilisant l'algorithme de dénombrement des unions de $\mathbb{Z}$-polytopes paramétrés, décrit dans le chapitre 4.

Par ailleurs, lorsqu'on n'est intéressé que par le nombre de points que comporte une image affine d'un $\mathbb{Z}$-polytope, il est parfois utile de ne pas calculer l'image elle même, mais plutôt un ensemble équivalent ayant le même nombre de points entiers, et qui peut être calculé plus facilement. Ceci est toujours possible lorsqu'il n'y a qu'une seule variable existentielle (après élimination des égalités). Dans ce qui suit nous présentons brièvement une méthode, due à Clauss [37], pour éliminer une variable existentielle *unique*. Cette méthode, utilisée (optionnellement) par notre algorithme de projection, est prouvée dans [127]. Nous l'appelons *méthode des facettes*.

Proposition 4. *Soit $\mathcal{Z}$ un $\mathbb{Z}$-polytope standard de dimension pleine d, défini par un ensemble de m inégalités de la forme : $\langle \mathbf{a}_i, \mathbf{x} \rangle + c_i \geq 0$, avec $1 \leq i \leq m$ (ici les paramètres sont considérés comme variables régulières), et soit T une transformation linéaire définie par une matrice entière A de dimensions $(n \times d)$, tel que $T(\mathcal{Z}) = \{A\mathbf{x},\ \forall \mathbf{x} \in \mathcal{Z}\}$. La transformation T est supposée linéaire, puisque la partie affine d'une transformation se traduit simplement par une translation par un vecteur entier, qui n'a aucune influence sur le nombre de points entiers de la transformation. Dénotons par $Ker(A)$ le noyau de la matrice A, i.e., l'ensemble $\{\mathbf{x} \in Q^d \mid A\mathbf{x} = \mathbf{0}\}$. $Ker(A) = \phi$ si et seulement s'il y a une bijection entre les points entiers de $\mathcal{Z}$ et ceux de sa transformation par T. Dans ce cas, le nombre de points entiers de la transformation est égal simplement au nombre de points entiers du $\mathbb{Z}$-polytope $\mathcal{Z}$. Lorsque la dimension de $Ker(A)$ est égal à 1, il est possible de trouver une région sur les bords du $\mathbb{Z}$-polytope $\mathcal{Z}$ contenant le même nombre de points entiers que celui de la transformation $T(\mathcal{Z})$. Cette région est calculée comme suit.*

Soit $\mathbf{b}$ le vecteur base de $Ker(A)$, La région que nous cherchons est donnée par une union de $\mathbb{Z}$-polytopes $\mathcal{Z}_j$ $(1 \leq j \leq m)$. Chaque $\mathbb{Z}$-polytope $\mathcal{Z}_j$ est obtenu en intersectant le $\mathbb{Z}$-polytope $\mathcal{Z}$ avec la contrainte : $-(\langle \mathbf{a}_j, \mathbf{x} \rangle + c_j) + \langle \mathbf{b}, \mathbf{a}_j \rangle - 1 \geq 0$, où $\langle \mathbf{a}_j, \mathbf{x} \rangle + c_j \geq 0$ est la $j^{ème}$ contrainte de $\mathcal{Z}$, telle que $\langle \mathbf{b}, \mathbf{a}_j \rangle > 0$.

Démonstration. La démonstration de cette proposition est donnée dans [127]. □

Exemple 29. Considérons le $\mathbb{Z}$-polytope $\mathcal{Z}$ de la figure 5.4, défini par le système d'inégalités suivant :

$$\begin{cases} C_1 : & x + 3y - 8 & \geq 0 & \Rightarrow \mathbf{a}_1 = (1, 3) \\ C_2 : & -2x + 3y + 7 & \geq 0 & \Rightarrow \mathbf{a}_2 = (-2, 3) \\ C_3 : & -2x - y + 19 & \geq 0 & \Rightarrow \mathbf{a}_3 = (-2, -1) \\ C_4 : & 2x + -5y + 23 & \geq 0 & \Rightarrow \mathbf{a}_4 = (2, -5) \\ C_5 : & 3x + y - 8 & \geq 0 & \Rightarrow \mathbf{a}_5 = (3, 1) \end{cases}$$

Le nombre de points entiers dans l'image de $\mathcal{Z}$ par la transformation $T = (2\ -1)\begin{pmatrix} x \\ y \end{pmatrix}$ est égal au nombre de points entiers dans la région grise de la figure 5.4. Cette région est construite en intersectant le $\mathcal{Z}$-polytope $\mathcal{Z}$ avec les contraintes H_1, H_2 et H_5 obtenues comme suit.

Sachant que le vecteur base de $Ker(A)$ est $\mathbf{b} = (1, 2)$, les contraintes à rajouter H_i sont calculées à partir des contraintes C_i dont le produit scalaire $w_i = \langle \mathbf{a}_i, \mathbf{b} \rangle$ est strictement positif, avec $1 \leq i \leq 5$.

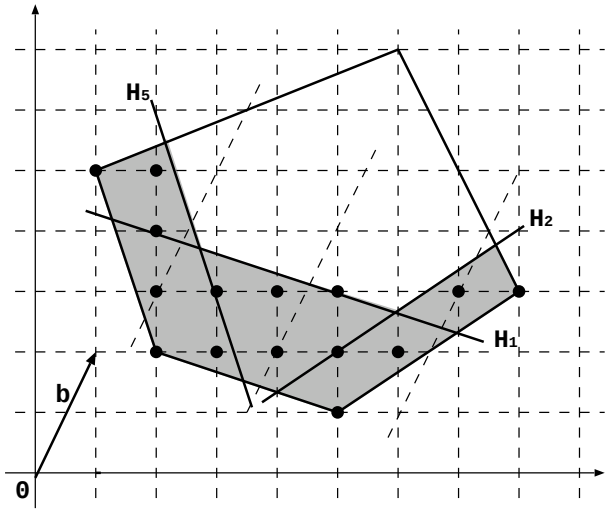


FIG. 5.4 – Image affine d'un $\mathbb{Z}$-polytope. On peut constater que dans la région grise, il ne peut y avoir qu'un seul point entier dans la direction de la projection (lignes en pointillé).

– C_1 : $w_1 = \langle \mathbf{a}_1, \mathbf{b} \rangle = 1 \times 1 + 3 \times 2 = 7 > 0$, la contrainte à rajouter est :

$$H_1 : \quad -x - 3y + 8 + 7 - 1 \geq 0 \quad \Rightarrow \quad -x - 3y + 14 \geq 0.$$

– C_2 : $w_2 = \langle \mathbf{a}_2, \mathbf{b} \rangle = -2 \times 1 + 3 \times 2 = 4 > 0$, la contrainte à rajouter est :

$$H_2 : \quad 2x - 3y - 7 + 4 - 1 \geq 0 \quad \Rightarrow \quad 2x - 3y - 4 \geq 0.$$

– C_3 : $w_3 = \langle \mathbf{a}_3, \mathbf{b} \rangle = -2 \times 1 + -1 \times 2 = -4 \leq 0$, pas de contrainte à rajouter.
– C_4 : $w_4 = \langle \mathbf{a}_4, \mathbf{b} \rangle = 2 \times 1 - 5 \times 2 = -8 \leq 0$, pas de contrainte à rajouter.
– C_5 : $w_5 = \langle \mathbf{a}_5, \mathbf{b} \rangle = 3 \times 1 + 1 \times 2 = 5 > 0$, la contrainte à rajouter est :

$$H_5 : \quad -3x - y + 8 + 5 - 1 \geq 0 \quad \Rightarrow \quad -3x - y + 12 \geq 0.$$

Lorsqu'une image affine d'un $\mathcal{Z}$-polytope se réduit à un problème de projection d'un $\mathcal{Z}$-polytope de dimension pleine d sur un espace de dimension $(d-1)$, il est possible d'utiliser la méthode des facettes ci-dessus pour la calculer. En pratique, il est utile de l'utiliser lorsque les coefficients de *l'unique* variable existentielle à éliminer sont grands. Ceci permet de réduire le temps d'exécution, puisque le nombre total des $\mathcal{Z}$-polytopes résultants de la projection est toujours inférieur au nombre de facettes du $\mathcal{Z}$-polytope à projeter (ce nombre est polynomial pour une dimension fixée). La matrice de transformation que nous utilisons pour calculer la projection entière d'un $\mathcal{Z}$-polytope de dimension pleine d sur un espace de dimension $(d-1)$ est de la forme : $A = \left(\begin{array}{c|c} 0 & 0 \\ \hline 0 & I_{(d-1)} \end{array}\right)$, où A est une matrice carrée $(d \times d)$ et $I_{(d-1)}$ est la matrice identité $(d-1) \times (d-1)$. Il est clair que $Ker(A) = A\mathbf{x} = \mathbf{0}$ est égal au sous-espace défini par : $x_i = 0$, avec $2 \leq i \leq d$, et dont le vecteur base est $\mathbf{b} = (1, 0, \cdots, 0)$. C'est ce vecteur

qui est utilisé pour calculer les différentes contraintes à intersecter avec le $\mathcal{Z}$-polytope à projeter.

À noter que contrairement à la méthode de la section 5.2, cette dernière méthode ne peut être appliquée à une union de $\mathcal{Z}$-polytopes dont les projections (entières) des différents $\mathcal{Z}$-polytopes ne sont pas disjointes. De plus, elle ne permet de traiter que des problèmes avec une seule variable existentielle.

5.4 Autres méthodes de calcul des images affines de $\mathbb{Z}$-polytopes

Dans cette section, nous décrivons quelques travaux récents concernant le calcul ou le dénombrement des images affines de $\mathbb{Z}$-polytopes.

5.4.1 Méthode de dénombrement des images affines de $\mathbb{Z}$-polytopes par calcul de minima lexicographiques

Contrairement à notre algorithme (section 5.2), la méthode que nous allons décrire dans cette section ne s'intéresse qu'au nombre de points entiers dans l'image affine d'un $\mathbb{Z}$-polytope. Ce qui lui permet de ne pas calculer la *vraie* projection d'un $\mathbb{Z}$-polytope, mais un ensemble équivalent ayant le même nombre de points entiers, comme c'est le cas pour la méthode de facettes (section 5.3). Lorsqu'on a besoin de transformer non pas un $\mathbb{Z}$-polytope unique mais une *union* de $\mathbb{Z}$-polytopes, cette union doit être séparée en un ensemble de $\mathbb{Z}$-polytopes dont les transformations correspondantes sont disjointes, en utilisant Omega [77] ou la `Polylib` [95] par exemple. À noter que le calcul de cette union disjointe est exponentiel dans le pire des cas.

Boulet et Redon [28] ont proposé d'utiliser PIP (*Parametric Integer Programming*) [54] pour dénombrer les points entiers de l'image affine d'un $\mathbb{Z}$-polytope paramétré. PIP permet de calculer le minimum lexicographique d'un $\mathbb{Z}$-polytope paramétré sous forme d'expressions affines rationnelles. Chaque expression est une fonction des paramètres originaux et éventuellement d'un certain nombre de paramètres supplémentaires. Ces derniers paramètres sont en réalité des parties entières inférieures d'expressions rationnelles des autres paramètres. Dans ce qui suit nous montrons comment est utilisé PIP pour calculer le nombre de points entiers de l'image affine d'un $\mathbb{Z}$-polytope paramétré.

Considérons la formule de Presburger suivante :

$$S = \{\mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{x}' \in \mathbb{Z}^{d'} : A\mathbf{x}' + B\mathbf{x} + C\mathbf{p} + \mathbf{c} \geq 0\}, \tag{5.23}$$

où $\mathbf{x}'$ est un vecteur de d' variables existentielles, $\mathbf{x}$ est un vecteur de d variables libres, et $\mathbf{p}$ est un vecteur de n paramètres.

Afin d'éliminer les variables existentielles de la formule S, Boulet et Redon [28] commencent par calculer le minimum lexicographique : $\mathbf{x}'_{min} = f(\mathbf{x}, \mathbf{p}, \mathbf{p}')$ des variables existentielles $\mathbf{x}'$, où les variables libres et les paramètres originaux sont considérés comme des paramètres. La solution est donnée en fonction des variables libres $\mathbf{x}$, des paramètres originaux $\mathbf{p}$ et des nouveaux paramètres $\mathbf{p}'$. En remplaçant $\mathbf{x}'$ par $\mathbf{x}'_{min}$ dans (5.23), le nombre de points entiers dans l'ensemble S ne change pas. Les nouveaux paramètres peuvent être traités comme des variables existentielles *uniques* [144, 142]. Ce sont des

variables existentielles qui peuvent être considérées comme des variables libres sans affecter le nombre de points entiers de la formule originale. Le nombre de points dans l'ensemble résultant est ensuite calculé en utilisant la méthode de Clauss/Loechner [39] ou notre méthode décrite dans le chapitre 3.

Exemple 30. Considérons l'ensemble suivant [142] :

$$S = \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8\}. \qquad (5.24)$$

La solution de : minlex$\{\exists y \in \mathbb{Z} \mid -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8\}$ est

$$y_{min}(x,p) = \begin{cases} 1 - x - \left\lfloor \frac{2-2x}{3} \right\rfloor & \text{si } x + 2p + 2 \geq 0 \\ -x - \left\lfloor \frac{p-x}{2} \right\rfloor & \text{sinon} \end{cases}.$$

L'ensemble S peut ensuite être écrit sous forme d'une union disjointe de deux ensembles S_1 et S_2 (en substituant $y = y_{min}(x,p)$ dans S). L'ensemble S_1 est obtenu en substituant $y = 1-x-q$ dans (5.24), où $q = \lfloor (2-2x)/3 \rfloor$ est un nouveau paramètre. Le résultat est :

$$S_1 = \{x \in \mathbb{Z} \mid \exists y, q \in \mathbb{Z}^2 : y = 1 - x - q \wedge 2 - 2x \leq 3q \leq 4 - 2x \wedge x + 3p + 2 \geq 0 \wedge \\ - x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8\}.$$

De la même façon, on peut calculer S_2 sous la forme :

$$S_1 = \{x \in \mathbb{Z} \mid \exists y, q' \in \mathbb{Z}^2 : y = x - q' \wedge p - x \leq 2q' \leq 1 + p - x \wedge x + 3p + 3 \leq 0 \wedge \\ - x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8\},$$

avec $q = \lfloor (p-x)/2 \rfloor$.

La méthode de dénombrement des projections entières par le calcul des minima lexicographiques peut toujours être appliquée, mais elle est aussi exponentielle dans le pire des cas (même pour une dimension fixée). Contrairement à notre méthode de la section 5.2, l'algorithme de Boulet et Redon ne calcule pas la vraie projection, mais des ensembles équivalents de dimensions souvent supérieures à la dimension de la projection effective. Ceci est dû aux paramètres supplémentaires résultants du calcul des minima lexicographiques. Le comptage des points entiers dans de tels ensembles est souvent plus coûteux que celui de la transformation elle-même que nous calculons, en particulier lorsque le nombre de $\mathbb{Z}$-polytopes que nous obtenons est relativement petit.

5.4.2 Règles de simplification de Verdoolaege

Verdoolaege et al. [144, 142] ont proposé de combiner PIP avec un ensemble de règles de simplification polynomiales pour améliorer le temps dénombrement des transformations. Leur idée consiste à ne pas utiliser directement PIP, mais à essayer d'abord d'appliquer un ensemble de règles simples correspondant à des cas particuliers de transformations. Lorsqu'aucune de ces règles de simplification ne peut être appliquée, ils font appel à PIP, comme dans [28], pour éliminer le reste des variables existentielles. Dans ce qui suit, nous résumons leurs règles de simplification, telles qu'elles sont décrites dans [144, 142].

Les variables existentielles uniques : les quantificateurs existentiels peuvent parfois être redondants. Ceci se produit lorsque pour chaque valeur de $\mathbf{x}$ (vecteur des variables libres), il existe au maximum une seule valeur d'une variable existentielle x'_i qui vérifie les contraintes. Dans ce cas, le quantificateur de x'_i peut être supprimé, i.e., x'_i peut être considéré comme une variable libre sans affecter la cardinalité de l'ensemble. Ceci peut se produire lorsqu'il y a une contrainte n'impliquant qu'une seule variable existentielle x'_i. Supposons que cette contrainte correspond à une borne supérieure sur x'_i[3]. Considérons cette borne inférieure

$$n_l x'_i + \langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l \geq 0$$

et une borne supérieure sur x'_i de la forme :

$$-n_u x'_i + \langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \overline{\mathbf{d}}_u, \overline{\mathbf{x}'} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u \geq 0,$$

où $\overline{\mathbf{x}'}$ correspond au reste des variables existentielles (sans x'_i), et n_l et n_u sont des constantes entières strictement positives. La variable existentielle x'_i est *unique* si la contrainte suivante est vérifiée.

$$n_l(\langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \overline{\mathbf{d}}_u, \overline{\mathbf{x}'} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u) + n_u(\langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l) + 1 \leq n_l n_u. \quad (5.25)$$

Les variables existentielles redondantes : dans certains cas, il peut exister au moins une valeur entière de la variable à éliminer x'_i pour toute valeur entière des autres variables existentielles et des variables libres. Dans ce cas, la variable existentielle peut être éliminée comme dans le cas rationnel, puisque tout point entier dans la projection rationnelle possède au moins un antécédent entier.

Considérons une paire de bornes, inférieure et supérieure, sur la variable existentielle x'_i : $\{n_l x'_i + l(\overline{\mathbf{x}'}, \mathbf{x}, \mathbf{p}) \geq 0, \ -n_u x'_i + u(\overline{\mathbf{x}'}, \mathbf{x}, \mathbf{p}) \geq 0\}$, où $l(\overline{\mathbf{x}'}, \mathbf{x}, \mathbf{p})$ et $u(\overline{\mathbf{x}'}, \mathbf{x}, \mathbf{p})$ sont des fonctions affines du reste des variables existentielles $\overline{\mathbf{x}'}$, des variables libres $\mathbf{x}$ et des paramètres $\mathbf{p}$. La variable existentielle x'_i est redondante si la contrainte suivante est vérifiée.

$$n_l(u(\overline{\mathbf{x}'}, \mathbf{x}, \mathbf{p})) + n_u(l(\overline{\mathbf{x}'}, \mathbf{x}, \mathbf{p})) + n_l + n_u - 1 \geq n_l n_u. \quad (5.26)$$

Noter que ce cas se produit uniquement si l'ombre noire est égale à l'ombre exacte, il est donc implicitement pris en compte par notre méthode de la section 5.2.

Découpages indépendants (*independent splits*) : lorsqu'aucune des deux règles de simplification vues ci-dessus ne peut être appliquée, Verdoolaege et al. proposent de découper le $\mathbb{Z}$-polytope à projeter, en l'intersectant avec des hyperplans indépendants de la variable existentielle à éliminer, pour obtenir (lorsque c'est possible) des $\mathbb{Z}$-polytopes dont les contraintes vérifient ou bien la condition (5.25), ou bien la condition (5.26). Lorsqu'il n'y a qu'une seule variable existentielle, ce découpage est toujours possible. Par exemple, le $\mathbb{Z}$-polytope de la figure 5.5 peut être décomposé en trois $\mathbb{Z}$-polytopes : le $\mathbb{Z}$-polytope du milieu vérifie la contrainte (5.26), et les deux $\mathbb{Z}$-polytopes situés sur les bords vérifient la contrainte (5.25).

[3]Le raisonnement reste le même si c'est la borne supérieure qui n'implique qu'une seule variable existentielle.

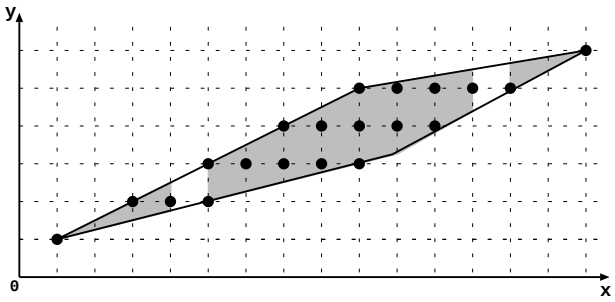


FIG. 5.5 – Décomposition d'un $\mathbb{Z}$-polytope de façon à ce que la variable existentielle y soit unique ou redondante.

5.4.3 Méthode analytique de calcul de l'image affine d'un $\mathbb{Z}$-polytope

Barvinok et Woods [13] ont proposé une méthode *analytique* qui permet de calculer la transformation, par une fonction affine, d'un $\mathbb{Z}$-polytope. En se basant sur les idées de l'algorithme de dénombrement analytique de polytopes paramétrés (chapitre 3), Verdoolaege et Woods[149] ont démontré que cette méthode de transformation peut être généralisée pour traiter des $\mathbb{Z}$-polytopes paramétrés quelconques. La méthode originale de Barvinok et Woods ainsi que sa généralisation au cas paramétré sont polynomiales en la taille des contraintes du $\mathbb{Z}$-polytope et les coefficients de la transformation (pour une dimension fixée). Cependant, ces méthodes de transformations sont trop théoriques et leur implémentation reste, au jour d'aujourd'hui, un vrai challenge. De plus, dans plusieurs cas, les méthodes de transformation vues jusque-là pourront faire beaucoup mieux que ces dernières méthodes.

Pour donner une idée sur la méthode de transformation analytique, nous considérons un cas simple correspondant à la projection $\pi(P \cap \mathbb{Z}^d) : \mathbb{Z}^d \rightarrow \mathbb{Z}^{d-2}$ d'un $\mathbb{Z}$-polytope standard non paramétré de dimension d. Afin de calculer cette projection, on peut procéder comme suit.

La projection à travers la première dimension, i.e., $\pi(P \cap \mathbb{Z}^d) : \mathbb{Z}^d \rightarrow \mathbb{Z}^{d-1}$ est donnée par la différence entre la fonction génératrice de $P \cap \mathbb{Z}^d$ et celle de $P' \cap \mathbb{Z}^d$, où P' est la translation de P par le vecteur $(0, \cdots, 0, 1)$, et où le résultat est projeté sur l'espace $\mathbb{Z}^{d-1}$ en appliquant la substitution $x_d = 1$. Autrement dit :

- calculer $f(S; \mathbf{x})$, avec $S = P \cap \mathbb{Z}^d$ et $\mathbf{x} = x_1, x_2, \cdots, x_d$,
- calculer $f(S'; \mathbf{x}) = f(S; \mathbf{x}) \setminus x_d f(S; \mathbf{x})$,
- calculer $f(\hat{S}_1, \mathbf{y}) = \{f(S'; \mathbf{x}) \wedge x_d = 1\}$, avec $\mathbf{y} = x_1, x_2, \cdots, x_{d-1}$.

Ensuite, la projection à travers la deuxième dimension, i.e., $\pi(\hat{S}_1) : \mathbb{Z}^{d-1} \rightarrow \mathbb{Z}^{d-2}$ est calculée comme suit :

- calculer $f(\hat{S_1}'; \mathbf{y}) = f(\hat{S}_1; \mathbf{y}) \setminus \bigcup_{i=1}^{\sigma} x_{d-1}^i f(\hat{S}_1; \mathbf{y})$, où σ est la distance maximale entre deux points entiers adjacents (dans la direction de x_{d-1}),
- calculer $f(\hat{S}_2, \mathbf{z}) = \{f(\hat{S_1}'; \mathbf{y}) \wedge x_{d-1} = 1\}$, avec $\mathbf{z} = x_1, x_2, \cdots, x_{d-2}$.

En regardant la procédure de projection ci-dessus, on peut constater que pour calculer la projection d'un $\mathbb{Z}$-polytope $P \cap \mathbb{Z}^d$ on doit pouvoir calculer : (i) la fonction génératrice de P, (ii) les opérations (union et différence) sur des fonctions génératrices.

Le calcul de la fonction génératrice d'un polytope est décrit dans la chapitre 3. Les opérations sur les fonctions génératrices quant à elles sont calculées comme suit :

Soit $f(S_1;\mathbf{x}) = \sum_{i\in I} \alpha_i g_{1i}(\mathbf{x})$, $f(S_2;\mathbf{x}) = \sum_{j\in J} \beta_j g_{2j}(\mathbf{x})$. L'union, la différence et l'intersection de $f(S_1;\mathbf{x})$ et $f(S_2;\mathbf{x})$ sont données respectivement par :

- $f(S_1;\mathbf{x}) \cup f(S_2;\mathbf{x}) = f(S_1;\mathbf{x}) + f(S_2;\mathbf{x}) - f(S_1 \cap S_2;\mathbf{x})$,
- $f(S_1;\mathbf{x}) \setminus f(S_2;\mathbf{x}) = f(S_1;\mathbf{x}) - f(S_1 \cap S_2;\mathbf{x})$,
- $f(S_1 \cap S_2;\mathbf{x}) = f(S_1;\mathbf{x}) \star f(S_2;\mathbf{x}) = \sum_{i\in I, j\in J} \alpha_i\beta_j g_{1i}(\mathbf{x}) \star g_{2j}(\mathbf{x})$, où $\star$ est le produit de Hadamard défini par :

$$\sum_{m\in\mathbb{Z}^d} \beta_{1m}\mathbf{x}^m \star \sum_{m\in\mathbb{Z}^d} \beta_{2m}\mathbf{x}^m = \sum_{m\in\mathbb{Z}^d} \beta_{1m}\beta_{2m}\mathbf{x}^m.$$

  Par exemple $(xy + xy^2 + x^2y^3) \star (x^2y + xy^2) = xy^2$.

Il est donc clair que pour réaliser les opérations sur les fonctions génératrices, il est indispensable de pouvoir calculer le produit de Hadamard. Barvinok et Woods [13] ont proposé un algorithme qui permet de calculer le produit de Hadamard de deux fonctions génératrices en un temps polynomial (pour une dimension fixée). Le principe de cet algorithme est le suivant :

Soient $g_1(\mathbf{x}) = \frac{\mathbf{x}^{\mathbf{p}_1}}{(1-\mathbf{x}^{\mathbf{a}_{11}})\cdots(1-\mathbf{x}^{\mathbf{a}_{1k}})}$ et $g_2(\mathbf{x}) = \frac{\mathbf{x}^{\mathbf{p}_2}}{(1-\mathbf{x}^{\mathbf{a}_{21}})\cdots(1-\mathbf{x}^{\mathbf{a}_{2k}})}$ deux termes de fonctions génératrices. Le produit de Hadamard $g_1(\mathbf{x}) \star g_2(\mathbf{x})$ est calculé en trois étapes :

- construire un polyèdre $P \in \mathbb{Z}^{2k}$

$$P = \begin{cases} \mathbf{p}_1 + \mathbf{z}_1\mathbf{a}_{11} + \cdots + \mathbf{z}_k\mathbf{a}_{1k} = \mathbf{p}_2 + \mathbf{z}_{k+1}\mathbf{a}_{21} + \cdots + \mathbf{z}_{2k}\mathbf{a}_{2k} \\ \mathbf{z}_i \geq 0 \text{ pour } 1 \leq i \leq 2k, \end{cases}$$

- calculer $\mathbf{x}^{\mathbf{p}_1} f(P \cap \mathbb{Z}^{2k};\mathbf{z})$,
- calculer la substitution $z_1 = \mathbf{x}^{\mathbf{a}_{11}}, \cdots, z_k = \mathbf{x}^{\mathbf{a}_{1k}}$, $z_{k+1} = 1, \cdots, z_{2k} = 1$.

Pour conclure, nous pensons que même si la complexité de cette méthode est *théoriquement* polynomiale (pour une dimension fixée), elle peut en pratique être plus coûteuse que notre méthode ou celle de Verdoolaege [144] (pour les transformations qui ne sont pas exponentielles). Ceci est principalement dû au calcul du produit de Hadamard qui se traduit par le calcul de fonctions génératrices de nouveaux polytopes. En effet, nous avons implémenté une partie de cette méthode qui permet de calculer un cas simple correspondant à la projection d'un cube 3D sur l'espace 2D. Nous avons utilisé cette implémentation pour calculer le nombre de points entiers dans la projection, sur (x, y) du cube $\{1 \leq x \leq 10 \wedge 1 \leq y \leq 5 \wedge 1 \leq z \leq 5\}$. La fonction génératrice de ce cube et celle de sa translation par le vecteur $(0, 0, 1)$ sont constituées de 8 termes chacune. Le produit de Hadamard des deux fonctions génératrices est donné par 72 nouvelles fonctions génératrices dont l'évaluation résulte en le nombre de points entiers dans la projection du cube. Le temps de ce calcul est $0.07s$, Alors que le temps de calcul en utilisant les autres méthodes est négligeable ($< 0.01s$), car la projection de ce cube ne comporte aucun trou.

5.5 Expériences

Dans cette section, nous comparons notre implémentation avec celle de Verdoolaege et al. [144], puisqu'elle est la plus récente à être entièrement implémentée et plus efficace

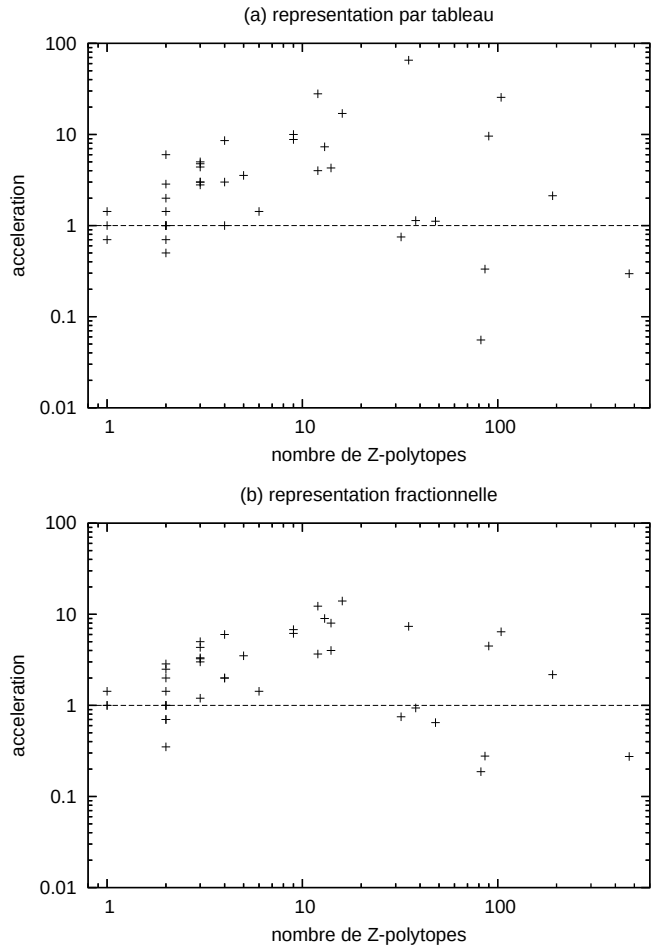


FIG. 5.6 – Comparaison du temps d'exécution avec l'implémentation de Verdoolaege.

comparée avec d'autres méthodes existantes [144]. Ces expériences ont été effectuées avec la version 5.22 de la librairie `Polylib` et la version 0.20 de la librairie `barvinok`. Dans les deux implémentations, la `Polylib` est utilisée pour réaliser les opérations polyédriques, et la librairie `barvinok` (implémentation de l'algorithme décrit dans le chapitre 3) est utilisée pour compter les points entiers de polytopes paramétrés. De plus, Verdoolaege et al. utilisent `PIP` [57] pour calculer les minima lexicographiques, et optionnellement `Omega` [77] pour simplifier les $\mathbb{Z}$-polytopes à projeter.

L'ensemble de $\mathbb{Z}$-polytopes avec lequel sont effectuées ces expériences est représentatif d'un nombre important de cas (qui ne se réduisent pas à la transformation rationnelle qu'on peut calculer facilement par les deux méthodes). La dimension des $\mathbb{Z}$-polytopes varie entre 3 et 7, le nombre de paramètres est compris entre 1 et 3, et le nombre de variables existentielles est compris entre 1 et 5. Nous avons choisi de comparer l'accélération (figure 5.6) et le ratio de la taille de la solution (figure 5.7) des deux méthodes, en fonction du nombre de $\mathbb{Z}$-polytopes générés par notre méthode, puisqu'il est le paramètre qui influe le plus sur sa complexité. La comparaison est effectuée en utilisant

les deux représentations de quasi-polynômes d'Ehrhart : la représentation sous forme de tableaux et la représentation fractionnelle.

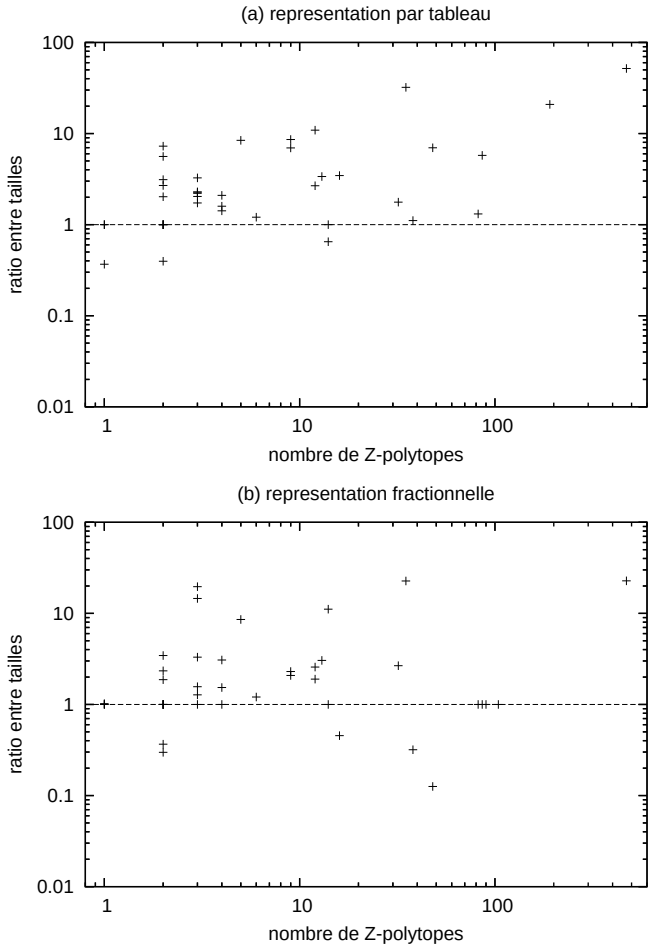


FIG. 5.7 – Comparaison de la taille de la solution avec l'implémentation de Verdoolaege.

La représentation sous forme de tableaux est exponentielle, dans le pire des cas, [148, 147]. Mais son avantage par rapport à la représentation fractionnelle est qu'elle peut être simplifiée facilement. Ce qui fait que, lorsque les périodes sont petites, les quasi-polynômes, où les nombres périodiques sont représentés par des tableaux, peuvent être de taille inférieure à celle de la représentation fractionnelle.

La figure 5.6 montre que pour la plupart des $\mathbb{Z}$-polytopes de ces expériences, nos temps d'exécution sont significativement inférieurs à ceux de Verdoolaege (l'échelle est logarithmique). L'accélération moyenne est de 2.05 dans le cas de la représentation fractionnelle, et 2.71 en représentant les nombres périodiques par des tableaux. En effet, la plupart des quasi-polynômes générés par la méthode de Verdoolaege ont des périodes plus grandes que ceux de notre méthode. C'est pourquoi lorsqu'on utilise la représentation fractionnelle, les performances de notre méthode restent pratiquement les mêmes, alors que l'algorithme de Verdoolaege devient plus rapide (pour certains

exemples). Noter que la méthode de Verdoolaege ne calcule pas la vraie projection, mais un ensemble équivalent de $\mathbb{Z}$-polytopes ayant le même nombre de points entiers que la projection effective. Ces $\mathbb{Z}$-polytopes sont parfois de dimension plus grande que celle de la projection elle-même. Il peuvent aussi avoir des coefficients de contraintes (périodes) plus grands, ce qui peut augmenter le temps d'exécution.

La figure 5.7 montre le ratio de la taille de la solution (quasi-polynômes). Encore une fois, puisque notre méthode génère des périodes plus petites, les quasi-polynômes qui en résultent sont de taille plus petite que ceux de Verdoolaege. Les ratios moyens de la taille de la solution sont 1.94 dans le cas de la représentation fractionnelle, et 3.23 lorsque les nombres périodiques sont représentés par des tableaux.

Chapitre 6

Applications

Dans ce chapitre, nous décrivons quelques applications des algorithmes de comptage de points entiers dans des polytopes paramétrés et dans leurs images affines. Tout au long de cette description, nous ne nous sommes focalisés que sur les applications issues du domaine de l'optimisation et de la parallélisation de programmes.

6.1 Linéarisation de tableaux et localité spatiale

Dans [97], Loechner et al. présentent une méthode d'optimisation de la localité spatiale à travers la linéarisation de tableaux. L'idée est de transformer un tableau de dimension quelconque en un tableau unidimensionnel (linéaire). Les éléments du nouveau tableau sont rangés dans l'ordre dans lequel ils sont accédés pour la première fois par le programme. Dans le cas des programmes "affines", cette linéarisation revient à calculer des polynômes d'Ehrhart correspondant à des images affines de $\mathbb{Z}$-polytopes paramétrés.

Soit A un tableau de dimension k référencé, à travers une fonction d'accès affine T, par un nid de boucles affine de profondeur d, et soit $\mathbf{x}_0 \in \mathbb{Z}^k$ un des éléments de ce tableau. L'adresse correspondante de $\mathbf{x}_0$ dans le nouveau tableau unidimensionnel B est obtenue en calculant :

1. la *première* itération $\mathbf{i}_{min}(\mathbf{x_0}, \mathbf{p})$ référençant l'élément $\mathbf{x_0}$, où $\mathbf{p}$ est un vecteur de paramètres du nid de boucles. Cette itération est le minimum lexicographique de l'ensemble des itérations $I(\mathbf{x_0})$ référençant l'élément $\mathbf{x_0}$. $I(\mathbf{x_0}) = \{\mathbf{i} \in (P \cap \mathbb{Z}^d) \mid T(\mathbf{i}) = \mathbf{x_0}\}$, où P est le polytope de dimension d dont les points entiers correspondent aux itérations du nid de boucles. $\mathbf{i}_{min}(\mathbf{x_0}, \mathbf{p})$ est calculé à partir de $I(\mathbf{x_0})$ en utilisant PIP [54] ;
2. le nombre des éléments du tableau A accédés par les itérations du nid de boucles qui sont lexicographiquement inférieures à l'itération $\mathbf{i}_{min}(\mathbf{x_0}, \mathbf{p})$. Ce nombre est égal au nombre des images, par la fonction T, des itérations lexicographiquement inférieures à $\mathbf{i}_{min}(\mathbf{x_0}, \mathbf{p})$.

En général, lorsqu'il y a deux références $\mathcal{R}_1$ et $\mathcal{R}_2$ à un même tableau dans un nid de boucles, les données (éléments du tableau) qui sont accédées par les deux fonctions de référence ne peuvent pas être optimisées pour les deux références. Dans [97], la méthode proposée consiste à choisir une des références à optimiser pour ces données,

et l'autre référence ne sera pas optimisée. Supposons qu'on ait choisi d'optimiser la référence $\mathcal{R}_1$, alors toutes les données accédées par $\mathcal{R}_1$ seront organisées dans le nouveau tableau B comme expliqué ci-dessus. Les données accédées par les deux références $\mathcal{R}_1$ et $\mathcal{R}_2$ utilisent aussi la même fonction de référence, non optimisée pour $\mathcal{R}_2$. Les données accédées uniquement par $\mathcal{R}_2$ peuvent ensuite être optimisées, indépendemment de celles accédées par $\mathcal{R}_1$, à travers une autre fonction. La génération de code pour cette solution consiste soit à ajouter des tests vérifiant si une donnée accédée par $\mathcal{R}_2$ est aussi accédée par $\mathcal{R}_1$ (pour décider quelle fonction de référence utiliser), soit à découper le nid de boucles, de façon à séparer les itérations de $\mathcal{R}_2$ accédant au tableau à travers la première fonction de celles y accédant à travers la deuxième. Autrement dit, il faut séparer les itérations accédant aux éléments du tableau référencés à la fois par $\mathcal{R}_1$ et $\mathcal{R}_2$ de celles accédant aux éléments référencés par $\mathcal{R}_2$ seulement.

Lorsque deux nids de boucles accèdent aux mêmes données, la solution proposée est similaire à celle de deux références au sein d'un même nid de boucles. Les données accédées par les deux nids de boucles sont optimisées par rapport à un premier nid de boucles et l'autre nid de boucles est découpé en deux parties : une partie (non optimisée) qui utilise la même fonction de référence que le premier nid de boucles, et une deuxième partie (optimisée) qui utilise une nouvelle fonction de référence correspondant aux données accédées seulement par le deuxième nid de boucles. Cette méthode peut aussi être appliquée à plusieurs références et nids de boucles [97].

Dans ce qui suit, nous proposons une amélioration de la méthode originale de Loechner et al. [97], en tirant avantage des nouveaux algorithmes de comptage des points entiers dans des polytopes paramétrés et dans leurs images affines (chapitres 3,4, et 5). La nouvelle technique a l'avantage de traiter les cas particuliers, pour lesquels la méthode originale échoue ou résulte en un replacement de données moins efficace.

6.1.1 Cas de deux références dans un même nid de boucles

Considérons deux références $\mathcal{R}_1$ et $\mathcal{R}_2$ à un même tableau A, qui apparaissent au même niveau d'un nid de boucles, et où $\mathcal{R}_1$ précède $\mathcal{R}_2$. Nous avons besoin de définir deux fonctions d'accès $\mathcal{E}_1(\mathbf{x_0}, \mathbf{p})$ et $\mathcal{E}_2(\mathbf{x_0}, \mathbf{p})$, optimisant respectivement les références $\mathcal{R}_1$ et $\mathcal{R}_2$, où $\mathbf{x_0}$ est un vecteur d'indices d'un élément accédé du tableau, et $\mathbf{p}$ est un vecteur de paramètres.

Soit P l'espace d'itération du nid de boucles, et soient $\mathcal{R}_1 = A(T_1(\mathbf{i}))$ et $\mathcal{R}_2 = A(T_2(\mathbf{i}))$ deux références au tableau A, où T_1 et T_2 sont des fonctions affines et $\mathbf{i}$ est un vecteur de l'espace d'itération P. La première itération accédant à la donnée $A(\mathbf{x_0})$ par rapport à la référence $\mathcal{R}_1$, (resp. $\mathcal{R}_2$) est donnée par le minimum lexicographique de l'ensemble $T_1^{-1}(\mathbf{x_0}) \cap P$ (resp. $T_2^{-1}(\mathbf{x_0}) \cap P$). Soient $\mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})$ et $\mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})$ ces minima lexicographiques, et soient D_1 et D_2 leurs domaines de validité respectifs[1]. Trois cas sont possibles :

1. Lorsque $A(\mathbf{x_0})$ est accédé seulement par $\mathcal{R}_1$, i.e., $\mathbf{x_0} \in D_1 \setminus D_2$, la nouvelle fonction d'accès pour la référence $\mathcal{R}_1$ est égale au nombre des éléments du tableau accédés par les deux références $\mathcal{R}_1$ et $\mathcal{R}_2$, avant l'itération $\mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})$. Ces éléments sont donnés par l'union des images par T_1 et T_2 des itérations qui sont

[1] Les contraintes sur $\mathbf{x_0}$ et $\mathbf{p}$ pour lesquelles le minimum lexicographique $\mathbf{i}_{minj}(\mathbf{x_0}, \mathbf{p})$ existe.

lexicographiquement inférieures à $\mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})$.

$$\mathcal{E}_1(\mathbf{x_0}, \mathbf{p}) = \#(\{T_1(\mathbf{i}) \mid \mathbf{i} \in P \wedge\ \mathbf{i} \prec \mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})\} \cup \{T_2(\mathbf{i}) \mid \mathbf{i} \in P \wedge\ \mathbf{i} \prec \mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})\}).$$

2. Lorsque $A(\mathbf{x_0})$ est accédé seulement par $\mathcal{R}_2$, i.e., $\mathbf{x_0} \in D_2 \backslash D_1$, la nouvelle fonction d'accès pour $\mathcal{R}_2$ est égale à l'union des éléments du tableau qui sont accédés par par $\mathcal{R}_2$ avant l'itération $\mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})$ et des éléments qui sont accédés par $\mathcal{R}_1$ avant l'itération $\mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})$ (incluse).

$$\mathcal{E}_2(\mathbf{x_0}, \mathbf{p}) = \#(\{T_1(\mathbf{i}) \mid \mathbf{i} \in P \wedge\ \mathbf{i} \preceq \mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})\} \cup \{T_2(\mathbf{i}) \mid \mathbf{i} \in P \wedge\ \mathbf{i} \prec \mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})\}).$$

3. Enfin, lorsque $A(\mathbf{x_0})$ est accédé par les deux références $\mathcal{R}_1$ et $\mathcal{R}_2$, i.e., $\mathbf{x_0} \in D_1 \cap D_2$, la nouvelle fonction d'accès $\mathcal{E}(\mathbf{x_0}, \mathbf{p})$, pour $\mathcal{R}_1$ et $\mathcal{R}_2$, est donnée par le nombre des éléments du tableau accédés, par les deux références, avant le minimum des itération $\mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})$ et $\mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})$.

$$\begin{aligned}\mathcal{E}(\mathbf{x_0}, \mathbf{p}) = \#(\{T_1(\mathbf{i}) \mid \mathbf{i} \in P\ \wedge\ \mathbf{i} \prec \mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})\ \wedge\ \mathbf{i} \preceq \mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})\} \cup \\ \{T_2(\mathbf{i}) \mid \mathbf{i} \in P\ \wedge\ \mathbf{i} \prec \mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})\ \wedge\ \mathbf{i} \prec \mathbf{i}_{min2}(\mathbf{x}_0)\}).\end{aligned}$$

Ainsi, la nouvelle fonction d'accès respecte exactement l'ordre d'accès aux éléments du tableau par le nid de boucles.

6.1.2 Cas de deux références dans des nids de boucles différents

Nous considérons maintenant deux références $\mathcal{R}_1$ et $\mathcal{R}_2$, au même tableau A, qui apparaissent dans des nids de boucles différents d'espaces d'itération P_1 et P_2, telles que $\mathcal{R}_1$ précède $\mathcal{R}_2$. Considérons à nouveau l'élément $A(\mathbf{x_0})$ accédé, pour la première fois, à l'itération $\mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p}) \in P_1$ par la référence $\mathcal{R}_1$ (resp. à l'itération $\mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p}) \in P_2$ par $\mathcal{R}_2$). La référence $\mathcal{R}_1$ est optimisée indépendamment de $\mathcal{R}_2$, puisqu'elle est la première à accéder à tous les éléments du tableau auxquels elle fait référence. En revanche, la référence $\mathcal{R}_2$ est optimisée de deux façons différentes :

1. Lorsque $A(\mathbf{x_0})$ est aussi accédé par $\mathcal{R}_1$, i.e., $\mathbf{x_0} \in D_1 \cap D_2$, la nouvelle fonction d'accès pour $\mathcal{R}_2$ est égale à celle de $\mathcal{R}_1$, qui est donnée par le nombre de tous les éléments du tableau qui sont accédés par $\mathcal{R}_1$ avant l'itération $\mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})$.

$$\mathcal{E}_2(\mathbf{x_0}, \mathbf{p}) = \mathcal{E}_1(\mathbf{x_0}, \mathbf{p}) = \#\{T_1(\mathbf{i}) \mid \mathbf{i} \in P_1\ \wedge\ \mathbf{i} \prec \mathbf{i}_{min1}(\mathbf{x_0}, \mathbf{p})\}.$$

2. Lorsque $A(\mathbf{x_0})$ est accédée seulement par $\mathcal{R}_2$, i.e., $\mathbf{x_0} \in D_2 \setminus D_1$, la nouvelle fonction d'accès $\mathcal{E}_2(\mathbf{x_0}, \mathbf{p})$ pour $\mathcal{R}_2$ est donnée par l'union de tous les éléments du tableau A qui sont accédés par $\mathcal{R}_1$ et des éléments qui sont accédés par $\mathcal{R}_2$ avant l'itération $\mathbf{i}_{min2}(\mathbf{x_0})$.

$$\mathcal{E}_2(\mathbf{x_0}, \mathbf{p}) = \#(\{T_1(\mathbf{i}) \mid \mathbf{i} \in P_1\} \cup \{T_2(\mathbf{i}) \mid \mathbf{i} \in P_2\ \wedge\ \mathbf{i} \prec \mathbf{i}_{min2}(\mathbf{x_0}, \mathbf{p})\}).$$

Ce principe de linéarisation est applicable à un nombre quelconque de références qui apparaissent éventuellement dans plusieurs nids de boucles. Mais dans ce cas, les fonctions de référence résultantes pourront être complexes.

```
do i=1, N
  do j=1, N
    do k=1, N
      A(3*i+6*k, 2*j+5)= ...
    enddo
  enddo
enddo
```

(a) nid de boucles original

```
do i=1, N
  do j=1, N
    do k=1, N
      if (i+2*k>=2*N+3)
        B((i+2*k-3)*N+j-1)=...
      else if (i+2*k<=2*N+1 & i mod 2 =1)
        B((j-1)*N+(i+2k-3)/2)=...
      else B((N+j-1)*N+i/2+k-2)=...
    enddo
  enddo
enddo
```

(b) nid de boucles transformé

FIG. 6.1 – Linéarisation d'un tableau : le tableau A dans le nid de boucles (a) est transformé en un tableau unidimensionnel B dans le nid de boucles (b).

L'avantage de la méthode de linéarisation utilisant les images affines de $\mathbb{Z}$-polytopes est qu'elle permet de traiter le cas général, pour lequel la méthode originale [97] échoue. Il s'agit du cas où il y a des trous dans l'espace de données accédé par un nid de boucles. i.e., lorsque la fonction de référence n'est pas unimodulaire. À titre d'exemple, la fonction $T(i,j) = 2i + 4j$, avec $i, j \in \{1, 2\}$ accède seulement aux éléments pairs compris entre 6 and 12. Les éléments impairs compris entre 7 et 11 sont donc des trous. Ce problème est résolu puisque nous calculons l'image affine exactes sous forme d'unions de $\mathbb{Z}$-polytopes, ce qui n'est pas le cas de la méthode originale.

Lorsque deux références à un même tableau apparaissent dans un nid de boucles, l'ancienne méthode [97] ne permet pas d'optimiser les deux références lorsqu'elles accèdent aux mêmes données. De plus, cette méthode optimise chaque référence indépendemment des autres, i.e., elle ignore le fait que les données sont accédées alternativement par les deux références, ce qui ne respecte pas le replacement dans l'ordre *strict* d'accès aux données. Ceci est aussi résolu en utilisant les images affines exactes de $\mathbb{Z}$-polytopes. Enfin, puisqu'il est difficile de séparer l'espace des itérations en régions convexes, où chaque région est accédée seulement par certaines références, nous remplaçons le découpage de boucles (*loop splitting*) par des tests vérifiant si une donnée (élément d'un tableau) possède un minimum lexicographique par rapport à chaque référence.

6.1.3 Exemple illustratif

Dans cet exemple, nous allons linéariser le tableau A du nid de boucles de la figure 6.1.(a). Pour montrer la généralité de notre méthode, nous avons considéré un exemple complexe où l'espace de données accédé comporte des trous. Les points entiers du polytope $P = \{(i,j,k) \in \mathbb{Z}^3 \mid 1 \leq i \leq N \ \wedge 1 \leq j \leq N \ \wedge 1 \leq k \leq N\}$ correspondent aux itérations du nid de boucles, et l'image de ces points par la fonction d'accès affine $T(i,j,k) = (x = 3i + 6k, y = 2j + 5)$ résulte en les éléments du tableau A accédés par ce nid de boucles. Les points entiers de l'image *rationnelle* de P par T, qui n'ont pas d'antécédents entiers dans P sont des trous, i.e., ils ne sont pas accédés par le nid de boucles.

L'image par T des points entiers de P est donnée par la formule de Presburger :

$$S = \{(x,y) \in \mathbb{Z}^2 \mid \exists (i,j,k) \in \mathbb{Z}^3 : \\ 1 \leq i \leq N \ \wedge \ 1 \leq j \leq N \ \wedge \ 1 \leq k \leq N \ \wedge \ x = 3i + 6k \ \wedge \ y = 2j + 5\}.$$

Le problème de calcul de l'image de P se réduit alors à l'élimination des variables existentielles i, j et k.

L'image du polytope P sans prendre en compte le problème des trous, i.e., en appliquant directement l'algorithme original de Fourier-Motzkin, résulte en

$$T(P) = \{(x,y) \in \mathbb{Z}^2 \mid 9 \leq x \leq 9N \ \wedge \ 7 \leq y \leq 2N + 5\}, \text{ avec } N \geq 1.$$

Le nombre de points entiers dans ce polytope est donné par le quasi-polynôme d'Ehrhart suivant :

$$\mathcal{E}(T(P)) = \begin{cases} 18N^2 - 25N + 8 & \text{si } N \geq 1 \\ 0 & \text{sinon.} \end{cases}$$

Alors que le nombre *exact* des images des points entiers de P calculé par notre algorithme, décrit dans le chapitre 5, est :

$$\mathcal{E}(S) = \begin{cases} 3N^2 - 2N & \text{si } N \geq 1 \\ 0 & \text{sinon.} \end{cases}$$

Cela veut dire que le nombre des éléments du tableau qui sont réellement accédés est seulement $3N^2 - 2N$. Par conséquent, la taille de la mémoire réservée aux éléments du tableau qui ne sont pas utilisés est de $15N^2 - 23N + 8$ éléments.

Nous décrivons maintenant la linéarisation du tableau A en utilisant les algorithmes de calcul et et de dénombrement des images affines de $\mathbb{Z}$-polytopes paramétrés. Pour simplifier cette description, nous procédons sur un ensemble équivalent aux éléments accédés. Cet ensemble est obtenu en appliquant la compression de variables [104] à la fonction d'accès $T(i,j,k) = (x = 3i + 6k,\ y = 2j + 5)$, ce qui résulte en $T'(i,j,k) = (x' = i + 2k,\ y' = j + 2)$. Cela veut dire que la référence de la figure figure 6.1.(a) est supposée égale à `A(i+2k, j+2)=...`.

Chaque élément $A(x_0, y_0)$ est replacé dans le tableau B à l'adresse $B(\mathcal{E}(x_0, y_0, N))$, où $\mathcal{E}(x_0, y_0, N)$ est le quasi-polynôme d'Ehrhart correspondant au nombre de tous les éléments du tableau A accédés avant $A(x_0, y_0)$. Ce polynôme est obtenu en dénombrant les points entiers dans *l'image exacte*, par T, de toutes les itérations qui sont lexicographiquement inférieures ($\prec$) à la première itération accédant à $A(x_0, y_0)$. Soit $T'^{-1}(x_0, y_0) \cap P$ l'ensemble de toutes les itérations référençant $A(x_0, y_0)$. La première itération qui accède à $A(x_0, y_0)$ est égale au minimum lexicographique $\mathbf{i}_{min}(x_0, y_0, N)$ de l'ensemble $T'^{-1}(x_0, y_0) \cap P$. PIP [54] permet de calculer le minimum lexicographique d'un ensemble de contraintes linéaires paramétrées. Dans notre exemple, ce minimum est donné par PIP sous la forme :

$$\mathbf{i}_{min}(x_0, y_0, N) = \begin{cases} \mathbf{i}_{min1}(x_0, y_0, N) = (x_0 - 2N,\ y_0 - 2,\ N) & \text{si } x_0 \geq 2N + 2 \\ \mathbf{i}_{min2}(x_0, y_0, N) = (x_0 - 2M + 2,\ y_0 - 2,\ M - 1) & \text{sinon,} \end{cases}$$

où M est un nouveau paramètre qui est égal à $\lfloor \frac{x_0+1}{2} \rfloor$.

Dans ce qui suit, nous nous focalisons sur la linéarisation du tableau A lorsque le minimum lexicographique est égal à $\mathbf{i}_{min1}(x_0, y_0, N)$.

Soit $\mathcal{S}_1(x_0, y_0)$ l'ensemble des itérations qui précèdent la première itération qui accède à $A(x_0, y_0)$. Cet ensemble est donné par :

$$\mathcal{S}_1(x_0, y_0) = \{(i,j,k) \in P \mid (i,j,k) \prec \mathbf{i}_{min1}(x_0, y_0, N) = \\ i < x_0 - 2N \ \vee \ (i = x_0 - 2N \ \wedge \ j < y_0 - 2) \vee (i = x_0 - 2N \ \wedge \ j = y_0 - 2 \ \wedge \ k < N)\}$$

Les éléments du tableau A accédés par cet ensemble d'itérations sont donnés par la formule de Presburger suivante :

$$\pi(\mathcal{S}_1(x_0, y_0)) = \{(x', y')\mathbb{Z}^2 \mid \exists (i,j,k) \in \mathcal{S}_1(x_0, y_0) : x' = i + 2k \ \wedge \ y' = j + 2\},$$

où x_0 et y_0 sont considérés maintenant comme des paramètres. Le nombre de solutions entières de cette formule de Presburger est donnée par le quasi-polynôme d'Ehrhart :

$$\mathcal{E}_1(x_0, y_0, N) = \begin{cases} N^2 + (y_0 - 2)N - 1 & \text{si} \ \ x_0 = 2N + 2 \\ (x_0 - 3)N + y_0 - 3 & \text{si} \ \ x_0 \geq 2N + 3 \\ 0 & \text{sinon.} \end{cases}$$

En procédant d'une façon similaire, nous pouvons calculer la nouvelle fonction d'accès lorsque le minimum lexicographique est égal à $\mathbf{i}_{min2}(x_0, y_0, N)$. Enfin, la référence $A(3i + 6k, 2j + 5)$ est remplacée par $B(\mathcal{E}(x_0, y_0, N))$, où B est un tableau unidimensionnel (linéaire), $x_0 = i + 2k$ et $y_0 = j + 2$ (voir la figure 6.1) :

$$\mathcal{E}(x_0, y_0, N) = \begin{cases} (x_0 - 3)N + y_0 - 3 & \text{si} \ \ x_0 \geq 2N + 3 \\ (y_0 - 3)N + (x_0 - 3)/2 & \text{si} \ \ x_0 \leq 2N + 1 \ \wedge \ x_0 \ \text{ impair} \\ N^2 + (y_0 - 3)N + (x_0 - 4)/2 & \text{si} \ \ x_0 \leq 2N + 2 \ \wedge \ x_0 \ \text{ pair.} \end{cases}$$

L'avantage de la nouvelle fonction d'accès par rapport à la fonction originale réside d'abord dans la *compression mémoire* : puisque seules les données réellement accédées sont stockées dans le tableau B, la taille de la mémoire allouée pour B est seulement $3N^2 - 2N$. Alors que celle de A est $18N^2 - 25N + 8$. En supposant que la taille d'une donnée est de 4 octets, le tableau A nécessiterait $4(15N^2 - 23N + 8)$ octets de plus que B pour qu'il soit stocké en mémoire. Ce qui correspond à plus de 576KB lorsque $N = 100$ et plus de 57MB lorsque $N = 1000$. Le deuxième avantage de la nouvelle fonction d'accès est *l'amélioration de localité spatiale*. Supposons que le stockage de tableaux en mémoire se fait par lignes (*row-major order*). Pour la référence originale $(3i + 6k, 2j + 5)$, un saut de six lignes est fait à chaque fois que l'indice de la boucle la plus interne est incrémenté. Puisqu'il y a $2N - 1$ éléments par ligne, un saut de $12N - 6$ éléments est effectué lorsque k change, quels que soient les autres indices. Cela pourrait conduire à un défaut de cache, à chaque itération (lorsque N est grand), et les défauts de pages de TLB peuvent aussi être plus fréquents. En revanche, la nouvelle fonction d'accès assure toujours des accès avec des pas de *un* pour les valeurs de i et j, telles que $i + 2k \leq 2N + 1$ (ce qui correspond au cas se produisant le plus fréquemment). Lorsque $i + 2k \geq 2N + 3$ un saut de seulement $2N$ éléments est effectué. Statistiquement parlant, lorsque $N = 100$, cette fonction conduit à environ 74.5% accès avec un pas de taille 1 profitant de la localité spatiale, 23.5% accès avec un pas de taille 200, et moins

de 2% accès avec des pas de taille entre 200 et 30000. Alors que la fonction de référence originale conduit à 99% de pas de taille 1194, et 1% de pas dont la taille est supérieure à 100000.

Notons que lorsqu'il n'y a pas de réutilisation de données, la nouvelle fonction d'accès assure toujours des accès avec des pas de taille 1. Dans notre exemple, il y a une réutilisation de données, mais le nombre de pas de taille 1 est tout de même amélioré. À titre d'exemple, si on fixe i à 3, j à 6 et N à 100, et on laisse varier k entre 1 et 3, la nouvelle fonction accède successivement aux éléments $B(501)$, $B(502)$ et $B(503)$. On peut aussi vérifier la cohérence de données, i.e., si deux itérations accèdent à la même donnée dans le nid de boucles original, ces deux itérations accèdent aussi à la même donnée dans le nid de boucles optimisé.

6.2 Stratégie de remplacement de cache basée sur les équations de distance de réutilisation

Dans [130], en collaboration avec S. Verdoolaege et K. Beyls, nous avons montré l'utilité des algorithmes de comptage de points entiers dans des polytopes paramétrés pour améliorer les performances du cache. Il s'agit précisément de leur application pour améliorer la stratégie de remplacement LRU [23, 24], en se basant sur le calcul des distances de réutilisation [22, 19].

Définition 27. *Une* **référence mémoire** *correspond à une instruction de lecture ou d'écriture, alors qu'une exécution particulière d'une telle lecture ou écriture correspond à un* **accès mémoire**. *Une* **paire de réutilisation** (a_1, a_2) *est une paire d'accès mémoire qui touchent la même localisation mémoire, sans qu'il y ait d'accès intermédiaires à cette même localisation. L'ensemble de données accédées* **ADS** *(*Accessed Data Set) d'une paire de réutilisation (a_1, a_2) est l'ensemble de localisations mémoire différentes accédées entre a_1 et a_2. Cet ensemble est dénoté par $\mathsf{ADS}(a_1, a_2)$. La **distance de réutilisation** d'une paire de réutilisation (a_1, a_2) est le nombre de localisations mémoire différentes accédées entre a_1 et a_2. Cette distance, dénotée par $\mathsf{RD}(a_1, a_2)$, est égale à $|\mathsf{ADS}(a_1, a_2)|$. La distance de réutilisation $\mathbf{FRD}(a_1)$ (*Forward Reuse Distance*) d'un accès mémoire a_1 est la distance de réutilisation de la paire (a_1, a_2). S'il n'y a pas de telle paire, on attribue ∞ à cette distance.

Lemme 4 (Beyls et D'Hollander [23])**.** *Dans un cache associatif de* CS *lignes, utilisant la politique de remplacement LRU, une ligne mémoire, référencée par un accès a, reste dans le cache jusqu'à sa prochaine utilisation si et seulement si* $\mathsf{FRD}(a) < \mathsf{CS}$.

La politique de remplacement de cache LRU peut être améliorée si la distance FRD est plus grande que la taille du cache. Dans ce cas, la donnée concernée peut ne plus être réutilisée ou n'être réutilisée que lorsque elle sera expulsée du cache. Il convient donc de l'expulser *prématurément* du cache et d'utiliser sa localisation pour stocker une autre donnée. La politique de remplacement LRU+CH (LRU+*Cache Hint*), résumée par l'algorithme 4, est basée sur la politique LRU, mais attribue une priorité inférieure

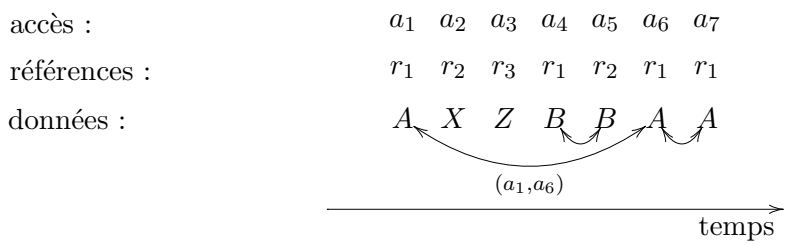


FIG. 6.2 – La première ligne montre 7 accès mémoire, qui sont générés par les références de la seconde ligne. La troisième ligne montre les localisations mémoire correspondantes A, B, X et Z. Les accès a_2 et a_3 ne font pas partie d'une paire de réutilisation, puisque les données auxquelles ils accèdent (X et Z) sont accédées une seule fois. $\mathsf{ADS}(a_1, a_6) = \{B, X, Z\}$; $\mathsf{RD}(a_1, a_6) = |\mathsf{ADS}(a_1, a_6)| = 3$; $\mathsf{RD}(a_6, a_7) = 0$. $\mathsf{FRD}(a_1) = 3$; $\mathsf{FRD}(a_6) = 0$.

Algorithme 4 Politique de remplacement LRU+CH

1. Si la ligne accédée l n'est pas présente dans le cache, remplacer la ligne de cache située à la base de la pile par la ligne l.
2. Si $\mathsf{FRD}(a) < \mathsf{CS}$, déplacer l au sommet de la pile, sinon si $\mathsf{FRD}(a) \geq \mathsf{CS}$, déplacer l à la base de la pile.

aux accès a pour lesquels $\mathsf{FRD}(a) \geq \mathsf{CS}$. Dans [73], Jain et al. prouvent qu'au pire, la politique LRU+CH garantit les mêmes performances que LRU.

Les distances de réutilisation des références d'un programme sont calculées en trois étapes :

1. Calcul des paires de réutilisation des accès mémoires. Pour chaque paire de références (r, s), un ensemble de polytopes $\mathsf{reuse}\,(r \rightarrow s)$ est généré. Ces polytopes représentent toutes les paires de réutilisation, pour lesquelles le premier accès est généré par une exécution de la référence r, et le second accès est généré par s.
2. Pour chaque ensemble de paires de réutilisation (correspondant à une paire de références), un ensemble de polytopes est construit. Ces polytopes décrivent l'ensemble de données accédées (ADS) des paires de réutilisations considérées.
3. Enfin, le nombre de localisations mémoire différentes dans l'ensemble ADS est calculé, pour obtenir la distance de réutilisation de la paire de réutilisation en question.

Les notations suivantes sont utilisées dans les équations de distance de réutilisation. L'ensemble de toutes les références d'un programme est dénoté par $\boldsymbol{\mathcal{R}}$. L'ensemble des variables d'un programme est dénoté par $\boldsymbol{\mathcal{V}}$. L'espace d'itération d'une instruction, dans laquelle une référence r apparaît, est dénoté par $\mathsf{IS}(\boldsymbol{r})$. La localisation mémoire qui est accédée par r à une itération I_r est dénotée par $\boldsymbol{r@I_r}$. Le fait que l'itération i de la référence r est exécuté avant l'itération j de la référence s est exprimé par $\boldsymbol{i_r \prec j_s}$. Enfin l'ensemble des paramètres du programme est dénoté par $\mathbf{p}$.

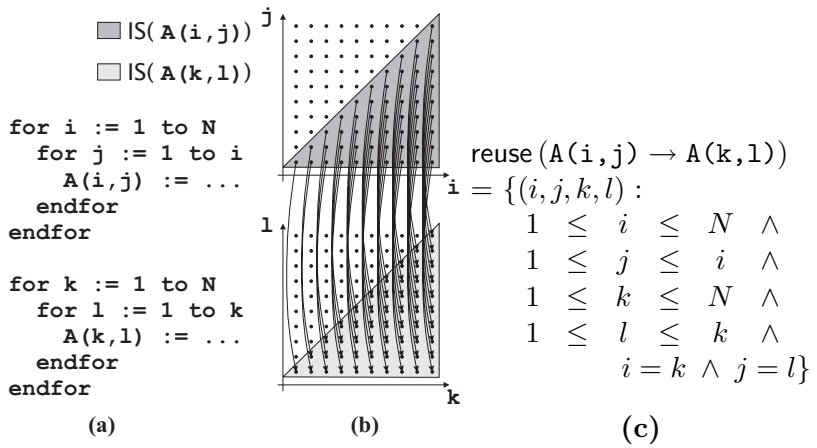


FIG. 6.3 – Paires de réutilisation pour un morceau de programme. Le morceau de programme est donné en (a). Dans (b), les paires de réutilisations sont montrées par des flèches entre les itérations des deux références. Dans (c), les paires de réutilisation sont décrites par des polytopes paramétrés.

6.2.1 Formules de paires de réutilisation

Tout accès mémoire est défini d'une façon unique par la référence r qui le génère, et l'itération I_r à laquelle il se produit. Toutes les paires de réutilisation (x, y), pour lesquelles le premier accès x est généré par la référence r et le second accès y est généré par la référence s, sont combinées dans un ensemble dénoté par $\mathsf{reuse}\,(r \to s)$, qui contient les itérations I_r et J_s générant la réutilisation de la donnée. Ces itérations sont décrites par les équations suivantes :

$$\forall r, s \in \mathcal{R} : \mathsf{reuse}\,(r \to s) = \{\, (I_r, J_s) \in \mathbb{Z}^n : \quad \text{vérifiant les conditions (6.1a)–(6.1d)} \,\}.$$

$$I_r \in \mathsf{IS}(r) \wedge J_s \in \mathsf{IS}(s) \qquad \text{(espace d'itération) (6.1a)}$$

$$I_r \prec J_s \qquad \text{(ordre d'exécution) (6.1b)}$$

$$r@I_r = s@J_s \qquad \text{(même localisation mémoire) (6.1c)}$$

$$\forall t \in \mathcal{R} : \neg\,(\exists K_t \in \mathsf{IS}(t) : I_r \prec K_t \prec J_s \wedge t@K_t = r@I_r) \qquad \text{(pas d'accès intermédiaires) (6.1d)}$$

Les formules ci-dessus correspondent aux contraintes qui doivent être satisfaites pour qu'une paire de réutilisation se produise entre $r@I_r$ et $s@J_s$. L'équation (6.1a) signifie que I_r et J_s font partie des espaces d'itération de r et s (respectivement), (6.1b) exige que I_r soit exécutée avant J_s, (6.1c) signifie que la même localisation mémoire doit être accédée, et (6.1d) assure qu'aucun accès intermédiaire ne touche la même localisation mémoire.

Un exemple d'équations de distance de réutilisation pour un morceau de programme est donné figure 6.3.

6.2.2 Ensemble de données accédées d'une paire de réutilisation

Soit map_r^A la fonction qui associe à un ensemble d'itérations, l'ensemble des éléments du tableau A accédés par la référence r, et soit $\mathsf{iters}_t(I_r, J_s)$ l'ensemble des itérations de la référence t exécutées entre les itérations I_r et J_s :

$$\mathsf{map}_r^A = \{I \rightarrow r@I : I \in \mathsf{IS}(r)\} \tag{6.2}$$

$$\mathsf{iters}_t(I_r, J_s) = \{K_t \in \mathsf{IS}(t) : I_r \prec K_t \prec J_s\} \tag{6.3}$$

Les éléments $\mathsf{ADS}^A(\mathsf{reuse}(r \rightarrow s))$ du tableau A qui sont dans l'ADS des paires de réutilisation de $\mathsf{reuse}(r \rightarrow s)$ sont donnés par :

$$\mathsf{ADS}^A(\mathsf{reuse}(r \rightarrow s)) \;=\; \bigcup_{t \in \mathcal{R}} \mathsf{map}_t^A(\mathsf{iters}_t(\mathsf{reuse}(r \rightarrow s))), \tag{6.4}$$

L'équation (6.4) signifie que l'ensemble de données accédées (ADS) d'une paire de réutilisation peut être trouvé en calculant d'abord les itérations entre utilisation et réutilisation. Ensuite, les ADS sont simplement données par les localisations mémoire touchées par les accès correspondant aux itérations entre utilisation et réutilisation. Le calcul de ADS pour une paire de réutilisation du programme de la figure 6.3(a) est donné figure 6.4.

6.2.3 Distance d'une paire de réutilisation

Afin d'obtenir la distance (de réutilisation) d'une paire de réutilisation, le nombre de localisations mémoire dans son ADS doit être compté :

$$\mathsf{RD}(\mathsf{reuse}(r \rightarrow s)) = \sum_{A \in \mathcal{V}} \mathcal{E}\left(\mathsf{ADS}^A(\mathsf{reuse}(r \rightarrow s)); I_r, J_s, \mathbf{p}\right), \tag{6.5}$$

où $\mathsf{ADS}^A(\mathsf{reuse}(r \rightarrow s))$ est un ensemble de polytopes paramétrés.

Outre le calcul de la distance de réutilisation d'une paire de réutilisation, il est aussi possible de calculer la $\mathsf{FRD}(r)$ (*Forward Reuse distance*) d'une référence mémoire r :

$$\mathsf{FRD}(r) = \sum_{s \in \mathcal{R}, A \in \mathcal{V}} \mathcal{E}\left(\mathsf{ADS}^A(\mathsf{reuse}(r \rightarrow s)); I_r, \mathbf{p}\right) \tag{6.6}$$

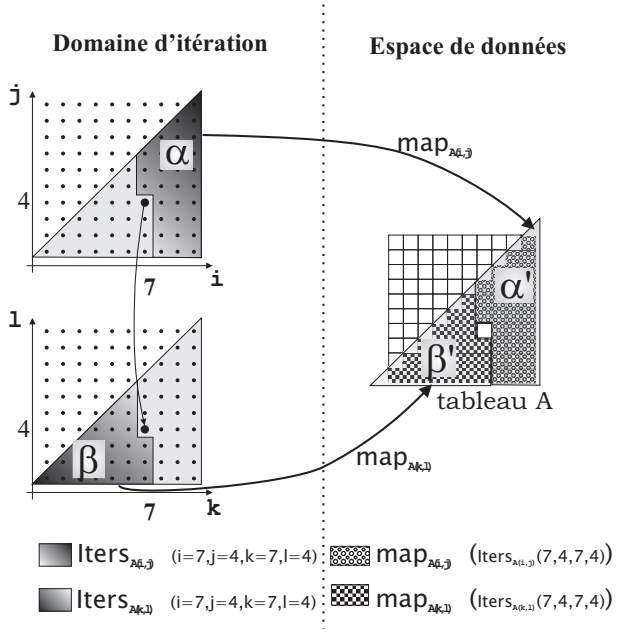
Dans les équations ci-dessus (formules de Presburger), si la distance de réutilisation des accès mémoire générés par la référence r ou s dépend d'un point d'itération I, alors I sera considéré comme un paramètre lors du comptage des points de la formule.

La figure 6.5 présente un exemple de calcul de RD et de FRD.

6.3 Autres applications

6.3.1 Équations de défauts de cache (CME)

Le nombre de défauts de cache, générés par l'exécution d'un programme, est un paramètre fondamental pour observer le comportement du cache et mesurer les perfor-

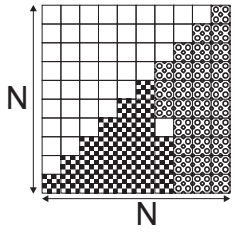


(a)

$$
\begin{aligned}
&\mathsf{ADS}^{\mathtt{A}}\,(\mathsf{reuse}\,(\mathtt{A(i,j)} \rightarrow \mathtt{A(k,l)})) = \\
&\big\{(x,y) : \big(1 \le i \le N \wedge 1 \le j \le i \wedge \quad \mathit{IS}(\mathtt{A(i,j)}) \\
&\qquad 1 \le k \le N \wedge 1 \le l \le k \wedge \quad \mathit{IS}(\mathtt{A(k,l)}) \\
&\qquad i = k \wedge j = l\big) \wedge \quad \text{même localisation mémoire} \\
&\qquad \big(x < k \vee x = k \wedge y < l \vee \quad \text{données accédées entre} \\
&\qquad x > i \vee x = i \wedge y > j\big)\big\} \quad \text{utilisation et réutilisation}
\end{aligned}
$$

(b)

FIG. 6.4 – Représentation graphique du calcul de $\mathsf{ADS}(\mathsf{reuse}\,(\mathtt{A(i,j)} \rightarrow \mathtt{A(k,l)}))$, pour le programme de la figure 6.3(a). U ne seule paire de réutilisation est montrée (de l'accès de la référence `A(i,j)` à l'itération $(i{=}7, j{=}4)$ vers l'accès de la référence `A(k,l)` à l'itération $(k{=}7, l{=}4)$). À gauche, $\mathsf{iters}_{\mathtt{A(i,j)}}(i{=}7, j{=}4, k{=}7, l{=}4)$ et $\mathsf{iters}_{\mathtt{A(k,l)}}(i{=}7, j{=}4, k{=}7, l{=}4)$ sont indiqués dans l'espace d'itération des références par les régions α et β. Après application des fonctions $\mathsf{map}^{\mathtt{A}}_{(\mathtt{i,j})}$ et $\mathsf{map}^{\mathtt{A}}_{(\mathtt{k,l})}$, les parties du tableau A qui sont accédées par les itérations comprises entre $(i{=}7, j{=}4)$ et $(k{=}7, l{=}4)$, sont montrées par les régions α' and β'.



(a)

- $\mathsf{RD}(\mathsf{reuse}\,(\mathtt{A(i,j)} \rightarrow \mathtt{A(k,l)})) =$

  $\mathcal{E}\left(\mathsf{ADS}^{\mathtt{A}}\,(\mathsf{reuse}\,(\mathtt{A(i,j)} \rightarrow \mathtt{A(k,l)}))\,;\mathtt{i},\mathtt{j},\mathtt{k},\mathtt{l},\mathtt{N}\right) = \dfrac{N^2+N}{2} - 1.$

- $\mathsf{FRD}(\mathtt{A}(\mathtt{i},\mathtt{j})) = \dfrac{N^2+N}{2} - 1.$

(b)

FIG. 6.5 – La figure **(a)** montre les éléments du tableau A accédés entre utilisation et réutilisation pour la paire de réutilisation de la figure 6.4. Le nombre des éléments dans cet ensemble est $\frac{N^2+N}{2} - 1$, ce qui est égal à la distance de la paire réutilisation considérée. Dans la figure **(b)**, la distance de réutilisation et la FRD sont décrites en fonction d'une matrice de taille N, utilisant les équations (6.5)-(6.6).

mances du programme. C'est aussi un outil essentiel pour guider les méthodes d'optimisations de programmes. Contrairement à la simulation, le calcul du nombre de défauts de cache par comptage de points entiers dans des polytopes (paramétrés), permet de fournir une réponse *exacte* et de capturer des comportements de cache qui peuvent ne pas être visibles par la simulation [33]. Dans ce qui suit, nous présentons brièvement une méthode due à Chatterjee et al. [33]. Il s'agit d'une alternative de la méthode originale de calcul des équations de défauts de cache CME (*Cache Miss Equations*) de Ghosh et al. [64]. La principale différence entre les deux méthodes est que la méthode de Ghosh est basée sur le calcul des vecteurs de réutilisation [152], alors que celle de Chatterjee est basée sur l'arithmétique de Presburger. L'avantage de la méthode de Chatterjee réside dans le fait qu'elle est exacte et plus générale que la méthode de Ghosh, lorsque le cache est à accès direct. Mais elle est moins robuste pour des caches associatifs par ensembles. De plus, sa complexité de calcul est souvent plus grande, puisqu'elle utilise l'arithmétique de Presburger, qui est exponentielle dans le pire cas [150].

Ordre lexicographique des accès mémoire

Soit (R_v, m) un accès mémoire correspondant à l'exécution de la référence R_v à l'itération m. Les accès mémoire qui précèdent (R_v, m) sont donnés par tous les accès se produisant à une itération ℓ, telle que $\ell \prec m$ (ordre lexicographique), auxquels sont ajoutés tous les accès qui sont effectués à l'itération m par les références qui précèdent R_v (ordre des instructions). Le fait qu'un accès (R_u, l) précède un autre accès (R_v, m) est dénoté par $(R_u, l) \prec (R_v, m)$.

Placement de données dans le cache

Considérons un cache associatif par ensembles de blocs, et soient A l'associativité de ce cache, B la taille d'un bloc, C la capacité du cache et $S = \frac{C}{AB}$ le nombre de ses ensembles de blocs. Avec cette organisation, une localisation mémoire m est placée dans l'ensemble s du cache, tel que $s = \lfloor \frac{m}{B} \rfloor \bmod S$. Supposons que la localisation mémoire m est référencée par un tableau Y^x de α_x éléments. La localisation m sera placée dans l'ensemble du cache défini par le formule de Presburger :

$$\begin{aligned} \mathrm{Map}(m, w, s) = 0 \leq s < S \wedge \\ B(wS + s) \leq m < B(wS + s) + B \wedge \\ \mu_x - B < B(wS + s) < \mu_x + \beta_x \alpha_x, \end{aligned} \tag{6.7}$$

où w est une variable supplémentaire bornée par la troisième clause de l'équation (6.7), β_x est la taille en octets d'un élément du tableau Y^x, et μ_x est l'adresse du début du tableau en mémoire. $B(wS+s)$ représente l'adresse du premier octet du bloc contenant la localisation mémoire m.

Lorsque le cache est à accès direct, Chatterjee et al. proposent trois types de formules de Presburger pour modéliser les défauts de cache.

Défauts intérieurs

Pour qu'un accès à un bloc mémoire b résulte en un défaut intérieur (*interior miss*), il faut que les deux conditions suivantes soient vérifiées : (i) il y a un accès antérieur à un bloc mémoire b' *différent de* b, et qui se place dans le même ensemble du cache que b, (ii) aucun accès au bloc b ne se produit entre l'accès antérieur au bloc b' et l'accès courant au bloc b. Soit $R_u = (Y^x, F_u, S_p)$ la référence accédant à un bloc mémoire b_u à l'itération i, où F_u est la fonction d'accès au tableau Y^x et S_p est l'instruction où apparaît cette référence, et soit $R_v = (Y^y, F_v, S_q)$ une référence accédant à un bloc mémoire b_v à l'itération j. Supposons que l'accès (R_v, j) précède (R_u, i), et que les blocs mémoire b_u et b_v sont différents, mais qu'ils se placent dans le même ensemble s (bloc pour un cache à accès direct). Alors, l'accès (R_u, i) correspond à un défaut interne s'il n'existe pas de référence $R_w = (Y^z, F_w, S_r)$ qui accède au bloc b_u à une itération k, telle que $(R_v, j) \prec (R_w, k) \prec (R_u, i)$. Ces conditions sont exprimées par la formule de Presburger :

$$\begin{aligned} ((R_u, i) \in \mathrm{IntMiss}(\mathbb{L})) = i \in \mathcal{I} \wedge \\ \exists d, s : \mathrm{Map}(\mathcal{L}_x(F_u(i)), d, s) \wedge \\ \exists e, j, v : (R_v, j) \prec (R_u, i) \wedge \\ \mathrm{Map}(\mathcal{L}_y(F_v(j)), e, s) \wedge \\ \neg(\exists k, w : (R_v, j) \prec (R_w, k) \prec (R_u, i) \wedge \\ \mathrm{Map}(\mathcal{L}_z(F_w(k)), d, s)) \wedge d \neq e, \end{aligned} \tag{6.8}$$

où $\mathbb{L}$ est un nid de boucles et $\mathcal{L}_x(F_u(i))$ est l'adresse (en mémoire) de l'élément du tableau accédé par la fonction $F_u(i)$.

Défauts de bord

Les défauts de bord (*boundary defaults*) sont des défauts qui dépendent de l'état initial du cache. Un accès $(R_u = (Y^x, F_u, S_p), i)$ à un bloc mémoire b_u correspond à un défaut de bord si les deux conditions suivantes sont vérifiées : (i) il n'y a aucun accès (R_v, j) qui précède (R_u, i), et qui accède à un bloc mémoire b_v placé dans le même ensemble du cache que b_u, (ii) b_u ne se trouve pas dans l'état de cache initial, dénoté par $\mathbb{C}_{in}$. Ces conditions sont équivalentes à la formule de Presburger :

$$\begin{aligned}((R_u, i) \in \text{BoundMiss}(\mathbb{L}, \mathbb{C}_{in})) = i \in \mathcal{I} \wedge \\ \exists d, s : \text{Map}(\mathcal{L}_x(F_u(i)), d, s) \wedge \\ \neg(\exists e, j, v : (R_v, j) \prec (R_u, i) \wedge \\ \text{Map}(\mathcal{L}_y(F_v(j)), e, s)) \wedge \\ \mathcal{B}(\mathcal{L}_x(F_u(i))) \notin \mathbb{C}_{in}(s),\end{aligned} \tag{6.9}$$

où $\mathcal{B}(\mathcal{L}_x(F_u(i)))$ est l'adresse du bloc contenant la localisation mémoire $\mathcal{L}_x(F_u(i))$.

État de cache final

Lorsque le nid de boucles $\mathbb{L}$ ne contient aucun accès mémoire à un ensemble s, l'état de cache final de l'ensemble s, dénoté $\mathbb{C}_{out}(s)$, est le même que l'état de cache initial $\mathbb{C}_{in}(s)$. Autrement dit, l'état de cache final de l'ensemble s est donné par l'adresse du bloc mémoire préalablement placé dans l'ensemble s, et qui n'est pas remplacé ultérieurement par un autre bloc. Ceci correspond à la formule de Presburger :

$$\begin{aligned}(\mathbb{C}_{out} = \Psi(\mathbb{L}, \mathbb{C}_{in})) = \forall s \in [0, S-1] : (\exists i : i \in \mathcal{I} \wedge \\ (\exists d : \text{Map}(\mathcal{L}_x(F_u(i)), d, s)) \wedge \\ \neg(\exists e, j, v : (R_u, i) \prec (R_v, j) \wedge \text{Map}(\mathcal{L}_y(F_v(j)), e, s) \wedge \\ \mathbb{C}_{out}(s) = \mathcal{B}(\mathcal{L}_x(F_u(i))))) \vee \\ (\neg(\exists e : \text{Map}(L_x(F_u(i)), e, s)) \wedge \mathbb{C}_{out}(s) = \mathbb{C}_{in}(s))\end{aligned} \tag{6.10}$$

Lorsque le cache est associatif ou associatif par ensembles de blocs, la solution proposée par Chatterjee et al. est moins robuste. Soit (R_u, i) l'accès, à l'itération i, à un bloc mémoire b_u. Pour que (R_u, i) corresponde à un défaut intérieur, les mêmes conditions que pour un cache à accès direct (formule (6.8)) doivent être vérifiées. Mais ici, il faut qu'il y ait au moins A (A étant l'associativité du cache) accès différents, précédant (R_u, i), à des blocs mémoire différents placés dans le même ensemble de cache que b_u. Cette solution est faisable pour une petite associativité, mais sa complexité explose pour des associativités plus grandes. Les défauts de bord et l'état du cache sont aussi difficiles à construire dans ce cas.

Une fois que les formules de défauts de cache sont construites, il suffit de compter leurs solutions entières pour trouver le nombre de défauts de cache. Ces formules peuvent être simplifiées par Omega pour obtenir des unions d'images de polytopes paramétrés. Puis, leurs nombres de points entiers sont calculés, en utilisant les algorithmes de comptage décrits dans les chapitres précédents. Il convient néanmoins de

signaler que les équations de défauts de cache résultent en des formules de Presburger souvent complexes. La résolution de certaines formules peut ne pas être possible avec les outils mathématiques dont nous disposons actuellement, à cause de leur complexité exponentielle en nombre de variables notamment.

6.3.2 Communication dans les réseaux de processus de Kahn

Les réseaux de processus de Kahn (*Kahn Process Networks*, ou KPN) permettent d'exprimer les applications de traitement de signal (DSP) sous une forme parallèle, afin de rendre leur exécution plus adaptée aux architectures ciblées. Turjan et al. [138] ont proposé une méthode de transformation automatique des applications DSP à des réseaux de processus de Kahn. Comme exemple de ces réseaux, ils présentent celui d'un processus producteur communiquant avec un processus consommateur via une file de type FIFO, où les données sont stockées dans l'ordre de leur production. Lorsque la consommation de ces données est faite dans un ordre différent de celui de leur production, un mécanisme de réorganisation de données est nécessaire pour assurer leur cohérence. La méthode de Turjan et al. [138] consiste en une approche qui permet de trouver, si nécessaire, une réorganisation de données basée sur le calcul de quasi-polynômes d'Ehrhart. L'outil *Compaan* [79] permet de transformer un programme DSP, écrit en Matlab, en un réseau de processus en trois étapes. D'abord le code est transformé en un code à assignation unique (SAC), représentant le graphe de dépendances des nids de boucles du programme initial. Ensuite, le code (SAC) est transformé en une structure de données PRDG (*Polyhedra Reduced Dependence Graph*), qui est une représentation mathématique compacte du graphe de dépendance sous forme de polyèdres. Enfin, le PRDG est converti en un réseau de processus parallèles. Ces processus communiquent entre eux selon les dépendances de données décrites par le graphe de dépendances.

Paire Producteur/Consommateur

Une paire producteur/consommateur correspond à deux morceaux de programmes, où des données écrites par le premier morceau de programme sont lues par le second. Considérons, par exemple, deux nids de boucles $\mathbb{L}_1$ et $\mathbb{L}_2$, tels que $\mathbb{L}_1$ écrit des données dans un tableau bidimensionnel A qui sont ensuite lues par $\mathbb{L}_2$. $\mathbb{L}_1$ et $\mathbb{L}_2$ définissent donc une paire de processus, producteur/consommateur, communiquant entre eux via le tableau A. Pour obtenir un réseau de processus de Kahn, il faut remplacer le tableau A par une liste *linéaire* de type FIFO. Dans cette file, les données sont écrites dans l'ordre de leur production par le nid de boucles $\mathbb{L}_1$, avant qu'elles ne soient lues par $\mathbb{L}_2$. Le remplacement d'un tableau de dimension quelconque par une file est appelé *linéarisation*. Ceci est en effet similaire à la linéarisation de tableaux décrite dans la section 6.1. Puisque la file est de type FIFO, les données qui y sont écrites sont consommées dans le même ordre que leur écriture. Deux cas sont donc possibles :

1. La séquence de données produites est la même que la séquence de données consommées. Dans ce cas, la liste FIFO suffit pour réaliser la linéarisation.
2. La séquence de données produites est différente de la séquence de données consommées. Dans ce cas, un changement de l'ordre des données produites est néces-

saire pour assurer l'exactitude des données consommées. Afin de réaliser une telle réorganisation, on doit être capable de comparer l'ordre de données produites avec celui des données consommées. Plus précisément, on a besoin d'une fonction qui permet de déterminer l'ordre de l'exécution de chaque itération d'un nid de boucles par rapport aux autres itérations. Cette fonction est appelée fonction de rang (*rank function*), et est notée *rank*.

En connaissant la fonction de correspondance entre les itérations du producteur et celles du consommateur (F), ainsi que la fonction de rang du producteur ($rank_P$), il est possible d'établir une fonction qui calcule, pour toute itération du consommateur, l'ordre dans lequel une donnée arrive au consommateur via la file FIFO. Cette fonction, dite *fonction de lecture*, est obtenue en composant les fonctions $rank_P$ et F, comme suit :

$$read(\mathbf{x}_C) = rank_P(\mathbf{x}_P) \circ F(\mathbf{x}_C),$$

où $\mathbf{x}_C$ et $\mathbf{x}_P$ sont respectivement les vecteurs d'itération des processus producteur et consommateur.

Calcul de la fonction de rang

Soit $P \in Q^d$ le polytope défini par les contraintes sur les bornes d'un nid de boucles. La fonction de rang correspondant à une itération donnée $\mathbf{x} \in P \cap \mathbb{Z}^d$ est égale au nombre des itérations qui sont exécutées avant $\mathbf{x}$. Ce nombre est obtenu en calculant les points entiers $\mathbf{x}'$ du polytope P qui sont lexicographiquement inférieurs à $\mathbf{x}$, i.e., $\{\mathbf{x}' \in P \mid \mathbf{x}' \prec \mathbf{x}\}$.

$$rank(\mathbf{x}) = \sum_{i=1}^{d} |J_i(\mathbf{x})| + 1, \tag{6.11}$$

avec

$$J_i(\mathbf{x}) = \{\mathbf{x}' \in P \cap \mathbb{Z}^d \mid x'_1 = x_1 \wedge \cdots \wedge x'_{i-1} = x_{i-1} \wedge x'_i < x_i\},$$

et $|J_i(\mathbf{x})|$ est le nombre de points entiers dans dans le polytope $J_i(\mathbf{x})$.

Exemple 31. Considérons le nid de boucles :

```
for (i=2; i<=N-1; i++) {
  for (j=i; j<=N-1; j++) {
    ...
  }
}
```

Le domaine d'itération de ce nid de boucles est donné par les points entiers du polytope paramétré : $P = \{i, j \in Q^2 \mid 2 \leq i \leq N-1 \wedge i \leq j \leq N-1\}$. Le rang d'une itération (i, j) ($rank(i, j)$) est donné par la somme des nombres de points entiers dans deux polytopes paramétrés $J_1(i, j)$ et $J_2(i, j)$ plus 1, avec

$$J_1(i, j) = \{(i', j') \in P \mid i' < i\},$$

$$J_2(i, j) = \{(i', j') \in P \mid i' = i \wedge j' < j\},$$

où, i et j sont considérés comme des paramètres. Le résultat est :

$$rank(i, j) = \mathcal{E}(J_1(i, j)) + \mathcal{E}(J_2(i, j)) + 1 = (i-2)N + (-i^2/2 - i/2 + j + 2).$$

Le comptage de points entiers dans les polytopes générés par le calcul de la fonction de rang se fait par les algorithmes décrits dans le chapitre 3. Par ailleurs, Turjan et al. [138] évoquent le fait que lorsque les pas des indices de boucles sont supérieurs à un, la fonction de rang ne peut être calculée, puisque seulement un sous-ensemble des points entiers du polytope défini par les bornes du nid de boucles correspondent à des itérations valides. Ceci ne pose plus un problème en utilisant notre algorithme de comptage de points entiers dans des $\mathbb{Z}$-*polytopes* (chapitre 4). À noter enfin que d'autres travaux récents sur la génération de réseaux de processus [139, 146], inspirés de [138], recourent aussi aux algorithmes de comptages de points entiers dans des polytopes paramétrés.

6.3.3 Calcul de la taille mémoire nécessaire pour des applications multimédia

Dans les programmes traitant principalement des tableaux, tels que les applications de traitement de signal numérique (DSP), le coût de l'espace mémoire utilisé et de l'énergie consommée est responsable d'une grande partie du coût global d'un système embarqué. Plus la taille de la mémoire est grande, plus l'énergie consommée par chaque opération mémoire est grande. Par conséquent, la taille de la mémoire est un des facteurs dominants qui affecte le coût d'un système embarqué. Il est donc très important de minimiser la taille de la mémoire nécessaire pour l'exécution d'un programme. Pour ce faire, il est essentiel de disposer d'un outil d'estimation [155] ou, encore mieux, de calcul de la taille exacte de mémoire nécessaire. Dans ce qui suit, nous décrivons brièvement la méthode de Zhu et al.[156], permettant de calculer la taille exacte de la mémoire nécessaire pour des applications multimédia.

Les codes sources des applications multimédia traitent souvent des signaux qui sont produits à des dates antérieures, et consommés à des dates ultérieures. Ces signaux sont utilisés durant plusieurs itérations de temps. i.e., ils doivent rester en mémoire pendant un certain temps appelé leur *durée de vie*. Le problème est de déterminer le nombre *minimum* de localisations mémoire nécessaires pour stoker les signaux d'un programme multimédia durant son exécution. Autrement dit, le but est de calculer l'occupation maximale de la mémoire en supposant que les signaux ne sont stockés en mémoire que pendant leurs durées de vie. En effet, un programme multimédia peut avoir un grand nombre de signaux, mais puisque les signaux ayant des temps de vie disjoints peuvent occuper la *même* localisation mémoire, la quantité de la mémoire nécessaire pour l'exécution du programme peut être beaucoup plus petite que le nombre total de ses signaux. À titre d'exemple, Zhu et al. [156] donnent un morceau de programme, où sur un ensemble de 3,759,063 signaux, seulement 33,284 peuvent être en mémoire en même temps.

Le calcul de la taille de la mémoire se fait en quatre étapes :

1. Extraire les références aux tableaux à partir du code source de l'application multimédia, et décomposer chaque référence à un tableau en une union disjointe de LBLs (*linearly bounded lattices*)[2]. Cette décomposition est motivée par le fait que le calcul de la taille de mémoire nécessaire peut être simplifié d'une manière significative lorsque les références aux tableaux sont disjointes. Afin de calculer

[2]Un LBL est le résultat de l'image des itérations d'un nid de boucles par une fonction affine (voir le chapitre 5).

une telle décomposition, Zhu et al. [156] proposent une méthode basée sur les opérations d'intersection et de différence sur les LBLs.

2. Déterminer la taille de la mémoire après chaque bloc (nid de boucles) du code. Après la décomposition des références en LBLs disjoints, on détermine, pour chaque LBL, quel est le bloc qui le produit et quel est le bloc qui le consomme. En se basant sur cette information, on peut calculer la taille *exacte* de la mémoire entre les différents blocs, puisqu'il est possible de calculer le nombre exact de points entiers d'un LBL.
3. Si, dans un bloc, aucune donnée n'est consommée pour la dernière fois, le bloc correspondant (producteur) peut être ignoré à l'étape 4. D'une manière similaire, si la somme de la quantité de mémoire nécessaire pour stocker les LBLs nouvellement créés par un bloc et de la taille de la mémoire à l'entrée de ce bloc, est inférieure à la taille maximale calculée précédemment, ce dernier bloc sera ignoré à l'étape 4. Cela veut dire qu'on ne se concentre que sur les portions du code qui pourraient augmenter la taille de mémoire nécessaire.
4. Pour le reste des blocs du code, calculer la taille maximale de la mémoire nécessaire pour stocker les données vivantes à chaque itération du bloc courant. Ceci est basé sur le calcul des vecteurs d'itération min et max référençant des données scalaires, qui correspondent respectivement à des augmentations et des diminutions de la taille de la mémoire.

Dans cette méthode de calcul de la taille minimale de mémoire nécessaire pour l'exécution d'un programme, il est indispensable de disposer d'un moyen de calculer la taille *exacte* d'un LBL, ou autrement dit, le nombre des éléments d'un tableau référencés par un ensemble convexe d'itérations. Ce nombre est égal au nombre de points entiers dans l'image affine d'un $\mathbb{Z}$-polytope. Zhu et al. [156] proposent une méthode leur permettant de faire ce calcul. Cependant, elle ne peut être appliquée à des nids de boucles paramétrés. De plus, sa complexité est clairement supérieure que celle de notre méthode ou celle de Verdoolaege, décrites dans le chapitre 5. La complexité de la méthode de Zhu et al. est due au fait qu'ils parcourent tous les points entiers de la projection (rationnelle) pour déterminer s'ils correspondent à des trous ou non (en calculant le nombre de points entiers dans leurs préimages).

6.3.4 Optimisation de la distribution de données sur des machines de type NUMA

Les machines de type NUMA (*Non-Uniform Memory Access*) sont des machines multiprocesseurs à mémoire distribuée, qui permettent d'exécuter des programmes écrits avec un langage parallèle, tels que HPF (*High Performance Fortran*) par exemple. Dans ce type de machine, les données et les calculs sont distribués sur les différents nœuds. Plus les calculs effectués par processeur accèdent à des données distantes, i.e., ne se trouvant pas dans la mémoire locale du nœud, moins le système est performant. Plusieurs méthodes de transformation automatique de programmes [69, 78, 6, 4], tentent de transformer les données de façon à ce que les calculs se fassent le plus souvent possible sur des données locales. Heine et Slowik [69] proposent d'utiliser les quasi-polynômes d'Ehrhart comme une métrique de mesure de la qualité de telles transformations. L'idée

de leur méthode consiste à proposer un ensemble de transformations et de distributions optimales. Ensuite, pour chaque transformation, les itérations de nids de boucles qui causent des accès locaux sont représentées par des polytopes convexes. Enfin, les polynômes d'Ehrhart, correspondant aux nombres de points entiers de ces polytopes, sont calculés et comparés pour juger quelle est la transformation qui améliore le mieux la localité de données.

Considérons un nid de boucles de domaine d'itération $\mathcal{I}$ s'exécutant sur une machine parallèle à mémoire distribuée de $\mathcal{P}$ processeurs. L'objectif est de juger si une transformation de données affine $\mathcal{T}$ est efficace, en se basant sur le calcul du nombre des accès locaux qu'elle génère. Pour simplifier cette présentation, on considère une référence unique R à un tableau A. On commence par décomposer le domaine $\mathcal{I}$ en un ensemble de sous-domaines $\mathcal{I}_p$, tels que $\mathcal{I}_p$ est un sous-ensemble des itérations $\mathcal{I}$ exécutées par le processeur numéro p. L'affectation des itérations aux différents processeurs est considérée cyclique dans une direction parallèle j, i.e.,

$$\mathcal{I}_p = \{\mathbf{i} \mid \mathbf{i} \in \mathcal{I} \wedge i_j \bmod \mathcal{P} = p\},$$

où, $\mathcal{P}$ est le nombre total de processeurs. L'expression $i_j \bmod \mathcal{P} = p$ est ensuite remplacée par $\{(i_j - \mathcal{P}.z) = p \wedge \mathcal{P}.z \leq i_j \leq \mathcal{P}.(z+1)\}$, où z est une nouvelle variable libre. $\mathcal{I}_p$ est donc un polytope dont le nombre de points entiers correspond au nombre des itérations affectées au processeur p. Soit $f(\mathbf{i})$ la fonction de référence aux éléments du tableau A, où $\mathbf{i}$ est un vecteur d'itération du nid de boucles. Après application de la transformation de données $\mathcal{T}$ au tableau A, la nouvelle fonction de référence à ce tableau devient $\mathcal{T}(f(\mathbf{i}))$, i.e., pour toute itération $\mathbf{i}$, l'élément accédé est $\mathbf{x} = \mathcal{T}(f(\mathbf{i}))$. La distribution cyclique des blocs du nouvel espace d'accès peut être examinée pour déterminer si un élément du tableau A ($\mathbf{x} = \mathcal{T}(f(\mathbf{i}))$) accédé par une itération $\mathbf{i}$ est local à un processeur p. La contrainte correspondante est :

$$C_p : B.p \leq \pi_d(\mathbf{x}) - (B.\mathcal{P}).z' < B.(p+1),$$

où z' est une nouvelle variable libre, B est la taille des blocs de données affectés cycliquement aux différents nœuds, d est la dimension de distribution du tableau, et $\pi_d(\mathbf{x})$ est la projection sur la composante x_d de l'élément $\mathbf{x}$. Par conséquent, l'ensemble des itérations $\mathcal{L}_p$ accédant à des données locales pour un processeur p sont :

$$\mathcal{L}_p(\mathbf{x})\{\mathbf{x} \in \mathcal{I}_p \mid C_p(\mathbf{x})\}.$$

Le nombre $\mathcal{R}_p$ des itérations accédant à des données distantes pour un processeur p peut aussi être calculé, en faisant la différence de $\mathcal{I}_p$ et $\mathcal{L}_p$ ($\mathcal{R}_p = \mathcal{I}_p - \mathcal{L}_p$). Le nombre de points entiers dans le polytope $\mathcal{L}_p$ peut être calculé, en utilisant les algorithmes du chapitre 3, en considérant les variables z et z' comme des variables libres supplémentaires.

En connaissant le nombre des accès locaux pour toutes les références et tous les processeurs, il suffit de faire leur somme pour trouver le nombre des accès locaux du nid de boucles global. Ce nombre correspond à la métrique de mesure de la qualité de transformation désirée.

Chapitre 7

Conclusion

Contexte

Nous nous sommes intéressés dans ce travail aux méthodes d'optimisation automatique de programmes, basées sur l'analyse et la transformation de nids de boucles affines. La particularité de ces méthodes est qu'elles obéissent aux spécifications du modèle polyédrique, qui représente les itérations de nids boucles par $\mathbb{Z}$-polytopes (paramétrés), et les références à des tableaux par des images affines de tels $\mathbb{Z}$-polytopes. Pour pouvoir répondre aux différents problèmes soulevés par de telles méthodes, il est indispensable de disposer de moyens mathématiques permettant de manipuler et de dénombrer des points entiers appartenant à des ensembles convexes paramétrés.

Travaux fondamentaux

Dans le cadre de cette thèse, nous avons développé un algorithme qui calcule analytiquement le nombre de points entiers d'un polytope paramétré : la solution est donnée par un ou plusieurs quasi-polynômes d'Ehrhart, définis sur des domaines de validité différents. Chaque polynôme est obtenu en évaluant une fonction génératrice rationnelle paramétrée, caractérisant le polytope sur le domaine considéré. Contrairement aux algorithmes précédents, la taille de la solution et le temps de calcul de notre algorithme sont polynomiaux (pour une dimension fixée). Ce comportement polynomial est dû principalement à l'utilisation de la décomposition en cônes unimodulaires de Barvinok et la représentation fractionnelle (ou par les parties entières inférieures ou supérieures) des nombres périodiques. Cet algorithme est particulièrement efficace, et est déjà largement utilisé par une communauté scientifique pluridisciplinaire.

Afin de permettre l'analyse exacte des références à des tableaux, nous avons aussi proposé un algorithme de calcul et de dénombrement des images affines d'unions de $\mathbb{Z}$-polytopes paramétrés. Ces images sont d'abord écrites sous forme de formules de Presburger. Ensuite, nous éliminons leurs égalités et leurs variables existentielles en utilisant une variante de l'algorithme de Fourier-Motzkin. Comme tous les algorithmes implémentés à ce jour, cet algorithme est exponentiel dans le pire des cas, mais il est utilisable en pratique, en particulier pour l'analyse des références à des tableaux. Contrairement aux algorithmes connus précédemment [144, 28], notre algorithme calcule l'image elle-même, et non pas un ensemble (de dimension supérieure) ayant le

même nombre de points entiers. Ceci a l'avantage de produire des quasi-polynômes de taille plus petite que celle des polynômes fournis par les autres algorithmes, ce qui est très important pour certaines optimisations qui intègrent ces quasi-polynômes dans les programmes optimisés. Par ailleurs, nous avons montré que le temps de calcul de notre méthode est souvent meilleur, en particulier lorsque les coefficients des variables existentielles sont petits.

Nous nous sommes également intéressés aux problèmes d'analyse de nid de boucles plus généraux, où les pas des indices sont supérieurs à un. Pour faciliter l'analyse de tels nids de boucles, nous avons développé un algorithme permettant de compter les points entiers d'une union (non disjointe) de $\mathbb{Z}$-polytopes paramétrés. Les algorithmes précédents proposés dans ce contexte, sont exponentiels dans le pire des cas, et ce même pour un petit nombre de $\mathbb{Z}$-polytopes, puisqu'ils sont basés sur le calcul des unions disjointes de $\mathbb{Z}$-polytopes. De plus, ces algorithmes sont ou bien non paramétrés, ou bien non implémentés. En revanche, notre algorithme utilisant le principe d'inclusion-exclusion est polynomial (pour une dimension et un nombre de $\mathbb{Z}$-polytopes fixés). Puisque le résultat de l'image affine d'un $\mathbb{Z}$-polytope est donné par une union de $\mathbb{Z}$-polytopes, ce dernier algorithme est aussi utilisé pour dénombrer de telles images.

Portée du travail

Les algorithmes que nous avons proposés sont très utiles en optimisation automatique de programmes. Ils permettent de répondre à différentes questions liées à l'analyse et la transformation de nids de boucles affines. Dans le cadre de cette thèse nous nous en sommes servis pour étendre l'utilisation de la méthode de linéarisation de tableaux, due à Loechner et al. [97], à des problèmes plus généraux contenant des nids de boucles où l'espace de données accédées n'est pas convexe. La linéarisation de tableaux permet à la fois de faire la compression mémoire et d'améliorer la localité spatiale des programmes transformés. Ces algorithmes ont été aussi utilisés pour calculer les distances de réutilisation de données, qui servent de métrique pour guider les stratégies de remplacement de données, afin de réduire le nombre de défauts de cache.

Bien entendu, les nouveaux algorithmes peuvent être utilisés par plusieurs méthodes d'optimisation qui, jusque-là, utilisaient des algorithmes de comptage moins efficaces ou non exacts. Parmi ces optimisations, on peut citer le calcul de la taille minimale de mémoire nécessaire pour l'exécution d'un programme, le nombre de défauts de cache, le calcul des fonctions de rang dans les réseaux de processus de Kahn et l'optimisation de la distribution de données dans des machines multiprocesseur à mémoire distribuée. Bien que nous nous soyons focalisés sur les utilisations de nos algorithmes par les méthodes d'optimisation automatique de programmes, il convient de signaler que ces algorithmes trouvent aussi des applications dans d'autres domaines, tels que les mathématiques et l'économie.

Perspectives

À court terme, un certain nombre d'optimisations sont à apporter à l'algorithme de calcul des images affines de $\mathbb{Z}$-polytopes. Il s'agit par exemple de mettre en place une heuristique permettant de choisir le meilleur ordre d'élimination des variables existentielles afin de réduire le temps de calcul. De plus, lorsqu'on n'est intéressé que par

le nombre de points que comporte une image d'un $\mathbb{Z}$-polytope et pas par l'image elle-même, il est possible d'intégrer les règles de simplification de Verdoolaege et al. [144]. Certaines de ces règles sont implicites dans notre méthode, et d'autres ne sont utiles que lorsque les coefficients des variables existentielles sont grands. Puisque ceci peut être détecté avant de commencer la projection, on a la possibilité de choisir, selon les cas, de les appliquer ou non. L'implémentation pourrait choisir, dans un futur proche, d'utiliser l'algorithme le plus efficace, en calculant l'union disjointe des images avant d'appliquer celui de Verdoolaege. La méthode de facettes [37] peut aussi être utilisée dans ce cas.

Nous envisageons également d'appliquer la linéarisation de tableaux et les autres méthodes d'optimisation concernées par nos travaux théoriques à des problèmes réels issus des benchmarks. Il convient aussi de profiter des outils mathématiques dont nous disposons maintenant pour développer de nouvelles méthodes d'optimisation de programmes et de les intégrer dans des compilateurs réels, qui permettront une analyse plus précise des codes source et une meilleure optimisation. À moyen et long terme, il convient d'étendre les algorithmes de dénombrement de points entiers pour traiter des problèmes plus généraux prenant en compte des contraintes non affines. Maslov et Pugh [102], Rabl [119] et Clauss et al. [38] ont déjà tenté de traiter quelques cas particuliers, mais la manipulation des contraintes non affines, telles que les contraintes polynomiales, reste un problème ouvert.

Bibliographie

[1] S. Anantharaman and S. Pande. Compiler optimizations for real time execution of loops on limited memory embedded systems. In *The 19th IEEE Systems Symposium (RTSS98)*, 1998.

[2] C. Ancourt and F. Irigoin. Scanning polyhedra with DO loops. In *Proceedings of the 3rd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pages 39–50, Williamsburg, VA, April 1991.

[3] J. M. Anderson, S. P. Amarasinghe, and M. S. Lam. Data and computation transformations for multiprocessors. In *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'95*, pages 166–178, Santa Barbara, California, 1995.

[4] J. M. Anderson, S. P. Amarasinghe, and M. S. Lam. Data and computation transformations for multiprocessors. In *PPOPP '95 : Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 166–178, New York, NY, USA, 1995. ACM Press.

[5] F. Aurenhammer and R. Klein. *Handbook of computational geometry.* North-Holland Publishing Co., Amsterdam, The Netherlands : North-Holland. pp. 201-290, 2000.

[6] E. Ayguade, J. Garcia, M. Girones, and J. Labarta. Detecting and using affinity in an automatic data distribution tool. *Lecture Notes in Computer Science*, 892 :61–75, 1995.

[7] D. F. Bacon, J.-H. Chow, D. ching R. Ju, K. Muthukumar, and V. Sarkar. A compiler framework for restructuring data declarations to enhance cache and TLB effectiveness. In *Proceedings of CASCON'94*, pages 270–282, Toronto, Canada, Oct. 1994.

[8] F. Balasa, F. Catthoor, and H. De Man. Background memory area estimation for multidimensional signal processing systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(2) :157–172, June 1995.

[9] M. W. Baldoni-Silva, M. Beck, C. Cochet, and M. Vergne. Volume computation for polytopes and partition functions for classical root systems. *Discrete & Computational Geometry*, 35(4) :551–595, 2006.

[10] C. Bartzis and T. Bultan. Automata-based representations for arithmetic constraints in automated verification. In *7th International Conference on Implementation and Application of Automata*, pages 282–288, Tours, France, July 2002.

[11] A. Barvinok. *A Course in Convexity*. American Mathematical Society, 2002.

[12] A. Barvinok and J. Pommersheim. An algorithmic theory of lattice points in polyhedra. *New Perspectives in Algebraic Combinatorics*, 38 :91–147, 1999.

[13] A. Barvinok and K. Woods. Short rational generating functions for lattice point problems. *Journal of the American Mathematical Society*, 16 :657–979, 2003.

[14] A. I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res.*, 19(4) :769–779, 1994.

[15] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 7–16, Juan-les-Pins, september 2004.

[16] M. Beck. Counting lattice points by means of the residue theorem. *Ramanujan Journal*, 4(3) :299–310, 2000.

[17] M. Beck. The partial-fractions method for counting solutions to integral linear systems. *Discrete Comp. Geom.*, 32 :437–446, 2004. (special issue in honor of Louis Billera).

[18] W. H. Beyer. *CRC Standard Mathematical Tables*. CRC Press, 1981.

[19] K. Beyls. *Software Methods to Improve Data Locality and Cache Behavior*. PhD thesis, Ghent University, 2004.

[20] K. Beyls and E. D'Hollander. Cache remapping to improve the performance of tiled algorithms. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, *Proceedings of the 6th International Euro-Par Conference*, number 1900, pages 998–1007, Munich, 8 2000. Springer.

[21] K. Beyls and E. D'Hollander. Compiler generated multithreading to alleviate memory latency. *Journal of Universal Computer Science*, 6(10) :968–993, 10 2000.

[22] K. Beyls and E. D'Hollander. Reuse distance as a metric for cache behavior. In *IASTED conference on Parallel and Distributed Computing and Systems 2001 (PDCS01)*, pages 617–662, 2001.

[23] K. Beyls and E. D'Hollander. Reuse distance-based cache hint selection. In *Proceedings of the 8th International Euro-Par Conference*, pages 265–274, 2002.

[24] K. Beyls and E. D'Hollander. Generating cache hints for improved program efficiency. *Journal of Systems Architecture*, 51(4) :223–250, 2005.

[25] A. J. C. Bik. *Compiler Support for Sparse Matrix Computations*. PhD thesis, University of Leiden, The Netherlands, 1996.

[26] B. Boigelot and L. Latour. Counting the solutions of Presburger equations without enumerating them. *Theoretical Computer Science*, 313(1) :17–29, Feb. 2004.

[27] P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-ultimate tiling ? *Integration, the VLSI Journal*, 17 :33–51, 1994.

[28] P. Boulet and X. Redon. Communication pre-evaluation in HPF. In *EURO-PAR'98*, volume 1470 of *LNCS*, pages 263–272. Springer Verlag, 1998.

[29] V. Braberman, D. Garbervetsky, and S. Yovine. On synthesizing parametric specifications of dynamic memory utilization. Technical report, Oct. 2003.

[30] M. Brion. Points entiers dans les polyèdres convexes. *Ann. Sci. École Norm. Sup. (4)*, 21(4) :653–663, 1988.

[31] D. Callahan, K. Kennedy, and A. Porterfield. Software prefetching. In *ASPLOS-IV : Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 40–52, New York, NY, USA, 1991. ACM Press.

[32] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *International Conference on Supercomputing*, pages 444–453, 1999.

[33] S. Chatterjee, E. Parker, P. J. Hanlon, and A. R. Lebeck. Exact analysis of the cache behavior of nested loops. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, pages 286–297. ACM Press, 2001.

[34] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5(2) :228–133, 1965.

[35] T. Chilimbi, J. Larus, and M. Hill. Improving pointer-based codes through cache-conscious data placement, 1998.

[36] T. M. Chilimbi and M. Hirzel. Dynamic hot data stream prefetching for general-purpose programs. In *PLDI '02 : Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 199–209, New York, NY, USA, 2002. ACM Press.

[37] P. Clauss. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials : Applications to analyse and transform scientific programs. In *Proceedings of the 10th International Conference on Supercomputing, ICS'96*, May 1996.

[38] P. Clauss, F. J. Fernandez, D. Gabervetsky, and S. Verdoolaege. Symbolic polynomial maximization over convex sets and its application to memory requirement estimation. Technical Report 06-04, Université Louis Pasteur, October 2006.

[39] P. Clauss and V. Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, 19(2) :179–194, July 1998.

[40] S. Coleman and K. S. McKinley. Tile size selection using cache organization and data layout. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 279–290, La Jolla, CA, June 1995.

[41] P. D'Alberto, A. Veidembaum, A. Nicolau, and R. Gupta. Static analysis of parameterized loop nests for energy efficient use of data caches. In *Workshop on Compilers and Operating Systems for Low Power (COLP01)*, Sept. 2001.

[42] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin elimination and its dual. *J. Comb. Theory, Ser. A*, 14(3) :288–297, 1973.

[43] A. Darte. On the complexity of loop fusion. In *IEEE PACT*, pages 149–157, 1999.

[44] J. A. De Loera, D. Haws, R. Hemmecke, P. Huggins, B. Sturmfels, and R. Yoshida. Short rational functions for toric algebra and applications, 2004.

[45] J. A. De Loera, D. Haws, R. Hemmecke, P. Huggins, J. Tauzer, and R. Yoshida. A user's guide for LattE v1.1, Nov. 2003. software package `LattE` is available at `http ://www.math.ucdavis.edu/~latte/`.

[46] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4) :1273–1302, 2004.

[47] P. Diaconis and A. Gangolli. Rectangular arrays with fixed margins. *Discrete Probability and Algorithms (Minneapolis, MN, 1993), 15-41, IMA Volumes in Mathematics and its Applications, 72, Springer, New York*, 1995.

[48] C. Ding and K. Kennedy. Improving effective bandwidth through compiler enhancement of global cache reuse. *J. Parallel Distrib. Comput.*, 64(1) :108–134, 2004.

[49] M. Dyer and R. Kannan. On barvinok's algorithm for counting lattice points in fixed dimension. *Math. Oper. Res.*, 22(3) :545–549, 1997.

[50] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer-Verlag New York, Inc., 1987.

[51] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen. Simultaneous multithreading : A platform for next-generation processors. *IEEE Micro*, 17(5) :12–19, 1997.

[52] E. Ehrhart. Sur les polyèdres rationnels homothétiques à n dimensions. *C.R. Acad. Sci. Paris*, 254 :616–618, 1962.

[53] E. Ehrhart. Polynômes arithmétiques et méthode des polyèdres en combinatoire. *International Series of Numerical Mathematics*, 35, 1977.

[54] P. Feautrier. Parametric integer programming. *Operationnelle/Operations Research*, 22(3) :243–268, 1988.

[55] P. Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1) :23–53, 1991.

[56] P. Feautrier. Automatic parallelization in the polytope model. In *The Data Parallel Programming Model*, pages 79–103, 1996.

[57] P. Feautrier, J. Collard, and C. Bastoul. Solving systems of affine (in)equalities. Technical report, PRiSM, Versailles University, 2002.

[58] F. Fernandez and P. Quinton. Extension of chernikova's algorithm for solving general mixed linear programming problems. Technical Report 437, IRISA, Rennes, France, 1988.

[59] J. Ferrante, V. Sarkar, and W. Thrash. On estimating and enhancing cache effectiveness. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Proceedings of the Fourth International Workshop on Languages and Compilers for ParallelComputing*, volume 589 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Aug. 1991.

[60] S. E. Fienberg, U. E. Makov, M. M. Meyer, and R. J. Steele. Computing the exact conditional distribution for a multi-way contingency table on its marginal totals. In *Data Analysis From Statistical Foundations, Huntington, NY*, pages 145–166, 2001.

[61] B. Franke and M. O'Boyle. Array recovery and high-level transformations for DSP applications. *ACM Transactions on Embedded Computing Systems*, 2(2) :132–162, May 2003.

[62] Free Software Foundation, Inc. GMP. Available from `ftp ://ftp.gnu.org/gnu/gmp`.

[63] K. Fukuda. Frequently asked questions in polyhedral computation, October 2000. http ://www.ifor.math.ethz.ch/ fukuda/polyfaq/polyfaq.html.

[64] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations : a compiler framework for analyzing and tuning memory behavior. *ACM Transactions on Programming Languages and Systems*, 21(4) :703–746, 1999.

[65] A. J. Goldmann. Resolution and separation theorems for polyhedral convex sets. In Kuhn and Tucker, editors, *Linear Inequalities and Related Systems*. Princeton University Press, Princeton, NJ, 1956.

[66] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springeri-Verlag, Berlin, 1993.

[67] A. Größlinger. *Extending the Polyhedron Model to Inequality Systems with Nonlinear Parameters using Quantifier Elimination*. PhD thesis, Universität Passau, 2003.

[68] F. Hannig and J. Teich. Design space exploration for massively parallel processor arrays. In *Proceedings of the Sixth International Conference on Parallel Computing Technologies*, volume 2127 of *Lecture Notes in Computer Science*, pages 51–65, 2001.

[69] F. Heine and A. Slowik. Volume driven data distribution for NUMA-machines. In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 415–424, 2000.

[70] P. Held. Hipars : a tool for automatic conversion of nested loop programs into single assignment programs. Technical report, Dept. Electrical Engineering, Delft University of Technology, 1994.

[71] J. Hennessy. Back to the future : Time to return to some long standing problems in computer systems ? In *The ACM Federated Computer Research Conference*, May 1999.

[72] P. Henrici. *Applied and computational complex analysis*. Wiley classics library. Wiley New York (N.Y.), 1974.

[73] P. Jain, S. Devadas, D. Engels, and L. Rudolph. Software-assisted replacement mechanisms for embedded systems. In *International Conference on Computer Aided Design*, pages 119–126, nov 2001.

[74] M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, and J. Ramanujam. A linear algebra framework for automatic determination of optimal data layouts. *IEEE Transactions on Parallel and Distributed Systems*, 10(2) :115–135, 1999.

[75] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of ACM*, 14(3) :563–590, July 1967.

[76] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. The Omega calculator and library. Technical report, University of Maryland, Nov. 1996.

[77] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. The Omega Library. Technical report, Institue for advanced computer studies, University of Maryland, College Park, 1996.

[78] K. Kennedy and U. Kremer. Automatic data layout for distributed-memory machines. *ACM Trans. Program. Lang. Syst.*, 20(4) :869–916, 1998.

[79] B. Kienhuis, E. Rijpkema, and E. Deprettere. Compaan : deriving process networks from matlab for embedded signal processing architectures. In *CODES '00 : Proceedings of the eighth international workshop on Hardware/software codesign*, pages 13–17, New York, NY, USA, 2000. ACM Press.

[80] A. N. Kirillov. Ubiquity of Kostka polynomials. *In Physics and Combinatorics, Proceedings Nagoya*, 1999. Edited by A. N. Kirillov, A. Tsuchiya and H. Umemura, World Scientific, 2001.

[81] P. G. Kjeldsberg, F. Catthoor, and E. J. Aas. Data dependency size estimation for use in memory optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits a nd Systems*, 22(7), July 2003.

[82] N. Klarlund and A. Møller. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, January 2001.

[83] G. Kreisel and J. L. Krevine. *Elements of Mathematical Logic*. The Netherlands : North-Holland Publishing, 1967.

[84] D. J. Kuck. *The Structure of Computers and Computations, volume 1*. John Wiley and Sons, New York, 1978.

[85] M. D. Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. In *ASPLOS-IV : Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 63–74, New York, NY, USA, 1991. ACM Press.

[86] H. Le Verge. A note on chernikova's algorithm. Technical Report 635, IRISA, Rennes, France, 1992.

[87] C. W. Lee. Subdivisions and triangulations of polytopes. pages 271–290, 1997.

[88] C. Lengauer. Loop parallelization in the polytope model. In *International Conference on Concurrency Theory*, pages 398–416, 1993.

[89] A. K. Lenstra, J. Hendrik W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261 :515–534, 1982. URL : `http ://cr.yp.to/bib/entries.html#1982/lenstra-lll`.

[90] D. Lepelley, A. Louichi, and H. Smaoui. On Ehrhart polynomials and probability calculations in voting theory. Technical report, Center for Research in Economics and Management (CREM), University of Rennes 1, University of Caen and CNRS, 2006. http ://ideas.repec.org/p/tut/cremwp/200610.html.

[91] E. Lewis, C. Lin, and L. Snyder. The implementation and evaluation of fusion and contraction in array languages. In *Proceedings of the SIGPLAN '98 Conference on Programming Language Design and Implementation*, Montreal, Canada, 1998.

[92] Z. Li and Y. Song. Automatic tiling of iterative stencil loops. *ACM Trans. Program. Lang. Syst.*, 26(6) :975–1028, 2004.

[93] B. Lisper. Fully automatic, parametric worst-case execution time analysis. In J. Gustafsson, editor, *Proc. Third International Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 77–80, Porto, July 2003.

[94] V. Loechner. *Contribution à l'étude des polyèdres paramétrés et applications en parallélisation automatique*. PhD thesis, Université Louis Pasteur, Strasbourg, 1997. http ://icps.u-strasbg.fr/pub-97/.

[95] V. Loechner. Polylib : A library for manipulating parameterized polyhedra. Technical report, LSIIT - ICPS UMR7005 Univ. Louis Pasteur-CNRS, Mar. 1999.

[96] V. Loechner, B. Meister, and P. Clauss. Data sequence locality : A generalization of temporal locality. *Lecture Notes in Computer Science*, 2150 :262+, 2001.

[97] V. Loechner, B. Meister, and P. Clauss. Precise data locality optimization of nested loops. *Journal of Supercomputing*, 21(1) :37–76, 2002.

[98] V. Loechner and D. K. Wilde. Parameterized polyhedra and their vertices. *International Journal of Parallel Programming*, 25(6) :525–549, Dec. 1997.

[99] C.-K. Luk and T. C. Mowry. Automatic compiler-inserted prefetching for pointer-based applications. *IEEE Trans. Comput.*, 48(2) :134–141, 1999.

[100] P. A. MacMahon. *Combinatorial Analysis*, volume 2. Chelsea, New York, 1960. reprint of 1915 edition.

[101] N. Manjikian and T. S. Abdelrahman. Fusion of loops for parallelism and locality. *IEEE Transactions on Parallel and Distributed Systems*, 8(2) :193–209, 1997.

[102] V. Maslov and W. Pugh. Simplifying polynomial constraints over integers to make dependence analysis more precise. In *ONPAR 94 - VAPP VI, International Conference on Parallel and Vector Processing*, Sept. 1994.

[103] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst.*, 18(4) :424–453, 1996.

[104] B. Meister. Projecting periodic polyhedra for loop nest analysis. In *Proceedings of the 11th Workshop on Compilers for Parallel Computers (CPC 04), Kloster Seeon, Germany*, pages 13–24, July 2004.

[105] H. Minkowski. *Geometrie der Strahlen (Erste Lieferung)*. Teubner, Leipzig (reprinted : Chelsea, New York, 1953), 1896.

[106] H. Minkowski. *Gesammelte Abhandlungen, Band I, II*. Teubner, Leipzig (reprinted : Chelsea, New York, 1967), 1911.

[107] K. Morin-Allory. *Vérification Formelle dans le Modèle Polyédrique*. PhD thesis, October 2004.

[108] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. *Contributions to the Theory of Games*. Kuhn and Tucker, Eds., Princeton University Press, Princeton, NJ, (Reprinted in : S. Motzkin : Selected Papers, Cantor, Gordon, Rotschild, Eds., Birkhäuser, Boston, 1983, pp. 81–103), 1953.

[109] T. C. Mowry, M. S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, volume 27, pages 62–73, New York, NY, 1992. ACM Press.

[110] A. Nijehuis and H. Wilf. Representations of integers by linear forms in nonegative integers. *Journal of Number Theory*, 4 :98–106, 1972.

[111] S. P. K. Nookala and T. Risset. A library for Z-polyhedral operations. Technical report, 1330, Irisa, 2000.

[112] B. Nootaert. Een verbeterde methode voor de berekening van Ehrhart-polynomen. Master's thesis, Ghent University, 2004.

[113] E. Parker and S. Chatterjee. An automata-theoretic algorithm for counting solutions to Presburger formulas. In *Compiler Construction 2004*, volume 2985 of *Lecture Notes in Computer Science*, pages 104–119, Apr. 2004.

[114] W. Pugh. The Omega test : a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 4–13. ACM Press, 1991.

[115] W. Pugh. Counting solutions to Presburger formulas : how and why. In *SIGPLAN Conference on Programming Language Design and Implementation (PLDI'94)*, pages 121–134, 1994.

[116] W. Pugh and D. Wonnacott. Experiences with constraint-based array dependence analysis. In *Principles and Practice of Constraint Programming*, pages 312–325, 1994.

[117] F. Quilleré and S. Rajopadhye. Optimizing memory usage in the polyhedral model. In *ACM Transactions on Programming Languages and Systems (TOPLAS),Volume 22, Issue 5*, pages 773–815, Sept. 2000.

[118] P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra using a canonical representation. *Parallel Processing Letters*, 7(2) :181–194, 1997.

[119] T. Rabl. *Volume Calculation and Estimation of Parameterized Integer Polytopes.* PhD thesis, Universität Passau, 2006.

[120] J. Ramanujam, J. Hong, M. Kandemir, and A. Narayan. Reducing memory requirements of nested loops for embedded systems. In *DAC '01 : Proceedings of the 38th conference on Design automation*, pages 359–364, New York, NY, USA, 2001. ACM Press.

[121] G. Rivera and C.-W. Tseng. Data transformations for eliminating conflict misses. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 38–49, Montreal, Canada, June 1998.

[122] G. Rivera and C.-W. Tseng. Eliminating conflict misses for high performance architectures. In *International Conference on Supercomputing*, pages 353–360, Melbourne, Australia, July 1998.

[123] S. Rosen. Electronic computers : A historical survey. *ACM Computing Reviews*, 1(1) :7–36, 1969.

[124] D. Rubin. Vertex generation and cardinality constrained linear programs. *Operations Research*, 23(3) :555–565, May 1975.

[125] J. R. Schmidt and A. M. Bincer. The konstant partition function for simple lie algebras. *J. Mathematical physics*, 25(8) :2367–2373, 1984.

[126] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1986.

[127] R. Seghir. Dénombrement des points entiers de l'union et de l'image des polyèdres paramétrés. Master's thesis, June 2002.

[128] R. Seghir and V. Loechner. Memory optimization by counting points in integer transformations of parametric polytopes. In *In Proceedings of International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, CASES 2006, Seoul, Korea*, pages 74–82, October 2006.

[129] R. Seghir, V. Loechner, and B. Meister. Minimizing memory strides using integer transformations of parametric polytopes. Technical report, LSIIT - ICPS UMR7005 ULP-CNRS, January 2006. http ://icps.u-strasbg.fr.

[130] R. Seghir, S. Verdoolaege, K. Beyls, and V. Loechner. Analytical computation of Ehrhart polynomials and its application in compile-time generated cache hints. Technical report, LSIIT-ICPS UMR7005 ULP-CNRS, Strasbourg, Feb. 2004.

[131] V. Shoup. NTL. Available from `http ://www.shoup.net/ntl/`.

[132] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 215–228, 1999.

[133] R. P. Stanley. *Enumerative combinatorics*, volume 1. Cambridge University Press, Cambridge, 1997. With a foreword by Gian-Carlo Rota, Corrected reprint of the 1986 original.

[134] B. Sturmfels. On vector partition functions. *J. Comb. Theory Ser. A*, 72(2) :302–309, 1995.

[135] E. Su, A. Lain, S. Ramaswamy, D. J. Palermo, E. Hodges IV, and P. Banerjee. Advanced compilation techniques in the PARADIGM compiler for distributed-memory multicomputers. In *International Conference on Supercomputing*, pages 424–433, 1995.

[136] N. Tawbi. Estimation of nested loops execution time by integer arithmetic in convex polyhedra. In *Proceedings of the 8th International Symposium on Parallel Processing*, pages 217–221, Washington, DC, USA, 1994. IEEE Computer Society.

[137] N. Tawbi and P. Feautrier. Processor allocation and loop scheduling on multiprocessor computers. In *1992 International Conference on Supercomputing*, pages 63–71. ACM, July 1992.

[138] A. Turjan, B. Kienhuis, and E. Deprettere. A compile time based approach for solving out-of-order communication in Kahn process networks. In *IEEE 13th International Conference on Aplication-specific Systems, Architectures and Processors (ASAP'2002)*, July 2002.

[139] A. Turjan, B. Kienhuis, and E. Deprettere. Translating affine nested-loop programs to process networks. In *CASES '04 : Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 220–229, New York, NY, USA, 2004. ACM Press.

[140] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun. A modified approach to data cache management. In *MICRO 28 : Proceedings of the 28th annual international symposium on Microarchitecture*, pages 93–103, Los Alamitos, CA, USA, 1995. IEEE Computer Society Press.

[141] T. Ungerer, B. Robič, and J. Šilc. A survey of processors with explicit multithreading. *ACM Comput. Surv.*, 35(1) :29–63, 2003.

[142] S. Verdoolaege. *Incremental loop transformations and enumeration of parametric sets*. PhD thesis, April 2005.

[143] S. Verdoolaege. barvinok : user guide, August 2006. http ://www.kotnet.org/škimo/barvinok/barvinok.pdf.

[144] S. Verdoolaege, K. Beyls, M. Bruynooghe, and F. Catthoor. Experiences with enumeration of integer projections of parametric polytopes. In R. Bodik, editor, *Compiler Construction : 14th International Conference*, volume 3443, pages 91–105, Edinburgh, 3 2005. Springer.

[145] S. Verdoolaege, M. Bruynooghe, G. Janssens, and F. Catthoor. Multi-dimensional incremental loop fusion for data locality. In D. Martin, editor, *IEEE 14th International Conference on Application-specific Systems, Architectures and Processors*, The Hague, The Netherlands, June 2003.

[146] S. Verdoolaege, H. Nikolov, and T. Stefanov. Improved derivation of process networks. In *4th Workshop on Optimization for DSP and Embedded Systems, ODES-4*, Mar. 2006.

[147] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe. Counting integer points in parametric polytopes using barvinok's rational functions. *Algorithmica*. article to appear.

[148] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe. Analytical computation of Ehrhart polynomials : Enabling more compiler analyses and optimizations. In *Proceedings of International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, Washington D.C.*, pages 248–258, Sept. 2004.

[149] S. Verdoolaege and K. Woods. Counting with rational generating functions, 2005. http ://arxiv.org/abs/math/0504059.

[150] V. Weispfenning. Complexity and uniformity of elimination in presburger arithmetic. In *ISSAC '97 : Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, pages 48–53, New York, NY, USA, 1997. ACM Press.

[151] D. K. Wilde. A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France, 1993.
http ://www.irisa.fr/EXTERNE/bibli/pi/pi785.html.

[152] M. E. Wolf and M. S. Lam. A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'91)*, pages 30–44, June 1991.

[153] M. Wolfe. More iteration space tiling. In *Supercomputing '89 : Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 655–664, New York, NY, USA, 1989. ACM Press.

[154] R. Yoshida. *Barvinok's Rational Functions : Algorithms and Applications to Optimization, Statistics, and Algebra*. PhD thesis, UC-Davis, 2004.

[155] Y. Zhao and S. Malik. Exact memory size estimation for array computations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(5) :517–521, October 2000.

[156] H. Zhu, I. I. Luican, and F. Balasa. Memory size computation for multimedia processing applications. In *Proceedings of 11th Asia and South Pacific Design Automation Conference*, pages 802–807, Jan. 2006.

Résumé

Le modèle polyédrique est un formalisme utilisé en optimisation automatique de programmes. Il permet notamment de représenter les itérations et les références à des tableaux, dans des nids de boucles affines, par des points à coordonnées entières de polyèdres bornés, ou $\mathbb{Z}$-polytopes (paramétrés). Dans cette thèse, trois nouveaux algorithmes de dénombrement ont été développés : des points entiers dans un $\mathbb{Z}$-polytope paramétré, dans une union non disjointe de $\mathbb{Z}$-polytopes paramétrés et dans leurs images par des fonctions affines. Le résultat de ces dénombrements est donné par un ou plusieurs polynômes multivariable à coefficients périodiques. Ces polynômes, connus sous le nom de quasi-polynômes d'Ehrhart, sont définis sur des sous-ensembles de valeurs des paramètres, dits domaines de validité.

De nombreuses méthodes d'analyse et d'optimisation de nids de boucles affines font appel à ces algorithmes. Nous les avons en particulier appliqués à la linéarisation de tableaux, dont l'objectif est la compression mémoire et l'amélioration de la localité spatiale. Outre l'optimisation de programmes, les algorithmes proposés ont des applications dans bien d'autres domaines, tels que les mathématiques et l'économie.

Mots-clés : modèle polyédrique, nids de boucles, dénombrement de points entiers de $\mathbb{Z}$-polytopes paramétrés, formules de Presburger, quasi-polynômes d'Ehrhart, optimisation de la mémoire.

Abstract

The polyhedral model is a well-known framework in the field of automatic program optimization. Iterations and array references in affine loop nests are represented by integer points in bounded polyhedra, or (parametric) $\mathbb{Z}$-polytopes. In this thesis, three new counting algorithms have been developed: counting integer points in a parametric $\mathbb{Z}$-polytope, in a union of parametric $\mathbb{Z}$-polytopes and in their images by affine functions. The result of such a counting is given by one or many multivariate polynomials in which the coefficients may be periodic numbers. These polynomials, known as Ehrhart quasi-polynomials, are defined on sub-sets of the parameter values called validity domains or chambers.

Many affine loop nest analysis and optimization methods require such counting algorithms. We applied them in array linearization which achieves memory compression and improves spatial locality of accessed data. Besides program optimization, the proposed algorithms have many other applications, as in mathematics and economics.

Keywords: polyhedral model, loop nests, counting integer points in parametric $\mathbb{Z}$-polytopes, Presburger formulas, Ehrhart quasi-polynomials, memory optimization.