A Faster Algorithm for Two-Variable Integer Programming

Friedrich Eisenbrand and Sören Laue

Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany {eisen,soeren}@mpi-sb.mpg.de

Abstract. We show that a 2-variable integer program, defined by m constraints involving coefficients with at most φ bits can be solved with $O(m + \varphi)$ arithmetic operations on rational numbers of size $O(\varphi)$. This result closes the gap between the running time of two-variable integer programming with the sum of the running times of the Euclidean algorithm on φ-bit integers and the problem of checking feasibility of an integer point for m constraints.

1 Introduction

Integer programming is the problem of maximizing a linear function over the integer vectors which satisfy a given set of inequalities. A wide range of combinatorial optimization problems can be modeled as integer programming problems. But integer programming is not only related to combinatorics. The *greatest common divisor* of two numbers a and $b \in \mathbb{Z}$ is the smallest integer combination $x\,a + y\,b$ such that $x\,a + y\,b \geqslant 1$. This is an integer program in two variables. This fact links integer programming also to the algorithmic theory of numbers.

The Euclidean algorithm requires $O(\varphi)$ arithmetic operations, if φ is the binary encoding length of the input. Checking an integer point for feasibility, requires to test it for all the constraints. In this paper we prove that an integer program $\max\{c^t x \mid Ax \leqslant b,\, x \in \mathbb{Z}^2\}$, where $c \in \mathbb{Z}^2$, $A \in \mathbb{Z}^{m\times 2}$ and $b \in \mathbb{Z}^m$ involve coefficients with at most φ bits, can be solved with $O(m + \varphi)$ arithmetic operations on rationals of binary encoding length $O(\varphi)$. In the arithmetic complexity model, this is the best one can hope for if one believes that greatest-common-divisor computation requires $\Omega(\varphi)$ arithmetic operations.

Related Work

The two-variable integer programming problem has a long history. Polynomiality was established by Hirschberg and Wong [8] and Kannan [10] for special cases and by Scarf [17,18] for the general case. Then, Lenstra [15] proved that integer programming in arbitrary fixed dimension can be solved in polynomial time.

Afterwards, various authors were looking for faster algorithms for the two-dimensional case. Here is a table which summarizes the development of the last 20 years. In this table, m denotes the number of constraints and φ denotes the maximal binary encoding length of an involved coefficient.

T. Ibaraki, N. Katoh, and H. Ono (Eds.): ISAAC 2003, LNCS 2906, pp. 290–299, 2003.

Method for integer programming	complexity
Feit [6]	$O(m \log m + m\varphi)$
Zamanskij and Cherkasskij [21]	$O(m \log m + m\,\varphi)$
Kanamaru, Nishizeki and Asano [9]	$O(m \log m + \varphi)$
Eisenbrand and Rote [5]	$O(m + (\log m)\,\varphi)$
Clarkson [2] combined with Eisenbrand [4] [1]	$O(m + (\log m)\,\varphi)$
This paper	$O(m + \varphi)$
Checking a point for feasibility	$\Theta(m)$
Greatest common divisor computation	$O(\varphi)$

For comparison, we have also given the complexity of greatest-common-divisor computation and of checking whether a given integer point is feasible. Thus the last two lines of the table is the goal that one should aim for. This paper achieves this goal.

Our algorithm is the fastest algorithm in the *arithmetic complexity model*. Here, the basic arithmetic operations `+,-,*,/` are unit-cost operations. This is in contrast to the *bit-complexity model*, where bit-operations are counted. In this model, the algorithm in [5] is the fastest known so far. Its complexity is $O(m + \log m \log \varphi)M(\varphi)$, where $M(\varphi)$ is the bit-complexity of φ-bit integer multiplication. In the bit-model, our algorithm can also be analyzed to require $O(m+\log m \log \varphi)M(\varphi)$ if the occurring shortest vector queries are individually carried out with Schönhage's algorithm [19].

It is well known, see, e.g. [4,5,9] that, by means of an appropriate unimodular transformation, we can assume that the objective is to maximize the value of the first component. In fact, a reduction of a general integer programming problem to this special objective function requires one extended gcd-computation and a constant number of arithmetic operations. Thus we define the integer programming problem as follows.

Problem 1 (2IP). Given a system of inequalities $Ax \leqslant b$, where $A \in \mathbb{Z}^{m\times 2}$ and $b \in \mathbb{Z}^m$, determine an integer point $x^* \in \mathbb{Z}^2$ which satisfies $Ax \leqslant b$ and has maximal first component $x^*(1)$, or assert that $Ax \leqslant b$ is integer infeasible.

In the following, the letter m denotes the number of constraints of $Ax \leqslant b$ and φ is an upper bound on the binary encoding length of each constraint $a^t x \leqslant \beta$ of $Ax \leqslant b$. We can also assume that the polyhedron $\{x \in \mathbb{R}^2 \mid Ax \leqslant b\}$ is bounded, thus that the constraints define a *convex polygon* $P = \{x \in \mathbb{R}^2 \mid Ax \leqslant b\}$.

The main idea of our approach is to dissect the polygon into *four* specially structured polygons and to solve the integer programming problem for each of them separately. To do so, we need to check their lattice width. This can be approximated by shortest vector queries for inscribed triangles. Since the polygons have a special structure, the corresponding triangles have similar shapes. This allows us to batch the shortest vector queries for them so that they can be carried out in time $O(\log \varphi)$ for each query after $O(\varphi)$ preprocessing time. Then, our algorithm follows a prune-and-search technique.

[1] This is a randomized method for arbitrary fixed dimension

2 Preliminaries from Algorithmic Number Theory

In this section, we review some basics from algorithmic number theory, which are necessary to develop our algorithm.

2.1 The Euclidean Algorithm and Best Approximations

The Euclidean algorithm for computing the *greatest common divisor* $\gcd(a_0, a_1)$ of two integers $a_0, a_1 > 0$ computes the remainder sequence $a_0, a_1, \ldots, a_{k-1}, a_k \in \mathbb{N}_+$, where $a_i,\ i \geqslant 2$ is given by $a_{i-2} = a_{i-1}q_{i-1} + a_i$, $q_i \in \mathbb{N}$, $0 < a_i < a_{i-1}$, and a_k divides a_{k-1} exactly. Then $a_k = \gcd(a_0, a_1)$. The *extended Euclidean algorithm* keeps track of the unimodular matrices $M^{(j)} = \prod_{i=1}^{j} \left(\begin{smallmatrix} q_i & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $0 \leqslant j \leqslant k-1$. One has $\left(\begin{smallmatrix} a_0 \\ a_1 \end{smallmatrix}\right) = M^{(j)} \left(\begin{smallmatrix} a_j \\ a_{j+1} \end{smallmatrix}\right)$. The extended Euclidean algorithm requires $O(\varphi)$ arithmetic operations on $O(\varphi)$-bit integers, if the binary encoding length of a_0 and a_1 is $O(\varphi)$, see also [13,1].

The fractions $M^{(i)}_{1,1}/M^{(i)}_{2,1}$ are called the *convergents* of $\alpha = a_0/a_1$. A fraction $x/y,\ y \geqslant 1$ is called a *best approximation*,[2] if one has $|y\alpha - x| < |y'\alpha - x|$ for all other fractions $x'/y',\ 0 < y' \leqslant y$. A best approximation to α is a convergent of α, see, e.g. [12].

2.2 Lattices

A 2-*dimensional (rational) lattice* Λ is a set of the form $\Lambda(A) = \{\, Ax \mid x \in \mathbb{Z}^2 \,\}$, where $A \in \mathbb{Q}^{2\times 2}$ is a nonsingular rational matrix. The matrix A is called a *basis* of Λ. One has $\Lambda(A) = \Lambda(B)$ for $B \in \mathbb{Q}^{2\times 2}$ if and only if $B = AU$ with some *unimodular matrix* U, i.e., $U \in \mathbb{Z}^{2\times 2}$ and $\det(U) = \pm 1$. Every lattice $\Lambda(A)$ has a unique basis of the form $\left(\begin{smallmatrix} a & b \\ 0 & c \end{smallmatrix}\right) \in \mathbb{Q}^{2\times 2}$, where $c > 0$ and $a > b \geqslant 0$, called the *Hermite normal form, HNF* of Λ, see, e.g. [20]. The Hermite normal form can be computed with an extended-gcd computation and a constant number of arithmetic operations.

A *shortest vector* of a lattice Λ is a nonzero vector $v \in \Lambda - \{0\}$ with minimal ℓ_∞-norm $\|v\|_\infty = \max\{|v(i)| \mid i = 1, 2\}$. There are many algorithms known to compute a shortest vector of a 2-dimensional lattice [7,14,19]. The following approach is very useful for our purposes.

Proposition 1 ([3] [3]). *Let $\Lambda \subseteq \mathbb{Q}^2$ be a rational lattice which is given by its Hermite normal form $\left(\begin{smallmatrix} a & b \\ 0 & c \end{smallmatrix}\right)$. A shortest vector of Λ with respect to the ℓ_∞-norm is either $\left(\begin{smallmatrix} a \\ 0 \end{smallmatrix}\right)$ or $\left(\begin{smallmatrix} b \\ c \end{smallmatrix}\right)$, or a vector of the form $\left(\begin{smallmatrix} -x & a+y\,b \\ y & c \end{smallmatrix}\right)$, where the fraction x/y is a best approximation of the number b/a.*

Later, we will have to deal with the following problem for which we provide an algorithm below.

[2] In [12] this is referred to as *best approximation of the second kind*

[3] In [3] this assertion is stated for integral lattices. It is easy to see that it also holds for rational lattices

Problem 2. Given a lattice basis $A \in \mathbb{Q}^{2\times 2}$ and a sequence of K positive rational numbers $\alpha_1, \ldots, \alpha_K$, find a shortest vector w.r.t. the ℓ_∞-norm for each of the lattices Λ_i generated by the matrices $\left(\begin{smallmatrix} 1 & 0 \\ 0 & \alpha_i \end{smallmatrix}\right) \cdot A$, for $i = 1, \ldots, K$.

Lemma 1. *Let $A \in \mathbb{Q}^{2\times 2}$ and $\alpha_1, \ldots, \alpha_K$ be parameters of Problem 2, where A and each of the α_i have binary encoding length $O(\varphi)$. Then Problem 2 can be solved with $O(\varphi + K \log \varphi)$ arithmetic operations on rational numbers of size $O(\varphi)$.*

Proof. First we compute the Hermite normal form $\left(\begin{smallmatrix} a & b \\ 0 & c \end{smallmatrix}\right)$ of A with the extended Euclidean algorithm. Then we compute all convergents x_j/y_j, $j = 1, \ldots, k$ of b/a with the extended Euclidean algorithm. From this, the convergents come out with the following property. The sequence $|-x_j\, a + y_j\, b|$ is monotonously decreasing and the sequence $y_j\, c$ is monotonously increasing and nonnegative.

By Proposition 1 and since a best approximation of b/a is a convergent of b/a, for each of the α_i, we have to determine the convergent x_j/y_j of b/a such that $\|\left(\begin{smallmatrix} -x_j\, a + y_j\, b \\ y_j\, \alpha_i\, c \end{smallmatrix}\right)\|_\infty$ is minimal. For this, we search the position j_i in the list of convergents, where $|-x_{j_i}\, a + y_{j_i}\, b| \geqslant y_{j_i}\, \alpha_i\, c$ and $|-x_{j_i+1}\, a + y_{j_i+1}\, b| < y_{j_i+1}\, \alpha_i\, c$. If $|-x_j\, a + y_j\, b| \geqslant y_j\, \alpha_i\, c$ holds for all convergents x_j/y_j, then j_i shall be the second-last position. Similarly, if $|-x_j\, a + y_j\, b| \leqslant y_j\, \alpha_i\, c$ for all convergents x_j/y_j, then j_i shall be the first position. The shortest vector of Λ_i is then the shortest vector among the vectors

$$\begin{pmatrix} -x_{j_i}\, a + y_{j_i}\, b \\ y_{j_i}\, \alpha_i\, c \end{pmatrix}, \begin{pmatrix} -x_{j_i+1}\, a + y_{j_i+1}\, b \\ y_{j_i+1}\, \alpha_i\, c \end{pmatrix}, \begin{pmatrix} a \\ 0 \end{pmatrix}, \begin{pmatrix} b \\ \alpha_i\, c \end{pmatrix}. \tag{1}$$

Since there are at most $O(\varphi)$ convergents of b/a, this position j_i, can be computed with binary search in $O(\log \varphi)$ many steps. Thus we have the desired running time of $O(\varphi + K \log \varphi)$.

2.3 The Flatness Theorem

A central concept of our algorithm, as in Lenstra's algorithm [15], is the lattice width of a convex body. Let $K \subseteq \mathbb{R}^d$ be a convex body. The width of K along a direction $c \in \mathbb{R}^d$ is defined as $w_c(K) = \max\{c^t x \mid x \in K\} - \min\{c^t x \mid x \in K\}$. The lattice width $w(K)$ of a K is defined as the minimum $w_c(K)$ over all nonzero vectors $c \in \mathbb{Z}^d - \{0\}$. Thus if a convex body has lattice width ℓ with a corresponding direction $c \in \mathbb{Z}^d$, then all its lattice points can be covered by at most $\lfloor \ell \rfloor + 1$ parallel hyperplanes of the form $c^t x = \delta$, where $\delta \in \mathbb{Z} \cap [\min\{c^t x \mid x \in K\}, \max\{c^t x \mid x \in K\}]$. If a convex body does not contain any lattice points, then it must be *thin* in some direction, or equivalently its lattice width must be small. This is known as Khinchin's Flatness Theorem [11].

Theorem 1 (Flatness theorem). *There exists a constant $f(d)$ depending only on the dimension d, such that each full-dimensional convex body $K \subseteq \mathbb{R}^d$ containing no integer point has width at most $f(d)$.*

How can the width of a convex body be computed? In this paper, we only need to do this for triangles. Let $T = \mathrm{conv}(u, v, w) \subseteq \mathbb{R}^2$ be a triangle. The width is invariant under translation. Thus the width of T is the width of the triangle $T' = \mathrm{conv}(0, v-u, w-u)$. The width of T' along a vector $c \in \mathbb{R}^2$ is then bounded from below by $\max\{|c^t(v-u)|, |c^t(w-u)|\}$ and bounded from above by $2\max\{|c^t(v-u)|, |c^t(w-u)|\}$. Let $A_T \in \mathbb{R}^2$ be the matrix $A_T = \binom{(v-u)^t}{(w-u)^t}$. The width along c thus satisfies the following relation

$$\|A_T\, c\|_\infty \leqslant w_c(T) \leqslant 2\,\|A_T\, c\|_\infty. \tag{2}$$

This means that the width of T is bounded from below by the length of the shortest (infinity norm) vector of $\Lambda(A_T)$ and bounded from above by twice the length of the shortest vector of $\Lambda(A_T)$. Furthermore, if $v = A_T\, c$ is a shortest vector, then the following relation holds

$$w_c(T) \leqslant w(T) \leqslant 2\, w_c(T). \tag{3}$$

In the sequel, we call a vector $c \in \mathbb{Z}^2$, such that $v = A_T\, c$ is a shortest vector of $\Lambda(A_T)$, a *thin direction* of T. A shortest vector of $\Lambda(A_T)$ w.r.t. the ℓ_∞-norm will be denoted as a *shortest vector of the triangle* T. Its length is denoted by $\mathrm{SV}(T)$.

3 Partitioning the Polygon

In a first step, we partition the polygon into four parts. Two of the parts belong to a class of polygons for which one already knows an $O(m + \varphi)$ algorithm for their corresponding integer programs [5]. In the following sections, we will deal with the other two polygons.

First we compute the rightmost point and the leftmost point of P and we consider the line g through these two points, see Figure 1. This line dissects P into an upper part P_U and a lower part P_L. Next we compute vertices of P_U and P_L which have largest distance from the line g and draw a vertical line h_U and h_L through these points. The line h_U dissects P_U again in two parts, an upper-left polygon P_{Ul} and an upper-right polygon P_{Ur}. The line h_L partitions P_L into two parts, a lower-left polygon P_{Ll} and a lower-right polygon P_{Lr}. The optimum integer point in P is the maximum of the optima of these four polygons. This partition can be found with linear programming. Using the algorithm of Megiddo [16], this requires $O(m)$ operations. Notice that the binary encoding length of each constraint describing the four polygons remains $O(\varphi)$.

The polygons P_{Ul} and P_{Ll} are *lower polygons* in the terminology of Eisenbrand and Rote [5]. This is because they have a line-segment parallel to the objective line as an edge and there are two parallel lines trough the endpoints of this edge which enclose the polygon.[4] Thus Proposition 1 and Theorem 2

[4] In [5] the objective is to find a highest integer point, while we find a rightmost integer point

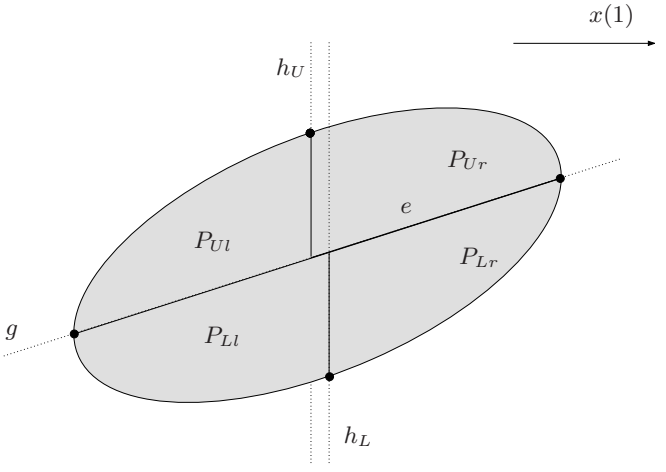

Fig. 1. The dissection of the polygon P. The arrow is the $x(1)$-direction in which we optimize.

of [5] implies that the integer program over P_{Ul} and P_{Ll} can be solved in with $O(m + \varphi)$ arithmetic operations on rationals with $O(\varphi)$ bits.

The polygons P_{Ur} and P_{Lr} are the ones we need to take care of. The polygon P_{Ur} has a special structure. It has an edge e, such that each point of P_{Ur} lies vertically above e and the two vertical lines through the endpoints of this edge enclose the polygon. Furthermore, the vertical line through the vertex on the left of e, defines a facet of P_{Ur}. All the other facets, from left to right, have decreasing slope and each slope is at most the slope of e. A polygon of this kind will be called a polygon of *upper-right* kind in the sequel. Notice that P_{Lr} becomes an upper-right polygon, when it is reflected around the $x(1)$ axis. Therefore we concentrate now on the solution of integer programming problems over polygons of upper-right kind.

4 A Prune-and-Search Algorithm

In the following, let P be a polygon of upper-right kind. We now present an $O(m + \varphi)$ algorithm for this case. Similar to the algorithm in [5] we use the prune-and-search technique of Megiddo [16] to solve the optimization problem over P.

The idea is to search for a parameter ℓ, such that the truncated polygon $P_\ell = P \cap (x(1) \geqslant \ell)$ has width $w(P_\ell)$ between $f(2) + 1$ and $4(f(2) + 1)$. If we have found such an ℓ, we know two important things. First, the flatness theorem guarantees that P_ℓ is feasible and thus that the optimum of the integer

programming problem over P lies in P_ℓ. Furthermore, all lattice points of P_ℓ, and therefore also the optimum, must lie on at most $4(f(2)+1)+1$ parallel line segments in the corresponding flat direction. Thus, we have reduced the integer programming problem over P to the problem of finding an optimum of a constant number of one-dimensional integer programming problems, which then can be solved in linear time.

We will approximate the width of P_ℓ as follows. Consider the edge f of P_ℓ induced by the constraint $x(1) \geqslant \ell$ and the edge e', which emerges from the lower edge e of P intersected with $(x(1) \geqslant \ell)$. The convex hull of both edges is a triangle T_ℓ, see, Figure 2.

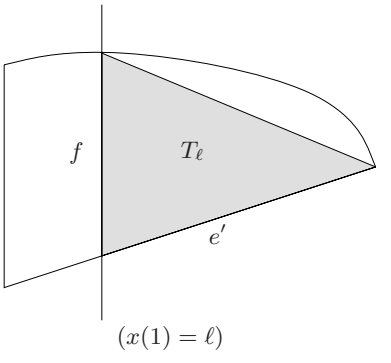

Fig. 2. The polygon P_ℓ and the triangle T_ℓ.

Obviously, we have $T_\ell \subseteq P_\ell$. It is easy to see that if we scale T_ℓ by a factor of 2 and translating T_ℓ appropriately, then it includes P_ℓ. Hence, the width $w(P_\ell)$ satisfies $w(T_\ell) \leqslant w(P_\ell) \leqslant 2\, w(T_\ell)$. From Section 2.3 we can conclude $\mathrm{SV}(T_\ell) \leqslant w(P_\ell) \leqslant 4\,\mathrm{SV}(T_l)$. Thus, we are interested in a parameter ℓ, such that the shortest vector of T_ℓ has length $f(2)+1$.

We start with m constraints and maintain two numbers ℓ_{thick} and ℓ_{thin}. In the beginning, ℓ_{thick} is the $x(1)$-component of the left endpoint of the edge e and ℓ_{thin} is the $x(1)$-component of the right endpoint of the edge e. If $\mathrm{SV}(T_{\ell_{thick}}) \leqslant f(2)+1$, then P itself is flat and we are done. Otherwise we keep the following invariant.

The shortest vector of $T_{\ell_{thick}}$ has length at least $f(2)+1$ and the shortest vector of $T_{\ell_{thin}}$ has length at most $f(2)+1$.

The idea is to prune constraints, while we search for the correct position ℓ, which cannot be facet defining for the intermediate part of the polygon $P \cap (x(1) \geqslant \ell_{thick}) \cap (x(1) \leqslant \ell_{thin})$, see Figure 3.

One iteration is as follows. We pair up all m constraints yielding $m/2$ intersection points. Then we compute the x-median ℓ_{med} of the intersection points.

Now we distinguish three cases. One is that ℓ_{med} lies to the right of ℓ_{thin}. In this case, we can delete from each pair of intersection points to the right of the median, the constraint with the smaller slope. We can do this, since this constraint cannot be facet-defining for the intermediate polygon. Similarly, if ℓ_{med} lies to the left of ℓ_{thick}, we can delete from each pair on the left of the median the constraint with the larger slope.

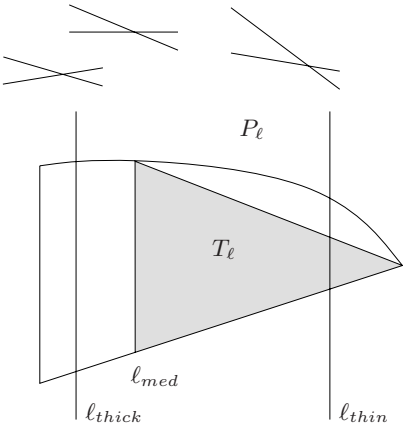

Fig. 3. The prune-and-search algorithm for a polygon of the upper-right kind

The more interesting case is the one, where ℓ_{med} lies in-between ℓ_{thick} and ℓ_{thin}. Then we compute the length of the edge f which is induced by $x(1) \geqslant \ell_{med}$. This edge is simply the line-segment spanned by the intersection of $x(1) = \ell_{med}$ with e and by the lowest intersection point of the line $x(1) = \ell_{med}$ with all m constraints.

Now we compute the shortest vector of $T_{\ell_{med}}$. If its length is smaller than $f(2) + 1$, then we set ℓ_{thin} to ℓ_{med} and delete from each intersection point that lies to the right of ℓ_{med} the constraint with the smaller slope. Otherwise we set ℓ_{thick} to ℓ_{med} and delete from each intersection point that lies to the left of ℓ_{med} the constraint with the larger slope.

We repeat this prune and search procedure until we have found a position ℓ, where the shortest vector of T_ℓ is $f(2)+1$ or we identified a constant number of constraints, which can be facet defining for $P \cap (x(1) \geqslant \ell_{thick}) \cap (x(1) \leqslant \ell_{thin})$.

In the first case, we know that the optimum lies in P_ℓ and we have a flat direction of P_ℓ, namely the vector $c \in \mathbb{Z}^2 - \{0\}$ such that $v = A_{T_\ell}\, c$ is a shortest vector of T_ℓ. Thus the optimum is the largest of the optima of the integer programs over the constant number line segments $P_\ell \cap (c^t x = \delta)$, where $\delta \in \mathbb{Z} \cap [\min\{c^t x \mid x \in P_\ell\}, \max\{c^t x \mid x \in P_\ell\}]$. In the second case, we know that the optimum lies in $P_{\ell_{thick}}$. Furthermore, $P_{\ell_{thick}}$ can be partitioned into $P_{\ell_{thin}}$, and the polygon $P \cap (x(1) \geqslant \ell_{thick}) \cap (x(1) \leqslant \ell_{thin})$. The first polygon is flat.

The second polygon is defined by a constant number of constraints, for which integer programming can be solved with $O(\varphi)$ arithmetic operations.

Analysis

We will prove that the presented algorithm runs in $O(m + \varphi)$ using rational numbers of size $O(\varphi)$.

Suppose we are in the i-th round of the prune-and-search algorithm and suppose we are left with m_i constraints. In this round we compute $m_i/2$ intersection points, the median of them, the corresponding triangle T_ℓ and query for the shortest vector of T_ℓ. Hence, the running time for round i, without considering the shortest vector queries, is $O(m_i)$. We discard $1/4$ of the constraints. Therefore, the overall running time of the prune-and-search algorithm without considering the shortest vector queries is $O(m)$.

Let us consider the shortest vector queries. Let T be the first triangle, for which we compute the shortest vector. The angle, which is enclosed by the edges f and e' is the same for all triangles for which we query a shortest vector. Let A_T be the matrix of T as it is defined at the end of Section 2.3. The matrices A_{T_ℓ} of the following triangles thus satisfy

$$A_{T_\ell} = \beta_\ell \cdot \left(\begin{smallmatrix} 1 & 0 \\ 0 & \alpha_\ell \end{smallmatrix}\right) \cdot A_T, \tag{4}$$

with rational numbers α_ℓ and β_ℓ which can be computed from T and T_ℓ in constant time. The length of the shortest vector of T_ℓ is equal to β_ℓ times the length of the shortest vector of the lattice $\Lambda\left(\left(\begin{smallmatrix} 1 & 0 \\ 0 & \alpha_\ell \end{smallmatrix}\right) \cdot A_T\right)$. Hence, we can apply Lemma 1. As we perform $O(\log m)$ queries, the total number shortest vector queries can be computed with $O(\varphi + \log m \cdot \log \varphi)$ arithmetic operations on rational numbers of size $O(\varphi)$. Thus the total running time amounts to $O(m + \varphi + \log m \cdot \log \varphi) = O(m + \varphi)$ arithmetic operations on rational numbers of binary encoding length $O(\varphi)$, which proves our main result.

Theorem 2. *A two-variable integer programming problem* $\max\{c^t x \mid Ax \leqslant b, x \in \mathbb{Z}^2\}$*, where* $A \in \mathbb{Z}^{m\times 2}$ *and* $b \in \mathbb{Z}^m$ *and* $c \in \mathbb{Z}^2$ *involve only coefficients of binary encoding length* $O(\varphi)$*, can be solved with* $O(\varphi + m)$ *arithmetic operations on rational numbers of size* $O(\varphi)$*.*

Acknowledgement. We thank the anonymous referees for their valuable comments and suggestions.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, 1974.
2. K. L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the Association for Computing Machinery*, 42:488–499, 1995.

3. F. Eisenbrand. Short vectors of planar lattices via continued fractions. *Information Processing Letters*, 79(3):121–126, 2001.
4. F. Eisenbrand. Fast integer programming in fixed dimension. Technical Report MPI-I-2003-NWG2-002, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2003. to appear in the Proceedings of ESA 2003.
5. F. Eisenbrand and G. Rote. Fast 2-variable integer programming. In K. Aardal and B. Gerards, editors, *Integer Programming and Combinatorial Optimization, IPCO 2001*, volume 2081 of *LNCS*, pages 78–89. Springer, 2001.
6. S. D. Feit. A fast algorithm for the two-variable integer programming problem. *Journal of the Association for Computing Machinery*, 31(1):99–113, 1984.
7. C. F. Gauß. *Disquisitiones arithmeticae*. Gerh. Fleischer Iun., 1801.
8. D. S. Hirschberg and C. K. Wong. A polynomial algorithm for the knapsack problem in two variables. *Journal of the Association for Computing Machinery*, 23(1):147–154, 1976.
9. N. Kanamaru, T. Nishizeki, and T. Asano. Efficient enumeration of grid points in a convex polygon and its application to integer programming. *International Journal of Computational Geometry & Applications*, 4(1):69–85, 1994.
10. R. Kannan. A polynomial algorithm for the two-variable integer programming problem. *Journal of the Association for Computing Machinery*, 27(1):118–122, 1980.
11. R. Kannan and L. Lovász. Covering minima and lattice-point-free convex bodies. *Annals of Mathematics*, 128:577–602, 1988.
12. A. Y. Khintchine. *Continued Fractions*. Noordhoff, Groningen, 1963.
13. D. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, 1969.
14. J. C. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1:142–186, 1980.
15. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.
16. N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the Association for Computing Machinery*, 31:114–127, 1984.
17. H. E. Scarf. Production sets with indivisibilities. Part I: generalities. *Econometrica*, 49:1–32, 1981.
18. H. E. Scarf. Production sets with indivisibilities. Part II: The case of two activities. *Econometrica*, 49:395–423, 1981.
19. A. Schönhage. Fast reduction and composition of binary quadratic forms. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 91*, pages 128–133. ACM Press, 1991.
20. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986.
21. L. Y. Zamanskij and V. D. Cherkasskij. A formula for determining the number of integral points on a straight line and its application. *Ehkon. Mat. Metody*, 20:1132–1138, 1984.