Quasi-polynomials, Z-polyhedra and parametric integer linear programming

Rui-Juan Jing[1][0000−0001−6932−8862], Yuzhuo Lei[2][0009−0005−7010−7932], Marc Moreno Maza[2][0000−0003−3011−0756], and Chirantan Mukherjee[2][0000−0002−6692−3305]

[1] School of Mathematical Science, Jiangsu University, Zhenjiang, China
rjing@ujs.edu.cn
[2] Ontario Research Center for Computer Algebra, University of Western Ontario, London, Ontario, Canada
{ylei83, mmorenom, cmukher}@uwo.ca

A problem instance in parametric integer linear programming (PILP) requests the maximum (or minimum) value of an affine objective function defined over a parametric $\mathbb{Z}$-polyhedron (that is, the intersection of a parametric polyhedron and an integer lattice). Of course, this maximum depends on the parameters and computing it generally leads to a partition of the parameter space so that, above each part of the partition, the maximum is given by a quasi-polynomial.

Implemented algorithms for PILP rely either on an adaptation of the simplex method due to Paul Feautrier, or an adaptation of Barvinok's algorithm proposed by Sven Verdoolaege. With each part of the partition, these algorithms provide a sample point (that is, a tuple of values of the variables) realizing the maximum of the objective function.

In this paper, we present a novel algorithm to solve PILP instances which enhance the previous approaches as follows. Indeed, for each part of the partition, we compute the locus of all points realizing the maximum of the objective function.

1 Introduction

When solving systems of polynomial equations and inequalities, the task of computing their solutions with integer coordinates is a much harder problem than that of computing their real solutions or that of computing all their solutions. In fact, in the presence of non-linear constraints, this task may simply become an undecidable problem [12, 14]. However, studying the integer solutions of linear systems of equations and inequalities is of practical importance in various areas of scientific computing. Two such areas are *combinatorial optimization* (in particular, integer linear programming) and *compiler optimization* (in particular, the analysis, transformation, and scheduling of nested loops in computer programs), where a variety of algorithms solve questions related to the points with integer coordinates in a given polyhedron.

As a driving example, we consider the problem of *array bound checking*, that is, checking whether all array references in a for-loop nest (and more generally in a computer program) are within their declared ranges. This problem admits

heuristic solutions [13, 15]. However, it remains hard in general, as we shall see now. Consider d nested for-loops as shown below.

for $(\mathbf{i}_1 \cdots ; \ \cdots ; \ \mathbf{i}_1{+}{+})$ **do**
  $\cdots$
  for $(\mathbf{i}_d \cdots ; \cdots ; \ \mathbf{i}_d{+}{+})$ **do**
    $\tilde{A}[f_1] \cdots [f_\delta] \leftarrow \cdots$
  end for
  $\cdots$
end for

Let $\mathbf{i}_1, \ldots, \mathbf{i}_d$ be the respective loop counters of these for-loops. Without loss of generality, we assume that $\mathbf{i}_1, \ldots, \mathbf{i}_d$ take non-negative integer values such that

$$L \begin{pmatrix} \mathbf{i}_1 \\ \vdots \\ \mathbf{i}_d \end{pmatrix} \ \leq \ \begin{pmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_d \end{pmatrix}, \tag{1}$$

where L is a lower-triangular full-rank matrix over $\mathbb{Z}$, known at compile time, and $\mathbf{r}_1, \ldots, \mathbf{r}_d$ are integer-valued parameters, known only at execution time. The integer tuples $(\mathbf{i}_1, \ldots, \mathbf{i}_d)$ satisfying Equation (1) form the so-called *iteration domain*, that we denote by $\mathcal{D}$. These integer tuples are the integer points of a rational polytope P in $\mathbb{Q}^d$, that is, the integer hull of P.

The array $\tilde{A}$ is an $\mathbf{M}_1 \times \cdots \times \mathbf{M}_\delta$-array, where $\mathbf{M}_1, \ldots, \mathbf{M}_\delta$ are positive integers known only at execution time. Note that the number δ of dimensions of the array $\tilde{A}$ may not be equal to the number d of nested for-loops. A classical example where $d \neq \delta$ holds is dense matrix multiplication, in particular the tiled version of dense matrix multiplication where $d = 6$ and $\delta = 2$. Finally, we assume that the array reference functions $f_1, \ldots, f_\delta$ are affine functions of $\mathbf{i}_1, \ldots, \mathbf{i}_d$ and that $f_1, \ldots, f_\delta$ are known at compile time. We can now express the fact that all references to the array $\tilde{A}$ are within declared ranges:

$$\forall (\mathbf{i}_1, \ldots, \mathbf{i}_d) \in \mathcal{D} \quad (0 \leq f_1 \leq \mathbf{M}_1 \ \wedge \ \cdots \ \wedge \ 0 \leq f_\delta \leq \mathbf{M}_\delta) \, . \tag{2}$$

Because the coefficients of the matrix L are literals (that is, actual numbers and not symbols) and because $f_1, \ldots, f_\delta$ are affine functions of $\mathbf{i}_1, \ldots, \mathbf{i}_d$, Formula (2) is a Presburger formula. Therefore, at compile time, one can apply quantifier elimination (QE) techniques such as those of [8] in order to replace Formula (2) with quantifier-free conditions on the parameters $\mathbf{r}_1, \ldots, \mathbf{r}_d$ and $\mathbf{M}_1, \ldots, \mathbf{M}_\delta$.

However, one may want to consider alternative approaches to deal with Formula (2). Indeed, eliminating universal quantifiers is done through *double negation* (that is, De Morgan's laws) unless some specific heuristics apply. As a result, eliminating a sequence of universal quantifiers usually produces a very large number of cases to examine. Instead, one may compute, for $i = 1 \cdots \delta$

$$E_i := \begin{matrix} \min f_i((\mathbf{i}_1, \ldots, \mathbf{i}_d)) \\ (\mathbf{i}_1, \ldots, \mathbf{i}_d) \in \mathcal{D} \end{matrix} \quad \text{and} \quad F_i := \begin{matrix} \max f_i((\mathbf{i}_1, \ldots, \mathbf{i}_d)) \\ (\mathbf{i}_1, \ldots, \mathbf{i}_d) \in \mathcal{D} \end{matrix} \tag{3}$$

and replace Formula (2) with

$$((0 \leq E_1) \wedge (F_1 \leq \mathbf{M}_1) \ \wedge \ \cdots \ \wedge \ (0 \leq E_\delta) \wedge (F_\delta \leq \mathbf{M}_\delta)) \, . \tag{4}$$

In other words, one can replace our QE problem with a number of problem instances from parametric integer linear programming (PILP), given by the computation of $E_1, \ldots, E_\delta$, $F_1, \ldots, F_\delta$.

A PILP problem instance requests the maximum (or minimum) value of an affine objective function defined over a parametric $\mathbb{Z}$-polyhedron (that is, the intersection of a parametric polyhedron and an integer lattice). Of course, this maximum (or minimum) depends on the parameters and computing it generally leads to a partition of the parameter space so that, above each part of the partition, the maximum is given by a quasi-polynomial.

Implemented algorithms for PILP rely either on an adaptation of the simplex method due to Paul Feautrier [5], or an adaptation of Barvinok's algorithm [1] proposed by Sven Verdoolaege [2, 16, 17]. With each part of the partition, these algorithms provide a sample point (that is, a tuple of values of the variables) realizing the maximum of the objective function.

In this paper, we present a novel algorithm to solve PILP instances which enhance the previous approaches as follows. For each part of the partition, we compute the *locus* of all points realizing the maximum of the objective function.

Our code is publicly available github.com/MarcMorenoMaza/PolyhedralSets/tree/main/QuantifierEliminationOverTheIntegers.

2 Quasi-polynomials and $\mathbb{Z}$-polyhedral sets

Let $s \in \mathbb{Z}[X]$ be a univariate polynomial with coefficients in $\mathbb{Z}$. Let m be a positive integer and let $f_0, \ldots, f_{m-1} \in \mathbb{Q}[X]$ be m univariate polynomials with coefficients in $\mathbb{Q}$. Then, the *univariate quasi-polynomial* $Q(s, m, f_0, \ldots, f_{m-1})$ is the function defined on $\mathbb{Z}$, and taking values in $\mathbb{Q}[X]$, so that, for $0 \leq i < m$, we have

$$Q(s, m, f_0, \ldots, f_{m-1})(k) \;=\; f_i \quad \Longleftrightarrow \quad s(k) \equiv i \mod m. \tag{5}$$

The polynomial s, the integer m, and the polynomials $f_0, \ldots, f_{m-1}$ are called the *switch*, the *modulo*, and the *constituents* of the quasi-polynomial given by $Q(s, m, f_0, \ldots, f_{m-1})$. Two quasi-polynomials are said to be *equal* if they have the same switch, the same modulo, and the same constituents (in the same order).

Given $s \in \mathbb{Z}[X]$ and a positive integer m, we denote by $Q(s, m)$ the quasi-polynomial $Q(s, m, f_0, \ldots, f_{m-1})$ where $f_i = i$, for $0 \leq i < m$. Moreover, if $s = X$, we simply write $X \mod m$ instead of $Q(X, m)$.

We fix a positive integer m. The following two results follow immediately from the previous definitions.

Proposition 1. *For every quasi-polynomial $q_1 := Q(s, m, f_0, \ldots, f_{m-1})$, there exists a quasi-polynomial $q_2 := Q(X, m, g_1, \ldots, g_{m-1})$, such that we have $q_1 = q_2$. We denote by $\mathbb{Q}[X \mod m]$ the set of all univariate quasi-polynomials of the form $Q(X, m, f_0, \ldots, f_{m-1})$.*

Proposition 2. *For $q_1 := Q(X, m, f_0, \ldots, f_{m-1})$ and $q_2 := Q(X, m, g_0, \ldots, g_{m-1})$ in $\mathbb{Q}[X \mod m]$, we define the sum $q_1 + q_2$ and the product $q_1 \cdot q_2$ as follows*

$$q_1 + q_2 = (X, m, f_0 + g_0, \ldots, f_{m-1} + g_{m-1}) \tag{6}$$

and

$$q_1 \cdot q_2 = (X, m, f_0 g_0, \ldots, f_{m-1} g_{m-1}). \tag{7}$$

Equipped with this addition and this multiplication, the set $\mathbb{Q}[X \mod m]$ is a commutative ring. Unless $m = 1$, this ring has zero divisors and is isomorphic to the direct product of m copies of $\mathbb{Q}[X]$.

A *$\mathbb{Z}$-polyhedron* of $\mathbb{Z}^d$ is the intersection, in $\mathbb{Z}^d$, of a polyhedron $P \subseteq \mathbb{Q}^d$ and a lattice $L \subseteq \mathbb{Z}^d$; we denote it by $\mathbb{Z}\mathsf{Polyhedron}(P, L)$. Hence, this is a subset of the integer points of P. This subset may be empty. However, there is an algorithm [11], called $\mathsf{IntegerPointDecomposition}$, which decomposes any $\mathbb{Z}$-polyhedron into *normalized* $\mathbb{Z}$-polyhedra. Denote by $x_1 < x_2 < \cdots < x_d$ the ordered coordinates of $\mathbb{Z}^d$. We say that $\mathbb{Z}\mathsf{Polyhedron}(P, L)$ is *normalized* if it is non-empty and P is given by a system of linear inequalities of the form

$$\begin{cases} a_0 \leq x_1 \leq b_0 \\ a_1 \leq x_2 \leq b_1 \\ \quad \vdots \quad \vdots \quad \vdots \\ a_{d-1} \leq x_d \leq b_{d-1}, \end{cases} \tag{8}$$

where a_i (resp. b_i) is either $-\infty$ (resp. $+\infty$) or an expression of the form $\max(\ell_{i,1}, \ldots, \ell_{i,e_i})$ (resp. $\min(\ell_{i,1}, \ldots, \ell_{i,e_i})$), and each $\ell_{i,j}$ is a polynomial of $\mathbb{Q}[x_1, \ldots, x_i]$ with total degree at most 1, so that all the integer points of P are obtained by *back substitution*, that is, by specializing x_1 to every integer value v_1 in the interval (a_0, b_0), then by specializing x_2 to every integer value v_2 in the interval $(a_1(v_1), b_1(v_1))$, then by specializing x_3 to every integer value v_3 in the interval $(a_2(v_1, v_2), b_2(v_1, v_2))$, and so on.

A *parametric $\mathbb{Z}$-polyhedron* is defined as the set of the points $\mathbf{x} \in \mathbb{Z}^m$ which,

1. are the solutions of a parameterized linear inequality system

$$M\mathbf{x} \ \leq \ N\mathbf{y} + \mathbf{p}, \quad \text{and} \tag{9}$$

2. satisfy a parameteric system of linear congruences

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \in L, \tag{10}$$

where $M \in \mathbb{Q}^{s \times m}$ and $N \in \mathbb{Q}^{s \times n}$ are matrices, $\mathbf{p} \in \mathbb{Q}^s$ is a vector of constants, $\mathbf{y}$ is a vector of parameters taking values in $\mathbb{Z}^n$, L is a lattice of $\mathbb{Z}^{m+n}$, and s, m, n are non-negative integers with s and m being positive. We denote such a parametric $\mathbb{Z}$-polyhedron by $\mathsf{ParametricZPolyhedron}(M, \mathbf{x}, N, \mathbf{y}, p, L)$. Note also that some rows of the matrix M can be the zero vector, hence the system (9) may contain constraints involving the parameters $\mathbf{y}$ and not involving the variables $\mathbf{x}$. The above concepts are presented in great details in [9, 10].

3 PILP via Z-polyhedral set decomposition

We present our algorithm for parametric integer linear programming (PILP). Let s, m, n be non-negative integers, with s and m positive. Let $M \in \mathbb{Q}^{s\times m}$ and $N \in \mathbb{Q}^{s\times n}$, let $\mathbf{p} \in \mathbb{Q}^s$, let $\mathbf{y} \in \mathbb{Z}^n$ be a parameter vector, and let L be a lattice of $\mathbb{Z}^{m+n}$. The parametric $\mathbb{Z}$-polyhedron $\mathsf{ParametricZPolyhedron}(M, \mathbf{x}, N, \mathbf{y}, \mathbf{p}, L)$ is denoted by $\mathcal{P}(\mathbf{y})$. Let $g(\mathbf{x}, \mathbf{y}) \in \mathbb{Q}[\mathbf{x}, \mathbf{y}]$ be a *linear* polynomial, that is, of degree at most 1. Our goal is to compute the functions

$$\mathsf{Max} : \mathbf{y} \longmapsto \max\{g(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in \mathcal{P}(\mathbf{y})\} \tag{11}$$

and

$$\mathsf{Xmax} : \mathbf{y} \longmapsto \{\mathbf{x} \mid g(\mathbf{x}, \mathbf{y}) = \mathsf{Max}(\mathbf{y}) \text{ and } \mathbf{x} \in \mathcal{P}(\mathbf{y})\}. \tag{12}$$

Step 1: Encoding the values of the objective function as a Z-polyhedron

Introduce a new variable z and consider the $\mathbb{Z}$-polyhedron

$$\begin{cases} z = g(\mathbf{x}, \mathbf{y}) \\ \mathbf{x} \in \mathcal{P}(\mathbf{y}) \end{cases}, \tag{13}$$

so that, for each $\mathbf{y} \in \mathbb{Z}^n$,

$$\mathsf{Max}(\mathbf{y}) = \max\{z \in \mathbb{Z} \mid z = g(\mathbf{x}, \mathbf{y}),\ \mathbf{x} \in \mathcal{P}(\mathbf{y})\}. \tag{14}$$

We apply the algorithm $\mathsf{IntegerPointDecomposition}$ [7,11] to the $\mathbb{Z}$-polyhedron (13) under the variable ordering $x_1 > \cdots > x_m > z > y_1 > \cdots > y_n$. This produces polyhedral sets $P_1, \ldots, P_e \subseteq \mathbb{Q}^{m+1+n}$ and lattices $L_1, \ldots, L_e \subseteq \mathbb{Z}^{m+1+n}$ such that each $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$ is normalized, and

$$\begin{cases} z = g(\mathbf{x}, \mathbf{y}) \\ \mathbf{x} \in \mathcal{P}(\mathbf{y}) \end{cases} \quad \Longleftrightarrow \quad \bigvee_{i=1}^{e} \begin{pmatrix} \mathbf{x} \\ z \\ \mathbf{y} \end{pmatrix} \in \mathbb{Z}\mathsf{Polyhedron}(P_i, L_i). \tag{15}$$

Step 2: Reducing to at most one upper bound for z

Although System (13) involves z in a single equality, the normalized components $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$ may contain several inequalities in z. To compute Max, it is desirable to ensure that each component has *at most one upper bound* for z.

Proposition 3 ([10, Prop. 4], adapted). *For $1 \le i \le e$, one can compute normalized $\mathbb{Z}$-polyhedra $\mathbb{Z}\mathsf{Polyhedron}(P_{i,1}, L_{i,1}), \ldots, \mathbb{Z}\mathsf{Polyhedron}(P_{i,e_i}, L_{i,e_i})$ whose union equals $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$, and such that the H-representation of each $P_{i,j}$ contains at most one upper bound for z.*

The key idea is to compare two upper bounds $b_1 z \le A_1(\mathbf{y})$ and $b_2 z \le A_2(\mathbf{y})$ by splitting on whether $\frac{A_1(\mathbf{y})}{b_1} \le \frac{A_2(\mathbf{y})}{b_2}$ holds or not, reducing to the case where one bound dominates. Each branch is again a parametric $\mathbb{Z}$-polyhedron, and $\mathsf{IntegerPointDecomposition}$ is called on the parameter constraints alone; the remaining inequalities in $\mathbf{x}$ and z are appended without extra computation. Iterating this procedure reduces the number of upper bounds for z until at most one remains.

Step 3: Computing Max

After applying Proposition 3, we may assume each normalized component of the $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$ has at most one upper bound for z.

Proposition 4 ([10, Prop. 5], adapted). *Let $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$ be normalized with at most one upper bound for z.*

1. *If the row of the lattice matrix L_i defining z is the zero vector, then z is constant on $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$, say $z = z_0$, and $\mathsf{Max}(\mathbf{y}) = z_0$.*
2. *Otherwise, there exist $f_i \in \mathbb{Q}[y_1, \ldots, y_n]$ and $q_i \in \mathbb{Q} \setminus \{0\}$ such that every point of $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$ satisfies $z = q_i t + f_i(\mathbf{y})$ for some $t \in \mathbb{Z}$.*
   (2a) If P_i has no upper bound for z, then $\mathsf{Max}(\mathbf{y}) = +\infty$.
   (2b) If P_i has exactly one upper bound $b_i z \leq A_i(\mathbf{y})$, then

$$\mathsf{Max}(\mathbf{y}) = |q_i|\, t(\mathbf{y}) + f_i(\mathbf{y}), \text{where} \quad t(\mathbf{y}) := \left\lfloor \frac{A_i(\mathbf{y}) - b_i f_i(\mathbf{y})}{b_i |q_i|} \right\rfloor. \tag{16}$$

The proof follows by substituting $z = q_i t + f_i(\mathbf{y})$ into the upper bound inequality and computing the largest admissible integer t. Note that Equation (16) yields Max as a *piecewise quasi-polynomial* in $\mathbf{y}$, which is consistent with classical Ehrhart theory [3, 4, 6].

Step 4: Computing Xmax

Once $\mathsf{Max}(\mathbf{y})$ is known from Proposition 4, computing $\mathsf{Xmax}(\mathbf{y})$ the locus of all maximizers proceeds as follows on each normalized component $\mathbb{Z}\mathsf{Polyhedron}(P_i, L_i)$:

1. Replace all lower and upper bounds for z in P_i by the equality $z = \mathsf{Max}(\mathbf{y})$, and erase the row for z in L_i.
2. Apply Proposition 3 to ensure that, in the resulting $\mathbb{Z}$-polyhedra, each variable $x_m, x_{m-1}, \ldots, x_1$ has at most one lower and one upper bound.
3. Use back-substitution via the lattice structure (as in the proof of Proposition 4) to express each x_j as a closed-form function of $\mathbf{y}$.

The output is a finite union of parametric $\mathbb{Z}$-polyhedra, described in normalized form. This output, which gives the complete locus $\mathsf{Xmax}(\mathbf{y})$, distinguishes our algorithm from those of Feautrier [5] and Verdoolaege [16], since these latter two return only a sample point.

4 A motivating example: array bound checking via PILP

We illustrate our approach on a concrete instance of array bound checking. For space consideration, we only check that the array reference functions do not exceed the upper bounds of ther corresponding ranges. Consider the following triply-nested for-loop, where $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ and $\mathbf{M}_2, \mathbf{M}_3$ are positive integer parameters known only at execution time:

```
for (i1 = 0; i1 ≤ r1; i1++) do
  for (i2 = 0; i2 ≤ r2 − i1; i2++) do
    for (i3 = 0; i3 ≤ r3 − i1; i3++) do
      Ã[i1 + i2 + 1][i1 + 2 i2 + 1][2 i1 + i3] ← ···
    end for
  end for
end for
```

Here $\tilde{A}$ is an $\mathbf{M}_1 \times \mathbf{M}_2 \times \mathbf{M}_3$-array whose dimensions $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3$ are known only at execution time. The loop counters $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$ take non-negative integer values satisfying

$$0 \le \mathbf{i}_1 \le \mathbf{r}_1, \qquad 0 \le \mathbf{i}_2 \le \mathbf{r}_2 - \mathbf{i}_1, \qquad 0 \le \mathbf{i}_3 \le \mathbf{r}_3 - \mathbf{i}_1,$$

so the iteration domain is the *triangular prism*

$$\mathcal{D} = \big\{(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \in \mathbb{Z}^3 \;\big|\; 0 \le \mathbf{i}_1 \le \mathbf{r}_1,\ 0 \le \mathbf{i}_2 \le \mathbf{r}_2 - \mathbf{i}_1,\ 0 \le \mathbf{i}_3 \le \mathbf{r}_3 - \mathbf{i}_1\big\},$$

which is non-cuboid: the upper bounds of $\mathbf{i}_2$ and $\mathbf{i}_3$ both depend on $\mathbf{i}_1$, coupling the loop counters. The array reference functions

$$f_1 = \mathbf{i}_1 + \mathbf{i}_2 + 1, \qquad f_2 = \mathbf{i}_1 + 2\,\mathbf{i}_2 + 1, \qquad f_3 = 2\,\mathbf{i}_1 + \mathbf{i}_3 \tag{17}$$

are affine in $\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3$ and known at compile time. Following the framework of Equation (2), we must verify

$$\forall(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \in \mathcal{D} \quad \big(f_2(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \le \mathbf{M}_2 \ \wedge\ f_3(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \le \mathbf{M}_3\big). \tag{18}$$

Because the coefficients of $\mathcal{D}$ are literals and f_2, f_3 are affine, Formula (18) is a Presburger formula.

From QE to PILP. Eliminating universal quantifiers via De Morgan's laws produces a large number of cases to examine. Instead, following Equations (11)–(4), we replace (18) with

$$F_2 \le \mathbf{M}_2 \quad \wedge \quad F_3 \le \mathbf{M}_3, \tag{19}$$

where, for $i \in \{2, 3\}$,

$$F_i := \begin{array}{l}\max f_i(\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \\ (\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \in \mathcal{D}\end{array} \tag{20}$$

is a PILP instance in the sense of Equation (11). Because $\mathcal{D}$ is not a cuboid, neither maximum is obtained by simply setting each loop counter to its individual upper bound.

Solving the PILP instances. Our algorithm partitions the parameter space and returns, for each part, the maximum value and the full locus Xmax of maximizers. For $F_2 = \max\{f_2 \mid (\mathbf{i}_1, \mathbf{i}_2, \mathbf{i}_3) \in \mathcal{D}\}$ with $f_2 = \mathbf{i}_1 + 2\,\mathbf{i}_2 + 1$, the partition of the $(\mathbf{r}_1, \mathbf{r}_2)$ parameter space yields three regions:

$$F_2 = \begin{cases} 2\,\mathbf{r}_2 + 1 & \text{if } \mathbf{r}_1 \ge 1 \text{ and } \mathbf{r}_2 \ge 1, \\ 2\,\mathbf{r}_2 + 1 & \text{if } \mathbf{r}_1 = 0 \text{ and } \mathbf{r}_2 \ge 1, \\ 1 & \text{if } \mathbf{r}_1 = 0 \text{ and } \mathbf{r}_2 = 0, \end{cases} \tag{21}$$

with $\mathsf{Xmax}(F_2) = \{(0, \mathbf{r}_2, \mathbf{i}_3) \mid 0 \leq \mathbf{i}_3 \leq \mathbf{r}_3\}$ in the first two regions, and $\mathsf{Xmax}(F_2) = \{(0, 0, \mathbf{i}_3) \mid 0 \leq \mathbf{i}_3 \leq \mathbf{r}_3\}$ in the degenerate case. The first two regions share the same value $2\,\mathbf{r}_2 + 1$: the coupling $\mathbf{i}_2 \leq \mathbf{r}_2 - \mathbf{i}_1$ causes $f_2 = 2\,\mathbf{r}_2 - \mathbf{i}_1 + 1$ to decrease as $\mathbf{i}_1$ grows, so the optimum is always attained at $\mathbf{i}_1 = 0$. A naive cuboid estimate would give $\mathbf{r}_1 + 2\,\mathbf{r}_2 + 1$, strictly larger than the true optimum.

For $F_3 = \max\{f_3 \mid (\mathbf{i}_1, \mathbf{i}_3) \in \mathcal{D}\}$ with $f_3 = 2\,\mathbf{i}_1 + \mathbf{i}_3$, the partition of the $(\mathbf{r}_1, \mathbf{r}_3)$ parameter space yields six regions:

$$F_3 = \begin{cases} \mathbf{r}_1 + \mathbf{r}_3 & \text{if } \mathbf{r}_1 \geq 1 \text{ and } \mathbf{r}_3 \geq \mathbf{r}_1 + 1, \\ 2\,\mathbf{r}_3 & \text{if } \mathbf{r}_3 \geq 1 \text{ and } \mathbf{r}_1 \geq \mathbf{r}_3 + 1, \\ \mathbf{r}_3 & \text{if } \mathbf{r}_1 = 0 \text{ and } \mathbf{r}_3 \geq 1, \\ 2\,\mathbf{r}_3 & \text{if } \mathbf{r}_3 \geq 1 \text{ and } \mathbf{r}_1 \leq \mathbf{r}_3, \\ \mathbf{r}_1 + \mathbf{r}_3 & \text{if } \mathbf{r}_1 \geq 1,\ \mathbf{r}_3 \geq 2, \text{ and } \mathbf{r}_3 \geq \mathbf{r}_1 + 1, \\ 0 & \text{if } \mathbf{r}_1 = 0 \text{ and } \mathbf{r}_3 = 0, \end{cases} \tag{22}$$

with maximizers $\mathsf{Xmax}(F_3)$ at $(\mathbf{r}_1, \mathbf{i}_2, \mathbf{r}_3 - \mathbf{r}_1)$, $(\mathbf{r}_3, \mathbf{i}_2, 0)$, and $(0, \mathbf{i}_2, \mathbf{r}_3)$ in the respective regions. Consolidating overlapping regions, the essential case split is:

$$F_3 = \begin{cases} \mathbf{r}_1 + \mathbf{r}_3 & \text{if } \mathbf{r}_1 \leq \mathbf{r}_3, \\ 2\,\mathbf{r}_3 & \text{if } \mathbf{r}_1 > \mathbf{r}_3. \end{cases} \tag{23}$$

The interplay between $\mathbf{i}_1 \leq \mathbf{r}_1$ and $\mathbf{i}_3 \leq \mathbf{r}_3 - \mathbf{i}_1$ creates this non-trivial partition: increasing $\mathbf{i}_1$ raises $2\,\mathbf{i}_1$ but shrinks the feasible range of $\mathbf{i}_3$, and the optimum depends on which constraint is binding. A naive cuboid estimate would give $2\,\mathbf{r}_1 + \mathbf{r}_3$, strictly larger than the true optimum in all cases.

Validity conditions. Substituting into (19), the bound-checking condition (18) holds if and only if

$$2\,\mathbf{r}_2 + 1 \leq \mathbf{M}_2 \qquad \text{and} \qquad \begin{cases} \mathbf{r}_1 + \mathbf{r}_3 \leq \mathbf{M}_3 & \text{if } \mathbf{r}_1 \leq \mathbf{r}_3, \\ 2\,\mathbf{r}_3 \leq \mathbf{M}_3 & \text{if } \mathbf{r}_1 > \mathbf{r}_3. \end{cases} \tag{24}$$

These are necessary and sufficient conditions on the run-time parameters $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$, $\mathbf{M}_2, \mathbf{M}_3$, computed at compile-time by our PILP solver.

5 Experimentation

We have implemented in Maple a PILP Solver for solving parametric and non-parametric integer linear problems. For the experimentation reported here, these testing cases were executed on Maple 2024, installed on a machine running Ubuntu 24.04.1 LTS, with 16GB of RAM and on a 12th Gen Intel(R) Core(TM) i5-1235U processor. Those testing cases are taken from other authors in Paul Feautrier [5] and Sven Verdoolaege [16]. We compared the running time of our solver with those reported in their implementations. Although our method appears significantly slower, this difference is largely due to implementation choices: our PILP solver is developed in Maple, whereas their tools are implemented in C.

test	Feautrier	Barvinok	Sample Point	Locus
boulet	0.032	0.021	2.514	2.493
bouleti	0.039	error	13.069	12.795
cg1	0.024	0.027	1.205	1.167
cnt_sum2	1.114	Not supported	> 30 mins	> 30 mins
difficult	0.079	0.555	1136.988	1133.288
esced	0.051	0.026	394.609	390.996
ex	0.027	0.03	0.900	0.900
ex2	0.027	0.03	0.907	0.900
fimmel	0.038	0.038	0.644	0.640
jcomplex	583.28	Not supported	> 30 min	>30min
max	0.026	0.03	1.437	1.397
negative	0.021	0.028	0.521	0.521
phideo	0.764	>30 mins	>30 mins	>30 mins
seghir−e1	0.068	error	1278.216	1265.387
seghir−e3	0.031	0.067	73.585	73.353
seghir−e4	0.045	11.675	>30 mins	>30 mins
seghir−e5	0.147	2.533	>30 mins	>30 mins
seghir−e6	0.045	0.366	447.975	445.734
seghir−e7	0.029	0.111	55.668	55.795
seghir−e8	0.031	0.075	925.595	903.961
seghir−e9	0.332	>30 min	>30 min	> 30 mins
small	0.018	0.026	(1.209)ILP	(1.200)ILP
sor1d	0.024	0.022	0.455	0.453
square	0.027	0.029	0.916	0.958
sven	0.024	0.019	0.228	0.278
tobi	0.026	0.025	45.504	45.355

6 Concluding remarks

We compare our PIP solver, based on the IntegerPointDecomposition approach, against the solvers provided by ISL and Barvinok's library. In addition, our algorithm supports two types of optimizer outputs. The first is the *SamplePoint*, which returns a concrete integer witness $(x_1^*, \ldots, x_n^*)$. The existence of such a witness is guaranteed by the normalization of the IPD output, although computing it may require solving an auxiliary integer linear program (ILP) in more difficult instances. The second is the *Locus* method, which performs back-substitution over the normalized $\mathbb{Z}$-polyhedron to express each x_i^* as a *closed-form function of the parameters*. This approach avoids the need for an auxiliary ILP and is slightly faster in practice. In future work, we plan to extend our algorithm to handle non-linear PILP problems, which commonly arise in compiler scheduling.

References

1. Barvinok, A.I.: A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. Math. Oper. Res. **19**(4), 769–779 (1994)
2. Clauss, P., Fernández, F.J., Garbervetsky, D., Verdoolaege, S.: Symbolic polynomial maximization over convex sets and its application to memory requirement estimation. IEEE Trans. Very Large Scale Integr. Syst. **17**(8), 983–996 (2009)

3. Ehrhart, E.: Sur un problème de géométrie diophantienne linéaire. i. Journal für die reine und angewandte Mathematik **226**, 1–29 (1967)
4. Ehrhart, E.: Polynômes arithmétiques et méthode des polyèdres en combinatoire. International Series of Numerical Mathematics, Birkhäuser Verlag, Basel **35** (1977)
5. Feautrier, P.: Parametric integer programming. RAIRO. Recherche opérationnelle **22**(3), 243 – 268 (1988)
6. Jing, R., Lei, Y., Maligec, C.F.S., Moreno Maza, M.: Counting the integer points of parametric polytopes: A maple implementation. In: CASC 2024, Proceedings. Lecture Notes in Computer Science, vol. 14938, pp. 140–160. Springer (2024)
7. Jing, R.J., Lei, Y., Maligec, C.F.S., Moreno Maza, M., Mukherjee, C.: Integer hulls, Z-polyhedra and presburger arithmetic in action. ACM Commun. Comput. Algebra **59**(3), 41–46 (Jan 2026). https://doi.org/10.1145/3787957.3787958
8. Jing, R., Lei, Y., Maligec, C.F.S., Moreno Maza, M., Mukherjee, C.: Quantifier elimination over the integers. In: Proceedings of the ISSAC 2025. pp. 353–362. ACM (2025). https://doi.org/10.1145/3747199.3747580
9. Jing, R.J., Lei, Y., Maligec, C.F.S., Moreno Maza, M., Mukherjee, C.: Quantifier elimination over the integers. p. 353–362. ISSAC '25, Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3747199.3747580
10. Jing, R.J., Lei, Y., Moreno Maza, M., Mukherjee, C.: Quantifier elimination over the integers. SSRN Preprint (2025). https://doi.org/10.2139/ssrn.6178290, available at SSRN: https://ssrn.com/abstract=6178290
11. Jing, R., Moreno Maza, M.: Computing the integer points of a polyhedron, I: algorithm. In: CASC 2017, Proceedings. LNCS, vol. 10490, pp. 225–241. Springer (2017)
12. Matiyasevič, Y.: Enumerable sets are diophantine. Mathematical logic in the 20th century pp. 269–273 (2003)
13. Nguyen, T.V.N., Irigoin, F.: Efficient and effective array bound checking. ACM Trans. Program. Lang. Syst. **27**(3), 527–570 (2005). https://doi.org/10.1145/1065887.1065893
14. Robinson, J.: Unsolvable diophantine problems. Proceedings of the AMS **22**(2), 534–538 (1969)
15. Venet, A., Brat, G.: Precise and efficient static array bound checking for large embedded c programs. ACM Sigplan Notices **39**(6), 231–242 (2004)
16. Verdoolaege, S.: isl: An integer set library for the polyhedral model. In: Mathematical Software - ICMS 2010, Lecture Notes in Computer Science, vol. 6327, pp. 299–302. Springer (2010)
17. Verdoolaege, S., Seghir, R., Beyls, K., Loechner, V., Bruynooghe, M.: Counting integer points in parametric polytopes using Barvinok's rational functions. Algorithmica **48**(1), 37–66 (2007)