Fast 2-Variable Integer Programming

Friedrich Eisenbrand[1] and Günter Rote[2]

[1] Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, eisen@mpi-sb.mpg.de
[2] Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany, rote@inf.fu-berlin.de

Abstract. We show that a 2-variable integer program defined by m constraints involving coefficients with at most s bits can be solved with $O(m + s\log m)$ arithmetic operations or with $O(m+\log m \log s)M(s)$ bit operations, where $M(s)$ is the time needed for s-bit integer multiplication.

1 Introduction

Integer linear programming is related to convex geometry as well as to algorithmic number theory, in particular to the algorithmic geometry of numbers. It is well known that some basic number theoretic problems, such as the greatest common divisor or best approximations of rational numbers can be formulated as integer linear programs in two variables. Thus it is not surprising that current polynomial methods for *integer programming in fixed dimension* [7, 12] use *lattice reduction* methods, related to the reduction which is part of the classical Euclidean algorithm for integers, or the computation of the *continued fraction expansion* of a rational number. Therefore, integer programming in fixed dimension has a strong flavor of algorithmic number theory, and the running times of the algorithms also depend on the binary encoding length of the input.

In this paper, we want to study this relation more carefully for the case of 2-*dimensional integer programming*. The classical Euclidean algorithm for computing the greatest common divisor (GCD) of two s-bit integers requires $\Theta(s)$ arithmetic operations and $\Theta(s^2)$ bit operations in the worst case. For example, when it is applied to two consecutive Fibonacci numbers, it generates all the predecessors in the Fibonacci sequence (see e.g. [10]). Schönhage's algorithm [17] improves this complexity to $O(M(s)\log s)$ bit operations, where $M(s)$ is the bit complexity of s-bit integer multiplication. Thus the greatest common divisor of two integers can be computed with a close to linear number of *bit operations*, if one uses the fastest methods for integer multiplication [19]. The speedup technique by Schönhage has not yet been incorporated into current methods for two variable integer programming. The best known algorithms for the integer programming problem in two dimensions [4, 22, 6] use $\Theta(s)$ arithmetic operations and $\Omega(s^2)$ bit operations when the number of constraints is fixed. This number of steps is required because these algorithms construct the complete sequence of convergents of certain rational numbers that are computed from the input.

Our goal is to show that integer programming in two variables is not harder than greatest common divisor computation. We achieve this goal for the case that the number of constraints is fixed. As one allows an arbitrary number of constraints, the nature of the problem also becomes combinatorial. For this general case we present an algorithm which requires $O(\log m)$ gcd-like computations, where m is the number of constraints. This improves on the best previously known algorithms.

Previous work. The 2-variable integer programming problem was extensively studied by various authors. The polynomiality of the problem was settled by Hirschberg and Wong [5] and Kannan [9] for special cases and by Scarf [15, 16] for the general case before Lenstra [12] established the polynomiality of integer programming in any fixed dimension. Since then, several algorithms for the 2-variable problem have been suggested. We summarize them in the following table, for problems with m constraints involving (integer) numbers with at most s bits.

Method for integer programming	arithmetic complexity	bit complexity
Feit [4]	$O(m\ \log m + ms)$	$O(m\ \log m + ms)M(s)$
Zamanskij and Cherkasskij [22]	$O(m\ \log m + ms)$	$O(m\ \log m + ms)M(s)$
Kanamaru et al. [6]	$O(m\ \log m + s)$	$O(m\ \log m + s)M(s)$
Clarkson [2] (randomized)[1]	$O(m + s^2\ \log m)$	$O(m + \log m\ s^2)M(s)$
This paper (Theorem 4)	$O(m + \log m\ s)$	$O(m + \log m\ \log s)M(s)$
Checking a point for feasibility	$\Theta(m)$	$\Theta(m)M(s)$
Shortest vector [18, 21], GCD [17]	$O(s)$	$O(\log s)M(s)$

Thus our algorithm is better in the arithmetic model if the number of constraints is large, whereas the bit complexity of our algorithm is superior to the previous methods in all cases.

For comparison, we have also given the complexity of a few basic operations. The greatest common divisor of two integers a and b can be calculated by the special integer programming problem $\min\{\, ax_1+bx_2 \mid ax_1+bx_2 \geq 1,\ x_1, x_2 \in \mathbb{Z}\,\}$ in two variables with one constraint. Also, checking whether a given point is feasible should be no more difficult than finding the optimum. So the sum of the last two lines of the table is the goal that one might aim for (short of trying to improve the complexity of integer multiplication or GCD itself). The complexity of our algorithm has still an extra $\log m$ factor in connection with the terms involving s, compared to the "target" of $O(m + s)$ and $O(m + \log s)M(s)$, respectively. However, we identify a nontrivial class of polygons, called *lower polygons*, for which we achieve this complexity bound.

[1] Clarkson claimed a complexity of $O(m + \log m\ s)$, because he mistakenly relied on algorithms from the literature to optimize small integer programs, whereas they only solve the integer programming *feasibility* problem.

Outline of the parametric lattice width method. The key concept of Lenstra's polynomial algorithm for integer programming in fixed dimension [12] is the lattice width of a convex body. Let $K \subseteq \mathbb{R}^d$ be a convex body and $\Lambda \subseteq \mathbb{R}^d$ be a lattice. The *width* of K along a direction $c \in \mathbb{R}^d$ is the quantity $w_c(K) = \max\{\, c^{\mathrm{T}}x \mid x \in K \,\} - \min\{\, c^{\mathrm{T}}x \mid x \in K \,\}$. The *lattice width* of K, $w_\Lambda(K)$, is the minimum of its widths along nonzero vectors c of the dual lattice Λ^* (see Section 2.1 for definitions related to lattices). For the standard integer lattice $\Lambda = \mathbb{Z}^d$, we denote $w_{\mathbb{Z}^d}(K)$ by $w(K)$ and call this number the *width* of K.[2] Thus if a convex body K has lattice width ℓ, then its lattice points can be covered by $\lfloor \ell + 1 \rfloor$ parallel lattice hyperplanes. If K does not include any lattice points, then K must be "flat" along a nonzero vector in the dual lattice. This fact is known as Khinchin's *flatness theorem* (see [8]).

Theorem 1 (Flatness theorem). *There exists a constant f_d depending only on the dimension d, such that each convex body $K \subseteq \mathbb{R}^d$ containing no lattice points of Λ has lattice width at most f_d.*

Lenstra [12] applies this fact to the *integer programming feasibility* problem as follows: Compute the lattice width ℓ of the given d-dimensional polyhedron P. If $\ell > f_d$, then one is certain that P contains integer points. Otherwise all lattice points in P are covered by at most $f_d + 1$ parallel hyperplanes. Each of the intersections of these hyperplanes with P corresponds to a $(d-1)$-dimensional feasibility problem. These are solved recursively. The actual integer programming *optimization* problem is reduced to the feasibility problem via binary search. This brings an additional factor of s into the running time.

Our approach avoids this binary search by letting the objective function slide into the polyhedron, until the lattice width of the truncated polyhedron equals f_d. The approach can roughly be described for $d = 2$ as follows. For solving the integer program

$$\max\{\, c^{\mathrm{T}}x \mid (x_1, x_2)^{\mathrm{T}} \in P \cap \mathbb{Z}^2 \,\} \tag{1}$$

over a polygon P, we determine the smallest $\ell \in \mathbb{Z}$ such that the width of the truncated polyhedron $P \cap (c^{\mathrm{T}}x \geq \ell)$ is at most f_2. The optimum of (1) can then be found among the optima of the constant number of 1-dimensional integer programs formed by hyperplanes of the corresponding flat direction. We shall describe this parametric approach more precisely in Section 3. The core of the algorithm, which allows us to solve the parametric problem in essentially one single shortest vector computation, is presented in Section 4.3 (Proposition 4).

In the remaining part of the paper, we restrict our attention to the 2-dimensional case of integer programming. We believe that the flatness constant of Theorem 1 in two dimensions is $f_2 = 1 + \sqrt{4/3}$, but we have not found any result of this sort in the literature.

[2] This differs from the usual geometric notion of width, which is the minimum of w_c over all unit vectors c.

Complexity models. We analyze our algorithms both in the *arithmetic complexity* model and in the *bit complexity* model. The arithmetic model treats arithmetic operations $+, -, *$ and $/$ as unit-cost operations. This is the most common model in the analysis of algorithms and it is appropriate when the numbers are not too large and fit into one machine word. In the bit complexity model, every single bit-operation is counted. Addition and subtraction of s-bit integers takes $O(s)$ time. The current state of the art method for multiplication [19] shows that the bit complexity $M(s)$ of multiplication and division is $O(s \log s \log\log s)$, see [1, p. 279]. The difference between the two models can be seen in the case of the gcd-computation, which is an inherent ingredient of integer programming in small dimension: The best algorithm takes $\Theta(s)$ arithmetic operations, which amounts to $O(M(s)s)$ bit operations. However, a gcd can be computed in $O(M(s)\log s)$ bit operations [17]. The bit complexity model permits a more refined analysis of the asymptotic behavior of such algorithms.

2 Preliminaries

The symbols $\mathbb{N}$ and $\mathbb{N}_+$ denote the nonnegative and positive integers respectively. The *size* of an integer a is the length of its binary encoding. The size of a vector, a matrix, or a linear inequality is defined as the size of the largest entry or coefficient occurring in it. The *standard triangle* T^0 is the triangle with vertices $(0,0)$, $(1,0)$ and $(0,1)$.

The general 2-variable integer programming problem is as follows: given an integral matrix $A \in \mathbb{Z}^{m\times 2}$ and integral vectors $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^2$, determine $\max\{\, c^{\mathrm{T}}x \mid x \in P(A,b) \cap \mathbb{Z}^2 \,\}$, where $P = P(A,b) = \{\, x \in \mathbb{R}^2 \mid Ax \le b \,\}$ is the polyhedron defined by A and b.

We can assume without loss of generality that P is bounded (see e.g. [20, p. 237]). We can also restrict ourselves to problems where c is the vector $(0,1)^{\mathrm{T}}$, by means of an appropriate unimodular transformation. These operations (as well as all the other reductions and transformations that will be applied in this paper) increase the size of the involved numbers by at most a constant factor. Therefore we define the 2-variable integer programming problem as follows.

Problem 1 (2IP). Given an integral matrix $A \in \mathbb{Z}^{m\times 2}$ and an integral vector $b \in \mathbb{Z}^m$ defining a polygon $P(A,b)$, determine $\max\{\, x_2 \mid x \in P(A,b) \cap \mathbb{Z}^2 \,\}$.

2.1 The GCD, Best Approximations, and Lattices

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 $a_0, a_1, \ldots, a_k \in \mathbb{N}_+$, where $a_i,\ i \ge 2$ 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 $M^{(j)} = \prod_{i=1}^{j} \left(\begin{smallmatrix} q_i & 1 \\ 1 & 0 \end{smallmatrix}\right),\ 0 \le j \le k-1$. 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 representation $\gcd(a_0,a_1) = ua_0 + va_1$ with two integers u, v with $|u| \le a_1$ and $|v| \le a_0$ can be computed with the extended

Euclidean algorithm with $O(s)$ arithmetic operations or with $O(M(s)\log s)$ bit operations [17]. More generally, given two integers $a_0, a_1 > 0$ and some integer K with $a_0 > K > \gcd(a_0, a_1)$, one can compute the elements a_i and a_{i+1} of the remainder sequence $a_0, a_1, \dots, a_k$ such that $a_i \geq L > a_{i+1}$, together with the matrix $M^{(i)}$ with $O(M(s)\log s)$ bit operations using the so-called *half-gcd* approach [1, p. 308].

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

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 $a > b \geq 0$, called the *Hermite normal form, HNF* of Λ. The Hermite normal form can be computed with an extended-gcd computation and a constant number of arithmetic operations. The *dual lattice* of $\Lambda(A)$ is the lattice $\Lambda^*(A) = \{\, x \in \mathbb{R}^2 \mid x^{\mathrm{T}} v \in \mathbb{Z},\ \forall v \in \Lambda(A) \,\}$. It is generated by $(A^{-1})^{\mathrm{T}}$. A *shortest vector* of Λ (w.r.t. some given norm) is a nonzero vector of Λ with minimal norm. A shortest vector of a 2-dimensional lattice $\Lambda(A)$, where A has size at most s, can be computed with $O(s)$ arithmetic operations [11]. Asymptotically fast algorithms with $O(M(s)\log s)$ bit operations have been developed by Schönhage [18] and Yap [21], see also Eisenbrand [3] for an easier approach.

2.2 Homothetic Approximation

We say that a body P *homothetically approximates* another body Q with *homothety ratio* $\lambda \geq 1$, if $P + t_1 \subseteq Q \subseteq \lambda P + t_2$ for some translation $x \mapsto x + t_1$ and some homothety (scaling and translation) $x \mapsto \lambda x + t_2$.

This concept is important for two reasons: (i) The lattice width of Q is determined by the lattice width of P up to a multiplicative error of at most λ, i.e., $w_\Lambda(P) \leq w_\Lambda(Q) \leq \lambda\, w_\Lambda(P)$. (ii) A general convex body Q can be approximated by a simpler body P; for example, any plane convex body can be approximated by a triangle with homothety ratio 2.

In this way, one can compute an approximation to the width of a triangle. Let $a\colon x \mapsto Bx + t$ be some affine transformation. Clearly the width $w(K)$ of a convex body K is equal to the lattice width $w_{\Lambda(B)}$ of the transformed body $a(K)$. Thus we get the following lemma.

Lemma 1. *Let $T \subseteq \mathbb{R}^2$ be a triangle which is mapped to the standard triangle T^0 by the affine transformation $x \mapsto Bx + t$. Let $\bar{v}$ be a shortest vector of $\Lambda^*(B)$ with respect to the ℓ_2-norm. Then*

$$(1 - \sqrt{1/2})\ \|\bar{v}\|_2 \leq w(T) \leq 1/\sqrt{2}\ \|\bar{v}\|_2.$$

Moreover, the integral vector $v := B^{\mathrm{T}}\bar{v}$ *is a good substitute for the minimum-width direction:*

$$w(T) \leq w_v(T) \leq (\sqrt{2}+1)\, w(T) \qquad \square$$

With linear programming, one can easily find a good approximating triangle $T \subseteq P$ for a given polygon $P = P(A,b)$. For example, we can take the longest horizontal segment ef contained in P. It is characterized by having two parallel supporting lines through e and f which enclose P, and it can be computed as a linear programming problem in three variables in $O(m)$ steps, by Megiddo's algorithm [13]. (Actually, one can readily adapt Megiddo's simple algorithm for *two-variable* linear programming to this problem.) Together with a point $g \in P$ which is farthest away from the line through e and f (this point can be found by another linear programming problem), we obtain a triangle $T = efg$ which is a homothetic approximation of P with homothety ratio 3. Together with the previous lemma, and the known algorithms for computing shortest lattice vectors, we get the following lemma.

Lemma 2. *Let* $P \subseteq \mathbb{R}^2$ *be a polygon defined by* m *constraints each of size* s. *Then one can compute with* $O(m+s)$ *arithmetic operations or with* $O(m + \log s)M(s)$ *bit operations an integral direction* $v \in \mathbb{Z}^2$ *with*

$$w_v(P) = \Theta(w(P)). \qquad \square$$

2.3 "Checking the Width"

We will several times use the following basic subroutine, which we call *checking the width.*

For a given polygon P, we first compute its approximate lattice width by Lemma 2, together with a direction v. This gives us an interval $[K, \alpha K]$ for the lattice width of P. If $K \geq f_2 + 1$, then we say that P is *thick*, and we know that P contains an integral point. This is one possible outcome of the algorithm. Otherwise, P is *thin* and we solve 2IP for P as follows. Enumerate the at most $\alpha(f_2+1) = O(1)$ lattice lines through P, solving a one-dimensional integer program for each of them. We may find that P is *empty*, that is, it contains no integral point, or otherwise find the optimum point in P. These are the other two possible results of the algorithm, and this will always mean that no further processing of P is required. It is easy to see that the following bounds hold.

Lemma 3. *Checking the width takes* $O(m+s)$ *arithmetic operations or* $O(m + \log s)M(s)$ *bit operations.* $\square$

3 The Approximate Parametric Lattice Width (APLW) Problem

As in the case of Lenstra's algorithm, the lattice width is approximated via homothetic approximations and a shortest vector computation. This brings in some error which complicates the parametric lattice width method described in the introduction. The following problem, called *approximate parametric lattice width problem*, APLW for short, is an attempt to live up to the involved approximation error. If P is a polygon, we denote by P_ℓ the *truncated polygon* $P_\ell = P \cap (x_2 \geq \ell)$.

Problem 2 (APLW). This problem is parameterized by an approximation ratio $\gamma \geq 1$. The input is a number $K \in \mathbb{N}$ and a polygon $P \subseteq \mathbb{R}^2$ with $w(P) \geq K$. The task is to find some $\ell \in \mathbb{Z}$ such that the width of the truncated polygon P_ℓ satisfies

$$K \leq w(P_\ell) \leq \gamma\, K.$$

Integer programming can be reduced to the APLW problem:

Proposition 1. *Suppose that the APLW problem with any fixed approximation ratio γ can be solved in $A(m,s)$ bit operations or in $\widetilde{A}(m,s)$ arithmetic operations for a polygon P, described by m constraints of size at most s. Then 2IP can be solved in $T(m,s)$ bit operations or in $\widetilde{T}(m,s)$ arithmetic operations, with*

$$T(m,s) = O(A(m,s) + (m + \log s)\ M(s)),$$
$$\widetilde{T}(m,s) = O(\widetilde{A}(m,s) + m + s).$$

Proof. First we check the width of P. If P is thin we are done with the claimed time bounds (see Lemma 3). Otherwise solve APLW for $K = f_2 + 1$, yielding an integer $\ell \in \mathbb{Z}$ such that $f_2 + 1 \leq w(P_\ell) \leq \gamma\,(f_2+1)$. Then the polytope $P_\ell = P \cap (x_2 \geq \ell)$ must contain an integer point. Therefore the optimum of 2IP over P is the optimum of 2IP over P_ℓ. Compute an integral direction v with $w_v(P_\ell) = O(1)$ by Lemma 2. As in the case of checking the width, the optimum can then be found among the corresponding constant number of univariate integer programs. □

4 Solving the Integer Program

An *upper polygon* P has a horizontal line segment ef as an edge and a pair of parallel lines through the points e and f enclosing P, and it lies above ef. A *lower polygon* is defined analogously, see Fig. 1. We now describe efficient algorithms for APLW for upper triangles and lower polygons. This enables us to solve 2IP for polygons with a fixed number of constraints. Polygons described by a fixed number of constraints are the base case of our prune-and-search algorithm for general 2IP.

4.1 Solving APLW for Upper Triangles

Let T be an upper triangle. By translating T, we can assume that the top vertex is at the origin, and hence T is described by inequalities of the form $Ax \leq 0,\ x_2 \geq L$, where $L < 0$. All the truncated triangles $T_\ell = T \cap (x_2 \geq \ell)$ for $0 > \ell \geq L$ are scaled copies of T, and the width of T_ℓ scales accordingly:

$w(T_\ell) = |\ell|\, w(T_{-1})$. Therefore we simply have to compute an approximation to the width of T by Lemma 1 and choose the scaling factor $|\ell|$ accordingly so that $K \le w(T_\ell) \le \gamma K$ holds. Hence we have the following fact.

Proposition 2. *APLW can be solved with $O(s)$ arithmetic operations or with $O(M(s) \log s)$ bit operations for an upper triangle which is described by constraints of size at most s.*

4.2 Solving APLW for Lower Polygons

Since the width is invariant under translation, we can assume that the left vertex e of the upper edge ef is at the origin. We want to find an $\ell \in \mathbb{Z}$ with $K \le w(P_\ell) \le \gamma\, K$ for some constant $\gamma \ge 1$.

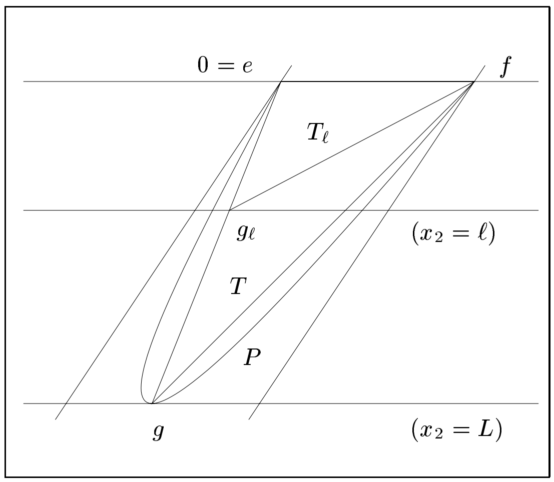

Fig. 1. The approximation of P_ℓ

Let $g = (g_1, g_2) \in P$ be a vertex of P with smallest second component $g_2 = L$. Let g_ℓ be the point of intersection between the line segment eg and the line $x_2 = \ell$, for $0 > \ell \ge L$ and denote the triangle $0 f g_\ell$ by T_ℓ (see Fig. 1).

Lemma 4. *The triangle T_ℓ is a homothetic approximation to the truncated lower polygon P_ℓ with homothety ratio 2.* $\square$

Thus we can solve APLW for P by finding the largest $\ell \in \mathbb{Z}$, $0 \ge \ell \ge L$ such that $w(T_\ell) \ge K$. For any 2×2 matrix A and a number p, we use the notation

$$A_p := \begin{pmatrix} 1 & 0 \\ 0 & p \end{pmatrix} A$$

for the matrix whose second row is multiplied by p. If the matrix B maps the triangle T_{-1} to the standard triangle T^0, then $B_{1/|\ell|}$ maps T_ℓ to T^0. By Lemma 4 and Lemma 1 we have the relation

$$(1 - \sqrt{1/2})\ \|v\|_2 \leq w(P_\ell) \leq \sqrt{2}\ \|v\|_2,$$

where v is a shortest vector of $\Lambda^*(B_{1/|\ell|})$ w.r.t. the ℓ_2-norm.

We can thus solve APLW by determining the smallest $p \in \mathbb{N}$ such that the length of a shortest vector v of $\Lambda^*(B_{1/p})$ w.r.t. the ℓ_2-norm satisfies the relation $\|v\|_2 \geq (1 - \sqrt{1/2})^{-1}\ K$. Since one has $\|v\|_\infty \leq \|v\|_2 \leq \sqrt{2}\|v\|_\infty$ we can as well search for the smallest p such that

$$\|v\|_\infty \geq (1 - \sqrt{1/2})^{-1} K,$$

where v is a shortest vector of $\Lambda^*(B_{1/p})$ w.r.t. the ℓ_∞-norm. Observe that one has $\Lambda^*(B_{1/p}) = \Lambda(((B^{-1})^{\mathrm{T}})_p)$. This shows that we can translate APLW into the following problem which we call the *parametric shortest vector problem*. The parameter is the factor p with which the second coordinate of the lattice is multiplied and we want to find the largest value of p such that the norm of the shortest vector does not exceed a given bound.

Problem 3 (PSV). Given a nonsingular matrix $A \in \mathbb{Q}^{2\times 2}$ and a constant $K \in \mathbb{N}_+$, find the largest $p \in \mathbb{N}_+$ such that $\|v\|_\infty \leq K$, where v is a shortest vector of $\Lambda(A_p)$ w.r.t. the ℓ_∞-norm.

The following statement is proved in [3]. It shows that a shortest vector of a 2-dimensional integral lattice can be found among the best approximations a rational number computed from the Hermite normal form of the lattice.

Proposition 3. *Let $\Lambda \subseteq \mathbb{Z}^2$ be given by its Hermite normal form $\left(\begin{smallmatrix} a & b \\ 0 & c \end{smallmatrix}\right) \in \mathbb{Z}^{2\times 2}$. 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 it is of the form $\left(\begin{smallmatrix} -xa+yb \\ yc \end{smallmatrix}\right)$, $x \in \mathbb{N}$, $y \in \mathbb{N}_+$ where the fraction x/y is a best approximation to the number b/a.*

By the relation between best approximations and the remainder sequence of the Euclidean algorithm we can now efficiently solve PSV.

Proposition 4. *PSV can be solved with $O(s)$ arithmetic operations or with $O(M(s)\log s)$ bit operations for matrices A and integers K of size s.*

Proof. Assume without loss of generality that A is an integral matrix. Let $\left(\begin{smallmatrix} a & b \\ 0 & c \end{smallmatrix}\right)$ be the HNF of A, the HNF of $\Lambda(A_p)$ is then the matrix $\left(\begin{smallmatrix} a & b \\ 0 & pc \end{smallmatrix}\right)$. Either $\left(\begin{smallmatrix} a \\ 0 \end{smallmatrix}\right)$ or $\left(\begin{smallmatrix} b \\ pc \end{smallmatrix}\right)$ is a shortest vector (these cases can be treated easily), or there exists a shortest vector $\left(\begin{smallmatrix} -xa+yb \\ pyc \end{smallmatrix}\right)$ such that x/y is a best approximation of b/a, thus a convergent of b/a.

Since we want to maximize p we have to find the convergent x/y of b/a with minimal $y \geq 1$ which satisfies $|-xa + yb| \leq K$. This convergent yields the candidate with which we can achieve a best scaling factor p. The best scaling factor is then simply $p = \lfloor K/(yc) \rfloor$.

The required convergent x/y can be determined by exploiting the relation between convergents and the Euclidean algorithm. Let $a_0, a_1, \ldots, a_k$ be the remainder sequence of $b = a_0$ and $a = a_1$. Multiplying the equation $\left(\begin{smallmatrix} a_0 \\ a_1 \end{smallmatrix}\right) = M^{(i)} \left(\begin{smallmatrix} a_i \\ a_{i+1} \end{smallmatrix}\right)$ by the inverse of the unimodular matrix $M^{(i)}$ gives the following equation for the i-th convergent $x/y = M^{(i)}_{1,1}/M^{(i)}_{2,1}$:

$$\pm(-xa + yb) = \pm(-M^{(i)}_{1,1} a_1 + M^{(i)}_{2,1} a_0) = a_{i+1}$$

Since subsequent convergents have strictly increasing denominators, we are looking for the first index i with $a_{i+1} \leq K$. This index is determined by the conditions $a_i > K$ and $a_{i+1} \leq K$. As mentioned in Section 2.1, the corresponding convergent $x/y = M^{(i)}_{1,1}/M^{(i)}_{2,1}$ can be computed with $O(M(s)\log(s))$ bit operations, or with $O(s)$ arithmetic operations. □

Theorem 2. *APLW for lower polygons defined by m constraints of size at most s can be solved with $O(m + s)$ arithmetic operations or with $O(m + \log s)M(s)$ bit operations.*

Proof. After one has found the point g which minimizes $\{x_2 \mid x \in P\}$ with Megiddo's algorithm for linear programming [13, 14] one has to solve PSV for the matrix B which maps T_{-1} to the standard triangle. The time bound thus follows from Proposition 4. □

4.3 An Efficient Algorithm for 2IP with a Fixed Number of Constraints

Theorem 3. *A 2IP problem defined by a constant number of constraints of size at most s can be solved with $O(s)$ arithmetic operations or with $O(M(s)\log s)$ bit operations.*

Proof. We compute the underlying polygon, triangulate it, and cut each triangle into an upper and a lower triangle. We get a constant number of 2IP's on upper and lower triangles, defined by inequalities of size $O(s)$. The complexity follows thus from Proposition 1, Proposition 2, and Theorem 2. □

4.4 Solving 2IP for Upper Polygons

It follows from Sect. 4.2 that 2IP over lower polygons can be solved with $O(m+s)$ basic operations or with $O((m + \log s)M(s))$ bit operations. Any polygon can be dissected into an upper and a lower part (by solving a linear programming problem); thus we are left with solving 2IP for upper polygons. Unfortunately, we cannot solve APLW for upper polygons directly. Instead, we use Megiddo's prune-and-search technique [13] to reduce the polygon to a polygon with a constant number of sides, to which the method from Theorem 3 in the previous section is then applied. This procedure works for general polygons and not only for upper polygons.

Our algorithm changes the polygon P by discarding constraints and by introducing bounds $l \leq x_2 \leq u$ such that the solution to 2IP remains invariant. Initially we check the width of P. If P is thin, we are done. Otherwise, we start with $l = -\infty$ and $u = \infty$. We gradually narrow down the interval $[l, u]$ and at the same time remove constraints that can be ignored without changing the optimum.

One iteration proceeds as follows: Ignoring the constraints of the form $l \leq x_2 \leq u$, the m constraints defining P can be classified into *left* and *right* constraints, depending on whether the feasible side lies to their right or left, respectively. We arbitrarily partition all left constraints into pairs and compute the intersection points of their corresponding lines, and similarly for the right constraints. We get roughly $m/2$ intersection points, and we compute the median μ of their x_2-coordinates. Now we check the width of P_μ. If P_μ is thick, we replace l by μ. In addition, for each of the $m/4$ intersection points *below* μ, there is one constraint among the two constraints defining it which is redundant in the new range $\mu \leq x_2 \leq u$. Removing these constraints reduces the number of constraints by a factor of $\frac{3}{4}$.

If P_μ is thin and contains integer points, we are done. If P_μ is empty, we replace u by μ. We remove constraints as above, but we now consider the intersection points *above* μ.

In this way, after $O(\log m)$ iterations, we have either solved the problem, or we have reduced it to a polygon with at most four constraints, which can be solved by Theorem 3.

Each iteration involves one operation of checking the width, plus $O(m)$ arithmetic operations for computing intersections and their median. Since the number m of constraints is geometrically decreasing in successive steps, we get a total of $O(m + \log m\ s)$ arithmetic operations and $O(m + \log m \log s)M(s)$ bit operations. The additional complexity for dealing with the final quadrilateral is dominated by this. This gives rise to our main result.

Theorem 4. *The two-variable integer programming problem with m constraints of size at most s can be solved in $O(m + \log m\ s)$ arithmetic operations or in $O(m + \log m \log s)M(s)$ bit operations.*

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*, 2001. to appear. `http://www.mpi-sb.mpg.de/~eisen/report_lattice.ps.gz`

4. 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.
5. 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.
6. N. Kanamaru, T. Nishizeki, and T. Asano. Efficient enumeration of grid points in a convex polygon and its application to integer programming. *Int. J. Comput. Geom. Appl.*, 4(1):69–85, 1994.
7. R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.
8. R. Kannan and L. Lovász. Covering minima and lattice-point-free convex bodies. *Annals of Mathematics*, 128:577–602, 1988.
9. Ravindran Kannan. A polynomial algorithm for the two-variable integer programming problem. *Journal of the Association for Computing Machinery*, 27(1):118–122, 1980.
10. D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1969.
11. J. C. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1:142–186, 1980.
12. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.
13. N. Megiddo. Linear time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12:759–776, 1983.
14. N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the Association for Computing Machinery*, 31:114–127, 1984.
15. H. E. Scarf. Production sets with indivisibilities. Part I: generalities. *Econometrica*, 49:1–32, 1981.
16. H. E. Scarf. Production sets with indivisibilities. Part II: The case of two activities. *Econometrica*, 49:395–423, 1981.
17. A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. (Fast computation of continued fraction expansions). *Acta Informatica*, 1:139–144, 1971.
18. 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.
19. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen (Fast multiplication of large numbers). *Computing*, 7:281–292, 1971.
20. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley, 1986.
21. C. K. Yap. Fast unimodular reduction: Planar integer lattices. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 437–446, Pittsburgh, 1992.
22. L. Ya. Zamanskij and V. D. Cherkasskij. A formula for determining the number of integral points on a straight line and its application. *Ehkonomika i Matematicheskie Metody*, 20:1132–1138, 1984.