1 710 078

9 0 C 1 0

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2285–2299

COMPUTING TWO-DIMENSIONAL INTEGER HULLS[*]

WARWICK HARVEY[†]

Abstract. An optimal algorithm is presented for computing the smallest set of linear inequalities that define the integer hull of a possibly unbounded two-dimensional convex polygon R. Input to the algorithm is a set of linear inequalities defining R, and the integer hull computed is the convex hull of the integer points of R. It is proven that the integer hull has at most $O(n \log A_{max})$ inequalities, where n is the number of input inequalities and A_{max} is the magnitude of the largest input coefficient. It is shown that the algorithm presented has complexity $O(n \log A_{max})$ and that this is optimal by proving that the integer hull may have $\Omega(n \log A_{max})$ inequalities in the worst case.

Key words. integer convex hull, linear inequalities, continued fractions

AMS subject classifications. 52C05, 11A55, 68Q25, 90C10

PII. S009753979528977X

1. Introduction and motivation.

In this paper we present an algorithm for finding the integer hull of a possibly unbounded two-dimensional (planar) convex region in polynomial time, given the set of linear inequalities defining the region. By the integer hull of a region we mean the convex hull of the integer points contained in that region. This algorithm grew out of work in the field of constraint logic programming (CLP) [11], specifically integer solvers for CLP. As a result, the algorithm is presented in an incremental formulation (that is, the input is processed one inequality at a time with the integer hull being updated fully after each) because that is what is most suitable for use in a CLP language. While a nonincremental formulation of the algorithm can likely be made to run faster in practice, it will not improve the (worst case) time complexity, since in section 7 we prove our algorithm is optimal.

We see this two-dimensional algorithm as a first step and hope to generalize it to an algorithm for efficiently finding the integer hulls of systems in an arbitrary number of dimensions. There are several reasons why being able to compute the integer hull of a system is useful. The integer hull provides a convenient and concise description of the set of all integer solutions of the input set, which is particularly useful if the number of such solutions is large or even infinite. It also implicitly answers the satisfiability question, which is the basis of constraint solvers for CLP. Thus if incrementally computing the integer hull is efficient enough, it may provide an interesting alternative to the partial solvers for integer CLP used to date (e.g., [7, 4, 10]). Finally, the integer hull allows integer linear optimization to be performed in polynomial time by using real optimization algorithms, though this is likely to be of limited utility unless a number of such optimizations are to be performed on the same feasible set.

In section 2 we discuss existing and related work. In section 3 we discuss some assumptions and notation. Section 4 shows how to find the integer hull of a single pair of inequalities, while section 5 uses this technique to compute the integer hull of

[*]Received by the editors August 1, 1995; accepted for publication (in revised form) September 18, 1997; published electronically August 16, 1999.
http://www.siam.org/journals/sicomp/28-6/28977.html

[†]Department of Computer Science, University of Melbourne, Parkville 3052, Australia. Present address: School of Computer Science and Software Engineering, Monash University, Clayton 3168, Australia (wharvey@cs.monash.edu.au).

a full set of inequalities. In section 6 we prove a time bound on the algorithm, and in section 7 we prove optimality. Finally, in section 8 we describe areas for future work.

2. Existing and related work. The only other existing algorithm for computing the integer hull of a polyhedron of which we are aware is presented in Schrijver [16, Chap. 23]. The algorithm works by successive approximation of the integer hull and is guaranteed to terminate after a finite number of such approximation steps. Each step involves finding the minimal total dual integral system describing the current approximation. Since such a system can be exponentially large, clearly each step can take exponential time. Moreover, Schrijver presents an example which shows that, even when restricted to the two-dimensional case, the number of steps taken may be exponential in the size of the problem. Our algorithm solves the given example in linear time.

Kannan [12] gave an algorithm for two-dimensional linear integer optimization in polynomial time (assuming the variables are nonnegative). Lenstra [13] showed that linear integer optimization in any fixed number of dimensions could be done in polynomial time. An interesting question is whether a similar result can be found for the problem of computing the integer hull, i.e., whether it can be done in polynomial time for any fixed number of dimensions.

There exist a number of algorithms for computing the convex hull of a finite set of points. Several algorithms and a review of others can be found in [9] and [15]. They include algorithms specifically for two dimensions, as well as algorithms for an arbitrary number of dimensions. Attempting to use these algorithms to help find the integer hull still leaves the problem of constructing the point set to give them and does not allow the handling of infinite solution sets.

Quite a number of algorithms have been developed for finding a (precise) finite description of all solutions of linear Diophantine equations and inequalities. For example, there are algorithms which focus on finding minimal solutions (e.g., [8]), algorithms which focus on finding nonambiguous solutions (e.g., [1]), and algorithms which focus on avoiding introducing slack variables (e.g., [2]). All of these algorithms resort to complete enumeration when the solution space is finite, which is inefficient if it is particularly large. Whether this matters, and whether the integer hull would be a better representation, depends on the end use for the algorithm.

3. Preliminaries. In the following, we assume that the coefficients of the input inequalities are integers; rational coefficients can be handled by appropriate multiplication. Because we are working in two dimensions, any given inequality C_i (for some i) can be written in the form

$$a_i x + b_i y \leq c_i.$$

For each such inequality, we assume that $\gcd(|a_i|, |b_i|) = 1$. If this is not the case (say, $\gcd(|a_i|, |b_i|) = k_i$), then it can be made so by replacing it with

$$\frac{a_i}{k_i} x + \frac{b_i}{k_i} y \leq \left\lfloor \frac{c_i}{k_i} \right\rfloor .$$

Note that the set of integer points satisfying this replacement inequality is exactly the set of integer points satisfying the original inequality.

We also use $C_i^=$ to denote the supporting line of C_i, namely,

$$a_i x + b_i y = c_i.$$

We make use of a number of results from the theory of continued fractions. For an introduction to continued fractions, see, for instance, [6] or [3, Chap. 32]. Briefly, any quantity X can be written as

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots}}},$$

where the a_k's (called *partial quotients*) are integers and all denominators are positive. The expansion terminates exactly when X is rational. The *principal convergents* of X are rational approximations p_k/q_k to X, obtained by truncating the continued fraction expansion of X after the kth term. p_k and q_k can be computed using the recurrence relations

$$\begin{array}{lll} p_0 = 1; & p_1 = a_1; & p_k = a_k p_{k-1} + p_{k-2},\ k \geq 2; \\ q_0 = 0; & q_1 = 1; & q_k = a_k q_{k-1} + q_{k-2},\ k \geq 2. \end{array}$$

The principal convergents are "good" approximations in that no simpler fraction is as close to X in value as any particular principal convergent. If any $a_k > 1$, then between p_{k-2}/q_{k-2} and p_k/q_k we define *intermediate convergents*

$$\frac{p_{k-2} + jp_{k-1}}{q_{k-2} + jq_{k-1}},\ j = 1 \cdots a_k - 1.$$

4. Finding the integer hull of a pair of inequalities.

Before we look at the full integer hull computation, we first show how to compute the integer hull of a pair of inequalities that are adjacent on the (real) hull. This is a key step in computing the integer hull of an arbitrary set of inequalities, which is described in section 5.

Assume the two inequalities are

$$\begin{array}{ll} a_1x + b_1y \leq c_1 & (C_1), \\ a_2x + b_2y \leq c_2 & (C_2). \end{array}$$

Let Δ be the determinant of the coefficient matrix (i.e., $\Delta = a_1b_2 - b_1a_2$). Without loss of generality, assume that the (counterclockwise) angle between C_1 and C_2 is less than 180 degrees so that $\Delta > 0$ (otherwise, swap C_1 and C_2).

We now compute the intersection point of $C_1^=$ and $C_2^=$, which will be an extreme point of the (real) feasible region. If this point is integral, then clearly we already have the integer hull and we are done. Otherwise, we need to perform cuts to obtain the integer hull, and we now describe how to compute exactly which cuts are needed to completely define the integer hull.

First, we perform a unimodular[1] transformation from x and y to X and Y such that one of the inequalities is transformed into an inequality in only one variable (say, X), specifically, an inequality of the form $X \leq c$. We would also like the other transformed inequality to have a nonpositive X coefficient and a positive Y coefficient to ensure a standard orientation for later parts of the algorithm. The unimodularity of the transformation ensures that computing the integer hull in XY space is equivalent

[1]A unimodular matrix (transformation) is an integral matrix with determinant ± 1. The main property of interest to us here is that both the transformation and its inverse preserve integrality: they both map integer points to integer points.

to doing it in xy space. To determine the transformation matrix, we thus need to solve

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} = \begin{bmatrix} t & u \\ 1 & 0 \end{bmatrix} \tag{4.1a}$$

such that

$$\text{(unimodularity)}\quad \alpha\delta - \beta\gamma = \pm 1, \tag{4.1b}$$
$$t \le 0, \tag{4.1c}$$
$$u > 0. \tag{4.1d}$$

The transformed inequalities will then be

$$tX + uY \le c_1, \tag{4.2a}$$
$$X \le c_2. \tag{4.2b}$$

LEMMA 4.1. *The transformation matrix*

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} = \begin{bmatrix} \alpha_0 + kb_2 & b_2 \\ \gamma_0 - ka_2 & -a_2 \end{bmatrix}$$

satisfies the conditions (4.1), *where* α_0 *and* γ_0 *are any integral solution of* $a_2\alpha_0 + b_2\gamma_0 = 1$ *and* $k = \left\lfloor \frac{-a_1\alpha_0 - b_1\gamma_0}{\Delta} \right\rfloor$.

Proof. It is straightforward that (4.1a), (4.1b), and (4.1d) hold. For (4.1c) we have

$$\begin{aligned} t &= a_1\alpha + b_1\gamma \\ &= a_1\alpha_0 + ka_1b_2 + b_1\gamma_0 - kb_1a_2 \\ &= \Delta\left\{\frac{a_1\alpha_0 + b_1\gamma_0}{\Delta} + \left\lfloor \frac{-a_1\alpha_0 - b_1\gamma_0}{\Delta} \right\rfloor\right\} \\ &\le 0, \text{ since } \Delta > 0. \quad \square \end{aligned}$$

$$= a_1\alpha_0 + b_1\gamma_0 + K\left[a_1b_2 - b_1a_2\right]$$
$$= a_1\alpha_0 + b_1\gamma_0 + K\Delta$$
$$= \Delta\left(\frac{a_1\alpha_0 + b_1\gamma_0}{\Delta}\right) + K$$

We now find the integer point on the supporting line of (4.2a) that is immediately on the feasible side of the supporting line of (4.2b), i.e., the point on the line with the largest X coordinate while still being feasible (call it (x_1, y_1)). Given an arbitrary integral point (x_0, y_0) on the supporting line of (4.2a) (which we can find by using, for example, a modified Euclid's algorithm), the point we are after is given by

$$(x_1, y_1) = \left(x_0 + \left\lfloor \frac{c_2 - x_0}{u} \right\rfloor u,\, y_0 - \left\lfloor \frac{c_2 - x_0}{u} \right\rfloor t\right).$$

We now translate the coordinate system so that this point becomes the new origin:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}.$$

This means the inequalities (4.2a) and (4.2b) now become

$$tX' + uY' \le 0, \tag{4.3a}$$
$$X' \le c_2 - x_1. \tag{4.3b}$$

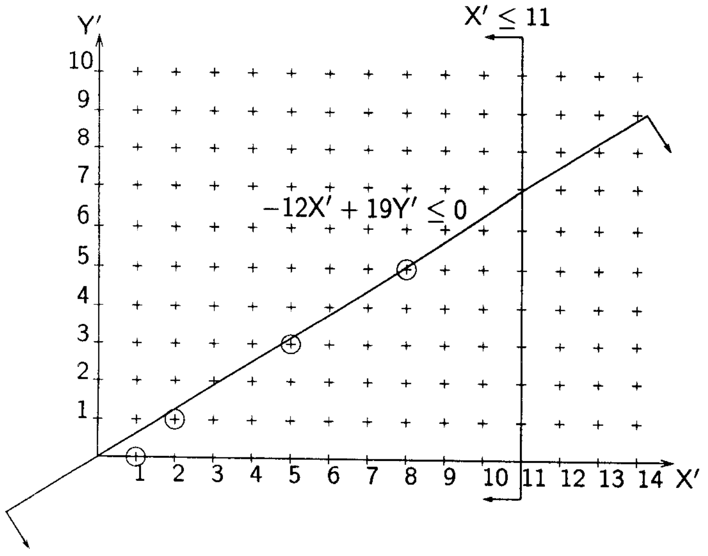

FIG. 4.1. *Two inequality integer hull example: the transformed initial inequalities.*

For example, if the initial inequalities are $5x + 2y \leq 8$ and $-2x + 3y \leq 4$, the transformed inequalities are $-12X' + 19Y' \leq 0$ and $X' \leq 11$, as shown in Figure 4.1.

We are now up to the "interesting" part, where we actually start computing the integer hull. If we think of the boundary of the real region as being a piece of string, and assume each integer point on the plane has a peg in it, finding the integer hull can be thought of as pulling the string tight. To compute this, we essentially need to find the points where the string bends. Due to our construction, there are no integer points on (4.3a) between the origin and where it meets (4.3b), so the first of these articulation points is the origin itself. The main task in finding the next one is determining the gradient of the next segment of "string." This involves rotating this "slack" section of (4.3a) (between the origin and where it meets (4.3b)) clockwise about the origin until it hits the first integer point(s) which satisfy (4.3b). The requirement that it satisfies (4.3b) means we only consider points with X'-coordinate no greater than $c_2 - x_1$. If we look at the problem in terms of the gradients of lines passing through the origin, the feasible integer points correspond to gradients p/q such that $q \leq c_2 - x_1$. Thus we are looking for the fraction p/q which is as close to $-t/u$ as possible (on the "less than" side) with a denominator no greater than $c_2 - x_1$. To find it, we make use of the following theorem.

THEOREM 4.2. *Suppose we are required to find the fraction, whose denominator does not exceed D, which most closely approximates, but is no greater than, the quantity X. If we construct a sequence of fractions containing all the odd principal convergents of X with their corresponding intermediate convergents (if such convergents exist), then the fraction we desire is the element of this sequence with the largest denominator no greater than D.*

Proof. See, for example, [3, Chap. 32, sects. 12–16]. □

Thus the fraction we are looking for can be found by searching the sequence of odd principal convergents for $-t/u$ and the corresponding intermediate convergents (these convergents correspond to the circled points in Figure 4.1). Since the intermediate convergents, if any, have denominators in arithmetic progression between the denominators of the principal convergents on either side, we actually only (construct and) search the sequence of odd principal convergents and then compute the appropriate intermediate convergent directly.

Let p/q be the fraction found by the search. Then this is the gradient of the line from the origin to the first integer point encountered when sweeping the hull segment around. Thus the next segment of the integer hull is given by

$$-pX' + qY' \leq 0. \tag{4.4}$$

Transformed back to the original coordinate system, it is

$$(p\delta + q\gamma)x - (p\beta + q\alpha)y \leq -px_1 + qy_1. \tag{4.5}$$

As the next step in our algorithm, we determine whether this new inequality (4.4) intersects (4.3b) (the vertical inequality) at an integer extreme point. If it does, then we have finished computing the integer hull of the original pair of inequalities. If it does not, then we need to proceed with computing the next cut. If we stay in $X'Y'$ space, this is just the first cut of the integer hull of (4.4) and (4.3b). The inequalities are already oriented appropriately, so we just need to translate them to bring them into our standard form. This involves finding the integer point on the supporting line of (4.4) that has greatest X' coordinate while still being feasible with respect to (4.3b) and moving it to the origin. Note that the "hard" part of finding this integer point, namely, finding an arbitrary point on the line, can be skipped because we already have one: the origin. Thus computing the translation is trivial. We also note that we need not compute the list of convergents for the gradient of (4.4) because it is a prefix of the list of convergents we already computed for $-t/u$ (p/q was basically formed by truncating the continued fraction expansion of $-t/u$).

5. Constructing the full two-dimensional integer hull. We now describe how to find the full two-dimensional integer hull of a set of inequalities in an incremental fashion. Adding the first inequality (to an empty existing set) is straightforward and obvious, so we concentrate on what happens when we add an inequality to an existing integer hull. We keep the inequalities sorted based on orientation (e.g., using the counterclockwise angle between the x-axis and the normal vector of the inequality). The question of what data structure to use to store the inequalities is deferred to section 6, where we analyze the computational complexity of the algorithm.

Since the existing system is an integer hull, we may assume that it is satisfiable and contains no redundant inequalities. Let the inequality being added be $C \equiv ax+by \leq c$. The first thing we do is check whether the augmented system is (real) unsatisfiable. This is equivalent to checking whether $ax + by > c$ (or equivalently $ax + by \geq c + 1$) is (real) redundant with respect to the existing system.

An inequality is (real) redundant if every point that is feasible with respect to every other inequality is also feasible with respect to this inequality. This means that if it is redundant, its supporting line lies on or outside the convex hull of the other inequalities and it is either parallel to the closest edge of the feasible region or there is a closest vertex. For the first case (parallel), we can just check whether there is another inequality with the same orientation and compare right-hand side constants. Otherwise, for the second case, we can find the inequalities that belong just before (C_{i-1}) and just after (C_{i+1}), the inequality being checked (C_i) in the sorted sequence, and see whether these form a vertex which is both feasible with respect to C_i and also the closest feasible point to $C_i^=$. This is the case if the (counterclockwise) angle between C_{i-1} and C_{i+1} is less than 180 degrees and the intersection of $C_{i-1}^=$ and $C_{i+1}^=$ is feasible with respect to C_i.

If we have discovered that the augmented system is unsatisfiable, then there is no integer hull and we stop. Otherwise, we proceed by checking whether the inequality we are adding is redundant. If it is, then we have nothing to do and the integer hull is unchanged. Otherwise, we proceed to check the inequalities immediately before and after it for redundancy. If the inequality immediately before (after) it is redundant, the redundant inequality is discarded and the next inequality before (resp., after) it is checked. This process continues until an irredundant inequality is found.

Note that at this point the set of inequalities define a convex hull (though most likely not an integer hull), and since all but one inequality (C) was part of the existing integer hull, all the vertices that are not on $C^=$ are guaranteed to be at integral locations.

We consider two cases, depending on whether or not the new inequality has a feasible integer point on its supporting line. The first case is where it does. We can just compute the integer hull of C pairwise with each of the adjacent inequalities, as per section 4 (as long as the relevant angle is less than 180 degrees; if it isn't, the region is unbounded between the inequalities and there are no cuts to compute), and add the resulting cuts to our description of the integer hull. Note that since the new inequality contains a feasible integer point, the pairwise integer hulls are guaranteed not to overlap. This is because at worst there is only one feasible integer point on the supporting line of the new inequality and the two pairwise integer hulls meet at this point. Note that any inequality involved in a nonintegral intersection before the cuts were made may have become redundant (if it only had one feasible integer point on its supporting line to begin with), so this must be checked for and the inequalities must be discarded if they are indeed redundant.

The second case is where there is no feasible integer point on the supporting line of the new inequality. In this case, computing the pairwise integer hulls of the new inequality with the inequalities on either side of it will generate cuts which overlap, and the new inequality is guaranteed to be redundant with respect to the final integer hull. The method described here shows how to deal with these overlapping cuts in a way which ensures that no more than one redundant cut is ever generated.

Let C_i be the new inequality with C_{i-1} and C_{i+1} the inequalities immediately to the clockwise and counterclockwise sides of C_i, respectively. We commence computing the cuts that form the integer hull of C_{i-1} and C_i, as per section 4, and continue until we make a cut (call it C_i') that causes C_i to become (real) redundant.[2] At this point, we now have at most one vertex remaining which is nonintegral, namely, the one at the intersection of C_i' and C_{i+1}. Thus to complete the integer hull, we simply compute the pairwise integer hull of C_i' and C_{i+1}. As before, some of the inequalities need to be checked for (real) redundancy. These are C_{i-1}, C_{i+1}, and C_i' (C_i is guaranteed to be redundant and thus need not be checked, just discarded).

This completes the integer hull algorithm.

6. Computational complexity. We now demonstrate that the algorithm presented has complexity $O(n \log A_{max})$. The time analysis is presented in terms of basic arithmetic operations $(+, -, \times, /)$.

[2]We start with C_{i-1} and C_i rather than C_i and C_{i+1} so that the pairwise hull algorithm generates cuts in an appropriate order ("outside in"), beginning with those adjacent to C_{i-1} and working toward C_i.

We start with the following definitions. Let A_{max} be the maximum absolute value of any coefficient of x or y in any original inequality added to the system.

We start by obtaining bounds for the coefficients of the inequalities that define the integer hull.

In the following, we assume all fractions p/q are in lowest terms (i.e., $\gcd(p,q) = 1$). Let p^+/q^+ (p^-/q^-) be the smallest (resp., largest) rational that is greater than (resp., less than) p/q with a denominator no larger than q.

LEMMA 6.1. *If p_{j-1}/q_{j-1} and p_j/q_j are consecutive principal convergents of p/q, then $p_jq_{j-1} - p_{j-1}q_j = (-1)^j$.*

Proof. See, for example, [3, Chap. 32, sect. 8]. □

LEMMA 6.2. *If p/q and p'/q' are two fractions such that $pq' - p'q = 1$ and $q > 0$, then no fraction can lie between them unless its denominator is greater than the denominator of both of them.*

Proof. See, for example, [3, Chap. 32, Sect. 12]. □

LEMMA 6.3. *Consider the two fractions*

$$\frac{p_{n-1}}{q_{n-1}}, \qquad \frac{p_n - p_{n-1}}{q_n - q_{n-1}}.$$

If $0 < p/q < 1$, then one of these is the closest above approximation (p^+/q^+) of $p/q = p_n/q_n$ and the other is the closest below approximation (p^-/q^-), depending on whether n is odd or even.

Proof. $0 < p/q < 1$ implies that $q > 0$, and also that $n \geq 1$, so that p_{n-1} and q_{n-1} are defined. Thus, using Lemmas 6.2 and 6.1, it is straightforward that these are the closest above and below approximations with denominators no greater than q. If n is odd, then p_{n-1}/q_{n-1} is the above approximation; if n is even, it is the below approximation. □

COROLLARY 6.4. *$pq^+ - p^+q = -1$ and $pq^- - p^-q = 1$ if $0 < p/q < 1$.*

Proof. This is obvious from Lemmas 6.3 and 6.1. □

LEMMA 6.5. *Consider two inequalities C_0 and C_n with $C_1 \cdots C_{n-1}$ being the cuts required to form the pairwise integer hull. Consider any cut C_i ($1 \leq i \leq n-1$). If the magnitude of the larger of the coefficients of C_i is greater than 1, then it is strictly smaller than the largest magnitude coefficient appearing in C_{i-1} and C_{i+1}.*

Proof. If $C_i \equiv a_ix + b_iy \leq c_i$, then we can assume, without loss of generality, that $0 \leq a_i \leq b_i$. However, $b_i > 1$ implies $b_i \neq a_i$ and $a_i \neq 0$ (since $\gcd(a_i, b_i) = 1$), so we have $0 < a_i < b_i$. Let a_i^+/b_i^+ and a_i^-/b_i^- be closest above and below approximations to a_i/b_i, respectively. Let $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$ be the (integral) intersection points of $C_i^=$ with $C_{i-1}^=$ and $C_{i+1}^=$, respectively, so that $x_0 > x_1$ and $y_0 < y_1$ (see Figure 6.1).Note that $x_0 - x_1 = kb_i$ and $y_1 - y_0 = ka_i$ for some integer $k > 0$. Also note that C_i is the cut required to form the integer hull of C_{i-1} and C_{i+1} and that there are no integer points which are infeasible with respect to C_i that are feasible with respect to C_{i-1} and C_{i+1} (i.e., the introduced cut excludes no feasible integer points). Consider the integer point $P_2 = (x_2, y_2) = (x_0 - b_i^+, y_0 + a_i^+)$. P_2 is infeasible with respect to C_i, so it must be infeasible with respect to at least one of C_{i-1} and C_{i+1}. We consider two possibilities:

1. Suppose P_2 is infeasible with respect to C_{i-1}. This means that the gradient of C_{i-1} must be between $a_i/-b_i$ and $a_i^+/-b_i^+$. By Corollary 6.4 and Lemma 6.2, any gradient that lies between these two has a denominator of magnitude larger than b_i. Since $b_i > a_i$, this means C_{i-1} must have a coefficient (namely, b_{i-1}) larger in magnitude than both a_i and b_i.

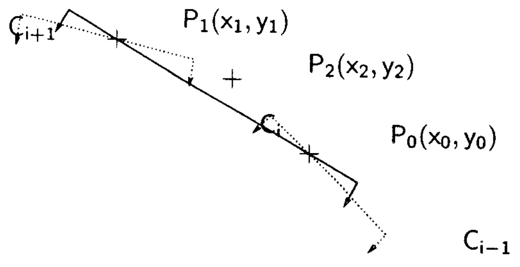

FIG. 6.1. *Determining bounds on the coefficients of adjacent cuts.*

2. Suppose P_2 is infeasible with respect to C_{i+1}. This means that the gradient of C_{i+1} must be between $a_i / -b_i$ and

$$\frac{y_1 - y_2}{-(x_1 - x_2)} = \frac{ka_i - a_i^+}{-(kb_i - b_i^+)}.$$

Again by Corollary 6.4 and Lemma 6.2, any gradient that lies between these two has a denominator of magnitude larger than b_i, which in turn means C_{i+1} must have a coefficient (namely, b_{i+1}) larger in magnitude than both a_i and b_i.

Since at least one of these two possibilities must be true, we have that at least one of C_{i-1} and C_{i+1} must have a coefficient larger in magnitude than those of C_i, as long as at least one of the coefficients of C_i is greater than 1. □

THEOREM 6.6. *For the integer hull of a pair of inequalities, the coefficients of the generated cuts are no larger in magnitude than the largest of those of the original inequalities.*

Proof. Let the two inequalities be C_0 and C_n with the generated cuts being $C_1 \cdots C_{n-1}$. Let A_i be the magnitude of the largest coefficient of C_i, $i = 0 \cdots n$. We note that no $A_i = 0$, since that would imply that $C_i \equiv 0x + 0y \leq c_i$. Let j be the smallest i such that A_i is minimal, so that $\forall i, A_i \geq A_j$. If $j > 0$, we have $A_{j-1} > A_j$, so $A_{j-1} > 1$. If $j > 1$, Lemma 6.5 implies $A_{j-2} > A_{j-1}$ since $A_j < A_{j-1}$. Further applications of Lemma 6.5 yield, in turn,

$$A_{j-2} < A_{j-3}, A_{j-3} < A_{j-4}, \ldots, A_1 < A_0.$$

Thus $\forall i \leq j, A_i \leq A_0$.

Let $k = \min\{i : i > j \wedge A_i > A_j\}$. If k is not defined (e.g., $j = n$), then $A_j = A_{j+1} = \cdots = A_n$. Otherwise, we have $A_j = A_{j+1} = \cdots = A_{k-1}$ and $A_k > A_{k-1}$. As before, repeated applications of Lemma 6.5 yield, in turn,

$$A_k < A_{k+1}, A_{k+1} < A_{k+2}, \ldots, A_{n-1} < A_n.$$

Thus $\forall i \geq j, A_i \leq A_n$.

This means that all A_i are no larger than the larger of A_0 and A_n, and the theorem is proved. □

THEOREM 6.7. *The coefficients of the inequalities defining an integer hull are no larger in magnitude than the largest coefficient of the input inequalities.*

Proof. Since all the inequalities in the system at any given time are either original input inequalities or are part of a pairwise integer hull of some combination of original inequalities and inequalities already in the system, the result follows by a simple induction argument on Theorem 6.6. □

Now that we have an upper bound on the size of the coefficients in the system, we can obtain an upper bound on the number of inequalities in the system.

LEMMA 6.8. *The integer hull of a pair of inequalities with largest magnitude coefficient A_{max} requires at most $O(\log A_{max})$ inequalities to define it.*

Proof. Each cut generated in the transformed space corresponds to either an odd principal convergent of the gradient $(-t/u)$ of the main transformed inequality or to an intermediate convergent between two such odd principal convergents. To achieve the bound, we start by showing that even if there are many intermediate convergents between two odd principal convergents, only one can correspond to a cut of the integer hull. Consider an intermediate convergent that is used for a cut. If p_{2k-1}/q_{2k-1} and p_{2k+1}/q_{2k+1} are the odd principal convergents it lies between, then it is of the form $\frac{p_{2k-1}+jp_{2k}}{q_{2k-1}+jq_{2k}}$ for some j. Note that j is chosen as large as possible subject to the bound on the denominator, so the "slack" between the denominator and the bound must be less than q_{2k}. Since the translation in preparation for the next cut reduces the bound by (at least) the size of the denominator selected, this means the new bound must be less than q_{2k}. This, in turn, precludes any other intermediate convergents between p_{2k-1}/q_{2k-1} and p_{2k+1}/q_{2k+1} from being used to generate cuts, since such a convergent must have a denominator of at least $q_{2k-1} + q_{2k}$.

Thus we can generate no more than two cuts for each odd principal convergent: one for the odd principal convergent itself and one for a corresponding intermediate convergent.

The number of principal convergents is the same as the number of steps in the Euclidean algorithm for finding g.c.ds, applied to $|t|$ and u. This is $O(\log(\min(|t|, u)))$ [5, p. 811], but

$$\min(|t|, u) \leq u = a_1\beta + b_1\delta = a_1b_2 - b_1a_2 = O(A_{max}^2)$$

(this last step by Theorem 6.7). So we have $O(\log A_{max})$ principal convergents and thus $O(\log A_{max})$ cuts generated. □

LEMMA 6.9. *When computing the integer hull of n (initial) inequalities with largest magnitude coefficient A_{max}, at most $O(n \log A_{max})$ cuts are added.*

Proof. Each time we add a new inequality to the integer hull, we perform at most two pairwise integer hull computations. By Lemma 6.8, each of these generates $O(\log A_{max})$ inequalities, and so after adding n (initial) inequalities, we can have added no more than $O(n \log A_{max})$ inequalities. □

LEMMA 6.10. *The integer hull of a set of any number of inequalities with coefficients of magnitude no larger than A_{max} requires at most $O(A_{max}^2)$ inequalities to define it.*

Proof. If the largest magnitude of an input coefficient is A_{max}, then by Theorem 6.7, the magnitudes of all coefficients in the integer hull are no greater than A_{max}. This means there are at most $O(A_{max}^2)$ different possible combinations of coefficients. Since there are no redundant inequalities in the integer hull, there can be no more than one inequality with a given combination of coefficients, and the result follows. □

We now proceed to analyze the time complexity of the algorithm. We start by analyzing the pairwise integer hull, and then we use that analysis to derive a time bound for the full algorithm.

THEOREM 6.11. *Computing the pairwise hull of two inequalities with largest magnitude coefficient A_{max} is $O(\log A_{max})$.*

Proof. To compute the integer hull of a pair of inequalities, the following steps are performed:

1. *If the intersection point is integral, we stop.* Finding the intersection point and determining whether it is integral is $O(1)$.

2. *Determine the transformation matrix.* The main step in determining the transformation matrix is finding a solution to $a_2\alpha + b_2\gamma = 1$; everything else is $O(1)$. Thus this step is $O(\log A_{max})$, using, for instance, a modified Euclid's algorithm to solve the equation.

3. *Compute the odd principal convergents.* We just compute all the principal convergents of $|t/u|$; this is $O(\log(\min(|t|, |u|))) = O(\log A_{max})$.

4. *Repeat the following steps until the intersection point is integral.* Each pass through this loop generates one cut, and so by Lemma 6.8, we loop no more than $O(\log A_{max})$ times.

(i) *Determine the appropriate translation.* The first time we perform this translation we must find an initial integer point on the supporting line of the "main" inequality, which is $O(\log A_{max})$, and then find the correct integer point, which is $O(1)$. All subsequent translation computations start with an integer point given, so we just require the $O(1)$ adjustment.

(ii) *Search for the appropriate principal convergent.* If we do a simple linear search backwards in the list of principal convergents, this is $O(\log A_{max})$. However, no principal convergent checked in one pass needs be checked again in a subsequent pass, so we can amortize the search cost for a total of $O(\log A_{max})$ over all passes through the loop.

(iii) *Compute the appropriate intermediate convergent.* This is $O(1)$.

(iv) *Transform the new cut back to the original coordinate system.* This is $O(1)$.

The result follows directly from the above analysis. □

We now turn to the full integer hull algorithm.

THEOREM 6.12. *Incrementally computing the integer hull of n inequalities with largest magnitude coefficient A_{max} is $O(n \log A_{max})$.*

Proof. For this analysis, we assume the inequalities in the system are stored in a level-linked (a,b)-tree, sorted based on their orientation. Level-linked (a,b)-trees are described in [14], along with proofs of the complexity results used here. Some modification of the standard operations on level-linked (a,b)-trees is required for our purposes, since our data is inherently circular in nature and wraps around from largest to smallest, but these do not affect the results.

Let N_j be the number of inequalities defining the existing integer hull just before we add the jth inequality ($C_j \equiv a_jx + b_jy \leq c_j$). By Lemma 6.10, N_j is $O(A_{max}^2)$. In particular, this means that $\log N_j$ is $O(\log A_{max})$, and thus we have bounds for various tree operations (search, split, concatenate) which are independent of the number of inequalities added so far.

To add the inequality to the system and recompute the integer hull, we perform the following steps:

1. *Tighten the inequality being added.* This step consists mainly of computing the g.c.d. of $|a_j|$ and $|b_j|$, which is $O(\log(\min(|a_j|, |b_j|))) = O(\log A_{max})$.

2. *Check satisfiability of augmented system.* This requires searching the data structure to find where the complement of C_j would go and accessing at most two adjacent items. This is $O(\log A_{max})$, and the actual check is $O(1)$.

3. *Eliminate initial redundancy.* This requires searching the data structure to find where C_j would go ($O(\log A_{max})$) and then performing a series of redundancy checks. Since no inequality can be found to be redundant more than once, and at most $O(n \log A_{max})$ inequalities are added to the system for n input inequalities (Lemma 6.9), we have a total of $O(n \log A_{max})$ eliminations performed, summed over all n incremental steps. Since all but a constant number of redundancy checks at each incremental step result in an elimination, this means $O(n \log A_{max})$ redundancy checks are performed in total. Accessing the adjacent inequalities to perform one of these checks is $O(1)$, as is the actual check, which means the total cost of redundancy checks over all n additions is $O(n \log A_{max})$. Since all the inequalities to be eliminated due to redundancy are adjacent to each other, we can delete them all at the same time by performing two splits of the tree, which is $O(\log A_{max})$ (we leave it split until we are ready to insert the new inequalities). Thus this step as a whole, summed over all n additions, is $O(n \log A_{max})$.

4. *Compute the cuts.* We treat this as two instances of finding the integer hull of a pair of inequalities. This is not strictly the case when there are no integer points on the feasible segment of the supporting line of the inequality being added, but the extra redundancy checks required have no bearing on the computational complexity. Thus this step is $O(\log A_{max})$, by Theorem 6.11.

5. *Add the cuts to the system.* Since all the inequalities added will be adjacent to each other, and will be inserted at the point the existing tree was split, we can insert them all at the same time. Constructing a new tree from a sorted set is linear in the number of items in the tree, so this is $O(\log A_{max})$ by Lemma 6.8. Then we just perform two concatenation operations to add the new tree between the two parts created in step 3. This is also $O(\log A_{max})$, so the step as a whole is $O(\log A_{max})$.

6. *Eliminate final redundancy.* This requires at most three redundancy checks and three deletions and hence is $O(\log A_{max})$.

Each step is either $O(\log A_{max})$ for each inequality added or is $O(n \log A_{max})$ summed over all n inequalities. Outputting the inequalities is linear in the number output, which is $O(n \log A_{max})$ by Lemma 6.9. The result follows. □

7. Proof of optimality.

We now show that our algorithm is optimal by demonstrating a worst case lower bound complexity of $\Omega(n \log A_{max})$. We do this by constructing a family of examples which generate $\Omega(n \log A_{max})$ output constraints.

LEMMA 7.1. *The integer hull of*

$$-2x + y \leq -1, \tag{7.1a}$$

$$-F_{2k+5}x + F_{2k+4}y \leq -1 \tag{7.1b}$$

has k cuts, where F_n is the nth Fibonacci number (with $F_0 = 0, F_1 = 1$).

Proof. Let $\phi = \frac{1+\sqrt{5}}{2}$ be the golden ratio. The continued fraction representation of ϕ is infinite with all partial quotients 1. Thus the sequence of closest below approximations to ϕ are given by p_{2j+1}/q_{2j+1} for $j \geq 0$, where p_i, q_i satisfy the recurrence relations

$$\begin{aligned} p_0 &= 1; & p_1 &= 1; & p_i &= p_{i-1} + p_{i-2}, i \geq 2; \\ q_0 &= 0; & q_1 &= 1; & q_i &= q_{i-1} + q_{i-2}, i \geq 2. \end{aligned}$$

Clearly, the p_i's and q_i's both form the Fibonacci sequence with $p_i = F_{i+1}$ and $q_i = F_i$. The below approximations are thus given by $F_{2j+2}/F_{2j+1}, j \geq 0$.

If we were to compute the (infinite) integer hull of $-\phi x + y \leq 0$ and $-x \leq -1$ (see Figure 7.1), then these below approximations clearly define the vertices of the integer

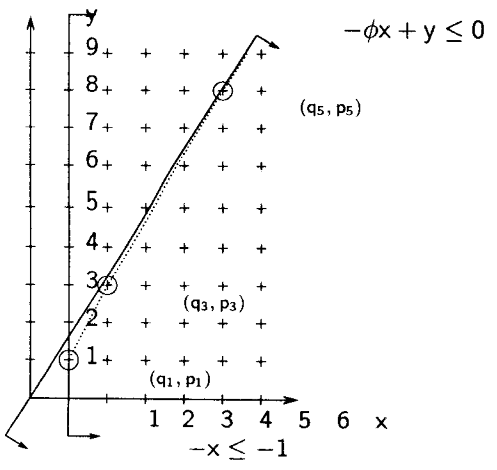

FIG. 7.1. *The infinite integer hull of* $-\phi x + y \leq 0$ *and* $-x \leq -1$.

hull. The inequality defining the jth edge (the edge between the jth and $(j+1)$th vertices) is then

$$-(p_{2j+3} - p_{2j+1})x + (q_{2j+3} - q_{2j+1})y \leq -(p_{2j+3} - p_{2j+1})q_{2j+1} + (q_{2j+3} - q_{2j+1})p_{2j+1},$$

i.e.,
$$-p_{2j+2}x + q_{2j+2}y \leq -p_{2j+2}q_{2j+1} + q_{2j+2}p_{2j+1},$$

i.e.,
$$-F_{2j+3}x + F_{2j+2}y \leq -1$$

(the last step by Lemma 6.1).

Note that we can select any two of these inequalities (say, $j = j_0$ and $j = j_1$), and the cuts needed to form the integer hull of the selected pair consist simply of all the inequalities between them in the above integer hull (namely, $j = j_0 + 1 \ldots j_1 - 1$). In particular, we can choose $j_0 = 0$ and $j_1 = k + 1$ and have the integer hull contain k cuts. The result follows. $\square$

LEMMA 7.2. *It is possible for a system with* $2m$ *inequalities with largest magnitude coefficient* $O(m\phi^{2k})$ *to require at least* mk *cuts to complete the integer hull.*

Proof. We note that we can perform both translations and unimodular transformations on the integer hull used in Lemma 7.1 without altering its basic structure. Applying the unimodular transformation $\begin{bmatrix} 1 & 0 \\ -i & 1 \end{bmatrix}$, we obtain

$$-(2+i)x + y \leq -1, \tag{7.2a}$$

$$-(F_{2k+5} + iF_{2k+4})x + F_{2k+4}y \leq -1. \tag{7.2b}$$

Note that, since $F_{2k+5}/F_{2k+4} > 1$, both the inequalities (and all the cuts) have gradients larger than $1 + i$, and no greater than $2 + i$, which means that the range of gradients for one value of i does not overlap with that of another. This means that, with appropriate translation, we can construct a system of $2m$ inequalities, by including a "copy" of (7.2) for all values of i from 0 to $m - 1$, such that the integer hulls of each component do not interfere with each other. By Lemma 7.1, each component has k cuts, so with m noninterfering components, we have (at least) mk cuts (depending on the exact translation used, there may be cuts between each "block").

The magnitude of the largest coefficient of the system is $F_{2k+5} + (m-1)F_{2k+4}$. Using the closed form expression for F_j in terms of ϕ ($F_j = \frac{1}{\sqrt{5}}(\phi^j - (-\phi)^{-j})$), we have that this is $O(m\phi^{2k+4}) = O(m\phi^{2k})$, and the result follows. ☐

THEOREM 7.3. *It is possible for a system of n inequalities with coefficients of magnitude no larger than A_{max} to have an integer hull consisting of $\Omega(n \log A_{max})$ inequalities.*

Proof. Consider Lemma 7.2 with $k = \Omega(\log m)$. Then $n = 2m, A_{max} = O(m\phi^{2k})$, and we have mk cuts. Thus $n \log A_{max} = O(m(k + \log m))$. But $k = \Omega(\log m)$, so $n \log A_{max} = O(mk)$. This means $mk = \Omega(n \log A_{max})$ and we have the result. ☐

THEOREM 7.4. *Our algorithm for computing two-dimensional integer hulls is optimal.*

Proof. By Theorem 7.3, any algorithm must be $\Omega(n \log A_{max})$ on some problem instances. By Theorem 6.12, our algorithm is $O(n \log A_{max})$ on all problem instances, and the result follows. ☐

8. Future work.

We have done some work toward developing an integer hull algorithm that works in three dimensions, but we have yet to establish whether this algorithm can be made to be polynomial time. If it can, then we hope to generalize to an arbitrary number of dimensions and tackle the very interesting question of whether such an algorithm is polynomial time for any fixed number of dimensions.

At some stage we also hope to determine whether such algorithms can be usefully employed in a constraint solver for a CLP language by buying enough of a reduction in domain sizes to offset the extra overhead incurred on more than just a few select problem classes.

Acknowledgments. The geometrical interpretation of continued fractions and their convergents used in this paper is from [6, Chap. IV, sect. 12] and appears to be due to Smith [17, Art. 20].

The author would like to thank Peter Stuckey for his comments on many drafts of this paper. The author would also like to thank the anonymous referees for their feedback, advice, and suggestions. In particular, credit must be given to one anonymous referee for a suggestion on how to obtain improved bounds on the coefficients of generated cuts (the bounds presented in an earlier version of this work were messy and not particularly tight, with the proofs hard to follow). While the ideas proposed by the referee are not used in the paper, the bounds yielded were sufficiently good for the whole question of optimality to be addressed and provided the motivation to find the proof of even tighter bounds that is presented in this version of the paper.

Finally, the author would like to thank Gary Eddy for his help in selecting an appropriate data structure for storing the inequalities.

REFERENCES

[1] H. ABDULRAB AND M. MAKSIMENKO, *General solution of systems of linear diophantine equations and inequations*, in Rewriting Techniques and Applications—6th International Conference, RTA-95, J. Hsiang, ed., Lecture Notes in Comput. Sci. 914, Springer-Verlag, Berlin, 1995, pp. 339–351.

[2] F. AJILI AND E. CONTEJEAN, *Complete solving of linear diophantine equations and inequations without adding variables*, in Principles and Practice of Constraint Programming—CP '95, U. Montanari and F. Rossi, eds., Lecture Notes in Comput. Sci. 976, Springer-Verlag, Berlin, 1995, pp. 1–17.

[3] G. CHRYSTAL, *Algebra–An Elementary Text-Book–Part* II, Adam and Charles Black, Edinburgh, 1889.

[4] P. CODOGNET AND D. DIAZ, *Compiling constraints in clp(FD)*, J. Logic Programming, 27 (1996), pp. 185–226.

[5] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, The MIT Electrical Engineering and Computer Science Series, MIT Press, Cambridge, MA, 1990.

[6] H. DAVENPORT, *The Higher Arithmetic*, 6th ed., Cambridge University Press, Cambridge, UK, 1992.

[7] M. DINCBAS, P. VAN HENTENRYCK, H. SIMONIS, A. AGGOUN, T. GRAF, AND F. BERTHIER, *The constraint logic programming language CHIP*, in Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88, Tokyo, Japan, Springer-Verlag, New York, 1988, pp. 693–702.

[8] E. DOMENJOD AND A. P. TOMÀS, *From Elliott-MacMahon to an algorithm for general linear constraints on naturals*, in Principles and Practice of Constraint Programming—CP '95, U. Montanari and F. Rossi, eds., Lecture Notes in Comput. Sci. 976, Springer-Verlag, Berlin, 1995, pp. 18–35.

[9] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[10] W. HARVEY AND P. J. STUCKEY, *A unit two variable per inequality integer constraint solver for constraint logic programming*, in Proceedings of the Twentieth Australasian Computer Science Conference (ACSC'97), Sydney, Australia, Macquarie University, Sydney, 1997, pp. 102–111.

[11] J. JAFFAR AND M. J. MAHER, *Constraint logic programming: A survey*, J. Logic Programming, 19/20 (1994), pp. 503–581.

[12] R. KANNAN, *A polynomial algorithm for the two variable integer programming problem*, J. ACM, 27 (1980), pp. 118–122.

[13] H. W. LENSTRA, JR., *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–547.

[14] K. MEHLHORN, *Data Structures and Algorithms 1: Sorting and Searching*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1984.

[15] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry—An Introduction*, Texts Monogr. Comput. Sci., Springer-Verlag, New York, 1985.

[16] A. SCHRIJVER, *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics, Wiley-Interscience, New York, 1986.

[17] H. J. S. SMITH, *A note on continued fractions*, in The Collected Mathematical Papers of Henry John Stephen Smith, vol. 2, J. W. L. Glaisher, ed., Clarendon Press, Oxford, UK, 1894, pp. 135–147.