Can a System of Linear Diophantine Equations be Solved in Strongly Polynomial Time?

Dorit S. Hochbaum *
Department of Industrial Engineering and Operations Research and
Walter A. Haas School of Business,
University of California, Berkeley
email: dorit@hochbaum.berkeley.edu

Anu Pathria †
Department of Industrial Engineering and Operations Research,
University of California, Berkeley
email: pathria@cimsim.berkeley.edu

Oct 17, 1994

Abstract

We demonstrate that the answer to the question posed in the title is "yes" and "no": "no" if the set of permissible operations is restricted to $\{+,-,\times,/,\mathsf{mod},<\}$; "yes" if we are also allowed a *gcd-oracle* as a permissible operation. It has been shown (see [Sto76, MST91]) that no strongly polynomial algorithm exists for the problem of finding the greatest common divisor (*gcd*) of two arbitrary integers, a and b. This result precludes the possibility of finding the set of solutions to a system of linear diophantine equations in strongly polynomial time. We show, however, that given an oracle that finds the *gcd* of two integers a and b and integer multipliers (x_0, y_0) satisfying $ax_0 + by_0 = \gcd(a, b)$, a system of linear diophantine equations *can* be solved in strongly polynomial time.

0 Introduction

The problem of solving a system of linear diophantine equations is that of finding the set of integer solutions to a system of linear equations. For a system, $A\mathbf{x} = \mathbf{b}$, of m equations in n variables, this problem can be solved using $O(n^3 \log \Delta)$ operations, where Δ is the magnitude of the determinant of a rank m submatrix of A (see [DKT87]). While this algorithm runs in polynomial time in the size of the input, the running time depends on the size of the entries of A; hence, the algorithm does not run in *strongly* polynomial time.

Recent results (see [Sto76, MST91]) have shown that no strongly polynomial algorithm exists for the problem of deciding whether or not two arbitrary integers are coprime; these results in

*Research supported in part by ONR contract N00014-91-J-1241.

†Author supported in part by an *NSERC '67* scholarship provided by the Natural Sciences and Engineering Research Council of Canada.

turn imply that no strongly polynomial algorithm exists for the problem of determining the *gcd* of two arbitrary integers. We show that, as a consequence of the lower bound provided for the *gcd* problem in [MST91], no strongly polynomial algorithm exists for the problem of finding the set of solutions to a system of linear diophantine equations in a complexity model allowing the operations $\{+,-,\times,/,\mathsf{mod},<\}$. We further show that the *gcd* operation is fundamental to solving linear diophantine equations; namely, if we are allowed to use a *gcd-oracle* as part of our set of operations, then we provide an algorithm that finds the set of solutions to a system of linear diophantine equations in strongly polynomial time. Our algorithm performs $O(n^3)$ operations, including $O(n^2)$ calls to the *gcd* oracle.

Our results amount to showing that the following two problems are *strongly polynomial time reducible* to each other: 1) find the *gcd* of two integers, a and b, along with multipliers x_0, y_0 such that $ax_0+by_0 = \gcd(a,b)$; and, 2) find the set of solutions to a system of linear diophantine equations. That is, there exists a strongly polynomial algorithm for the first problem if and only if there exists a strongly polynomial algorithm for the second. We provide first a proof for the easy (*if*) direction of the above result; this is done in Section 1. The reduction in the reverse (*only if*) direction is the subject of Section 2, and is the main result (Theorem 2.2) of this paper. Throughout this paper, we assume that our available set of operations is $\{+,-,\times,/,\mathsf{mod},<\}$.

1 A Lower Bound

The set of solutions, if any exist, to a system of linear diophantine equations, $A\mathbf{x} = \mathbf{b}$, where we assume A is a $m \times n$ full row rank integer matrix, is expressed by a particular solution, $\mathbf{x}_0$, and a set of linearly independent spanning vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_{n-m}$, so that

$$\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b},\, \mathbf{x} \in Z^n\} \;=\; \{\mathbf{x} \mid \mathbf{x} = \mathbf{x}_0 + \sum_{i=1}^{n-m} \lambda_i \mathbf{x}_i,\, \lambda_i \in Z, \mathbf{x}_i \in Z^n, \forall i\}\ .$$

By providing a strongly polynomial reduction from the *gcd* problem, we show that no strongly polynomial algorithm exists for the problem of finding the set of solutions to a system of linear diophantine equations. In contrast, one *can* find the set of solutions to system of linear equations over the *rationals* in strongly polynomial time (see [Edm67]).

Theorem 1.1 *Suppose there exists a strongly polynomial algorithm for the problem of finding the set of solutions to a system of linear diophantine equations. Then, for integers a and b, we can find* $\gcd(a,b)$ *and integers x_0, y_0 such that $ax_0 + by_0 = \gcd(a,b)$ in strongly polynomial time.*

Proof: The set of solutions, if any exist, to the diophantine equation

$$a\,x \;+\; b\,y \;=\; c\,,$$

is provided in the form, (x_0, y_0), (α, β), where the set of solutions to the diophantine equation is

$$\{(x,y) \mid (x,y) = (x_0,y_0) + k(\alpha,\beta), k \in Z\}\,, \text{where } \alpha = -\frac{b}{\gcd(a,b)}, \beta = \frac{a}{\gcd(a,b)}\ .$$

So, the *gcd* of a and b is implicit in the solution set. Hence, using a strongly polynomial algorithm for solving a single diophantine equation in two variables, we can find $\gcd(a, b)$ (by solving $ax + by = 0$, say). The integer multipliers can then be found in strongly polynomial time by solving the diophantine equation $ax + by = \gcd(a, b)$. ■

It has recently been shown (see [MST91]) that no finite (fixed length) computation tree with operations $\{+,-,\times,/,\mathsf{mod},<\}$ can decide whether a and b are coprime, for arbitrary integers a and b; as a direct consequence of this result, there is no strongly polynomial algorithm for the *gcd* problem (find the *gcd* of two arbitrary integers, a and b) when restricted to this set of operations. An earlier paper (see [Sto76]) established a similar result when restricted to the set of operations $\{+,-,\times,/,<\}$ (i.e. no mod operation). The above result, [MST91], implies the following corollary to Theorem 1.1 (as previously mentioned, we will assume that our available set of operations is $\{+,-,\times,/,\mathsf{mod},<\}$):

Corollary 1.1 *There is no strongly polynomial algorithm for the problem of finding the set of solutions to a system of linear diophantine equations. In particular, no strongly polynomial algorithm exists for the problem of finding the set of integer solutions* $\{(x, y)\}$ *to the equation* $ax + by = c$, *where* $a, b, c \in Z$.

Proof: From Theorem 1.1, we see that a strongly polynomial to solve a system of linear diophantine equations (indeed, a strongly polynomial algorithm to solve a single diophantine equations in two variables) would imply a strongly polynomial algorithm for the *gcd* problem, which we know does not exist. ■

We show, however, that with an appropriately defined *gcd* oracle, we *can* solve a system of linear diophantine equations in strongly polynomial time. We will use the oracle to modify an existing *polynomial* time algorithm for solving a system of linear diophantine equations to provide a *strongly polynomial* algorithm. Suppose that we have an oracle that, given a and b, returns two things:

GCD-Oracle
1. $\gcd(a, b)$
2. Integers (x_0, y_0), bounded in length by a polynomial in the length of a and b, such that $ax_0 + by_0 = \gcd(a, b)$

We will refer to this oracle as the *GCD-Oracle*. [1] Note that, in essence, the *GCD-Oracle* returns the solution to a single diophantine equation in two variables. The next section of this paper shows that, using our standard set of operations along with a *GCD-Oracle*, we can find the set of solutions to a system of linear diophantine equations in strongly polynomial time. THis is done by providing a strongly polynomial time reduction to the problem solved by the *GCD-Oracle*.

Before proceeding to the next section, we mention that Euclid's algorithm can be used to return both the *gcd* and corresponding polynomial sized multipliers for a pair of integers, a and b, in $O(\log(\min\{|a|, |b|\}))$ steps (see [Sch87]). Note that this running time is polynomial, but not *strongly* polynomial. A more thorough treatment of *gcd*'s can be found in [Knu81].

[1] Observe that the oracle actually need only return the multipliers x_0, y_0, as the *gcd* of a and b can then be calculated immediately in strongly polynomial time.

2 A "Strongly Polynomial" Algorithm

Our presentation for solving a linear diophantine system follows fairly closely that given in [Sch87]. The purpose of this paper is not to redevelop the relevant theory of integer lattices; rather, our aim is to use the *GCD-Oracle* defined in the previous section to modify an existing *polynomial* algorithm that solves a system of linear diophantine equations in to a *strongly polynomial* algorithm. As such, we will freely use known results from integer lattice theory in our presentation (the reader is referred to [Sch87] for a more thorough development).

Our algorithm for solving a system of linear diophantine equations, $A\mathbf{x} = \mathbf{b}$, is based on finding the *Hermite Normal Form* of the matrix, A. The Hermite Normal Form of an integer matrix, A, of full row rank, is the matrix $[B\ 0]$, where B is lower triangular, nonsingular, nonnegative, and each row of B has its unique maximum entry located on the diagonal. B is unique and polynomially bounded in the size of A. $[B\ 0]$ is obtained from A by elementary column operations, where an elementary column operation is one of three types: exchange two columns; multiply a column by -1; add an integer multiple of one column to another column. In addition, there exists a unimodular matrix U, polynomially bounded in the size of A, such that $AU = [B\ 0]$.

Lemma 2.1 *Suppose that there exists an algorithm, HNF, that finds the Hermite Normal Form of an integer matrix of full row rank in strongly polynomial time. Then, we can solve the system of linear diophantine equations, $A\mathbf{x} = \mathbf{b}$, where A is a full row rank $m \times n$ integer matrix and $\mathbf{b}$ is an integer column vector, in strongly polynomial time. Such algorithm requires $O(m^2 n)$ operations from the set $\{+,-,\times,/,<\}$ and a single call to HNF with an $n \times n$ matrix.*

Proof: Let A be an $m \times n$ matrix (where $m \leq n$ because A is of full row rank). We will assume that the first m columns of A are linearly independent; if not, we can use Gaussian elimination to permute the columns of A, in strongly polynomial time ($O(m^2 n)$ operations, see [Edm67]), so that this is the case.

Let the Hermite Normal Form of A be $[B\ 0]$, where B is a lower triangular $m \times m$ matrix of rank m. If we let $A = \begin{bmatrix} A' & A'' \end{bmatrix}$, where A' is a nonsingular $m \times m$ matrix, then the Hermite Normal Form of the nonsingular $n \times n$ matrix

$$\begin{bmatrix} A' & A'' \\ 0 & I \end{bmatrix} \text{ can be written as } \begin{bmatrix} B & 0 \\ B' & B'' \end{bmatrix}.$$

The latter matrix can be found, using *HNF*, in strongly polynomial time (with respect to the size of A). Define

$$U = \begin{bmatrix} A' & A'' \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} B & 0 \\ B' & B'' \end{bmatrix}.$$

Because the sizes of $B, B', \text{ and } B''$ are polynomially bounded in the size of A, so too is the size of U. Noting that B and B'' are triangular, observe that U can be found in $O(m^2 n)$ steps by decomposing U into four components U_1, U_2, U_3, U_4 and solving the system:

$$\begin{bmatrix} A' & A'' \\ 0 & I \end{bmatrix} \begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \end{bmatrix} = \begin{bmatrix} B & 0 \\ B' & B'' \end{bmatrix}.$$

Now, $AU = \left[A' \; A''\right] U = [B \; 0]$. That is, U is a unimodular matrix such that $AU = HNF(A)$.

The general solution to the linear diophantine system is

$$\{\mathbf{x}_0 + \sum_{i=1}^{n-m} \lambda_i \mathbf{x}_i \mid \lambda_i \in Z, \forall i\} \; ,$$

where

$$[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-m}] = U \begin{bmatrix} B^{-1}\mathbf{b} & 0 \\ 0 & I \end{bmatrix} \; ,$$

which can be found in $O(mn)$ operations. ■

It is therefore sufficient, in order to solve a system of linear diophantine equations in strongly polynomial time, to be able to find the Hermite Normal Form of an integer matrix in strongly polynomial time. We now present an algorithm that, using a *GCD-Oracle*, computes the Hermite Normal Form of an integer matrix in strongly polynomial time. The algorithm follows closely that presented in [Sch87], which in turn is based on [DKT87].

The *GCD-Oracle* will be used in the following way: for given integers a, b, let

$$U_{a,b} = \begin{bmatrix} x_0 & -\frac{b}{\gcd(a,b)} \\ y_0 & \frac{a}{\gcd(a,b)} \end{bmatrix} \; ,$$

where x_0, y_0 are returned from the *GCD-Oracle*.

One can easily verify the following two facts about $U_{a,b}$:

Lemma 2.2 *For given integers a and b, let $U_{a,b}$ be as defined above. Then,*

1. $det(U_{a,b}) = 1$.
2. $[a \; , \; b] \, U_{a,b} = [\gcd(a, b) \; , \; 0]$.

The unimodular matrix, $U_{a,b}$, allows our algorithm for finding the Hermite Normal Form to combine $O(\log(\min\{a, b\}))$ column operations from the algorithm in [Sch87] into $O(1)$ column operations, by summarizing the $O(\log(\min\{a, b\}))$ column operations in a single matrix. This is the essence of converting the polynomial time Hermite Normal Form algorithm in [Sch87] into a strongly polynomial time algorithm.

Given a full row rank integer matrix, A, of dimension $m \times n$, algorithm *HNF* (see Figure 1) finds the Hermite Normal Form of A. We use $A_{i,j}$ to denote the matrix made up of the i^{th} and j^{th} columns of A. Matrices are named using upper case letters, and we use subscripted lower case letters to denote a particular entry in a matrix: for example, $a_{i,j}$ refers to the entry in the i^{th} row and j^{th} column of A.

Theorem 2.1 *Using the GCD-Oracle, algorithm HNF finds the Hermite Normal Form of a full row rank integer matrix, A, in strongly polynomial time. The algorithm performs $O(m^2 n)$ operations from the set $\{+, -, \times, /, \mathsf{mod}, <\}$, and makes $O(mn)$ calls to the GCD-Oracle.*

Step 1: Augment Matrix

Let Δ be the absolute value of the determinant of a nonsingular $m \times m$ submatrix of A.
Define $D = \Delta I$, where I is the m-dimensional identity matrix.
Let $\bar{A} = [A\ D]$. (The Hermite Normal Form of A is the same as the Hermite Normal Form of $\bar{A}$, with the last m columns removed.)
Add integer multiples of the last m columns of $\bar{A}$ to the first n columns of $\bar{A}$ so that all entries are no more than Δ in absolute value (i.e. reduce modulo Δ).

Step 2: Triangulate System

```
for i = 1 ..  m do
    for j = i+1 ..  n+i do
        {
```
(1) *Call the GCD-Oracle to find $U_{\bar{a}_{i,i},\bar{a}_{i,j}}$;*
(2) $\bar{A}_{i,j} \leftarrow \bar{A}_{i,j} U_{\bar{a}_{i,i},\bar{a}_{i,j}}$
(i.e. postmultiply columns i and j of $\bar{A}$ by $U_{\bar{a}_{i,i},\bar{a}_{i,j}}$, *to zero out $\bar{a}_{i,j}$);*
(3) *Add integer multiples of the last $m - i$ columns of $\bar{A}$ to columns i and j so that all entries of $\bar{A}$ remain bounded by Δ in magnitude (i.e. reduce modulo Δ) ;*
```
        }
```

Step 3: Bring into Hermite Normal Form

At the end of Step 2, $\bar{A}$ is of the form $[B\ 0]$, where B is lower triangular. We must make B nonnegative, with the maximum value for each row on the diagonal:

```
for i = 2 ..  m do
    for j = 1 ..  i-1 do
        Add an integer multiple of column i to column j
        so that b_{i,j} is nonnegative and less than b_{i,i} ;
```

Step 4: Remove extra columns

Remove the last m columns of $\bar{A}$.
Output the resulting matrix as the Hermite Normal Form of A.

Figure 1: Algorithm *HNF*

Proof: We first prove the correctness of the algorithm, followed by a proof that it runs in strongly polynomial time.

Each column of D can be expressed as an integer linear combination of the columns of A; it follows that the Hermite Normal Form of A is the same as that of $\bar{A}$ with the last m columns removed (see [Sch87]). So, we find, in Step 2, the Hermite Normal Form of $\bar{A}$. After the completion of line (2) in Step 2, we have zeroed out the (i, j) entry of $\bar{A}$ using the unimodular matrix returned by the *GCD-Oracle* (this follows from fact 2 of Lemma 2.2). It follows that the procedure in Step 2 maintains the invariant that, at the completion of an iteration of the outer loop, the first i rows of $\bar{A}$ are of the form $[B_i\ 0]$, where B_i is lower triangular. Hence, Step 3 is entered with a matrix of the form $[B\ 0]$, where B is lower triangular. After each completion of an iteration of the outer loop in Step 3, the first i rows of B contain nonnegative entries, with each row having it's largest entry on the diagonal. Observe that we have only performed elementary column operations on $\bar{A}$. This is clear in all but Step 2: in Step 2, we postmultiply columns of $\bar{A}$ by a unimodular (as previously observed: integer valued, with determinant 1) matrix returned by the *GCD-Oracle*, which is equivalent to performing a series of column operations on $\bar{A}$. Hence, at the end of Step 3, we are left with the Hermite Normal Form of $\bar{A}$. By our earlier remarks, removing the last m columns of the Hermite Normal Form of $\bar{A}$ gives us the Hermite Normal Form of A.

Having established the validity of the algorithm, we now prove that it has strongly polynomial running time. First we establish that the number of operations is a polynomial function of m and n: in Step 1, the value Δ can be found in $O(m^2n)$ operations using Gaussian elimination; in Step 2, we enter the inner loop $O(mn)$ times, each time performing $O(m)$ operations including a single call to the *GCD-Oracle* (i.e. we make a total of $O(mn)$ calls to the *GCD-Oracle*); Step 3 can be performed using $O(m^2n)$ operations. This establishes that the algorithm performs $O(m^2n)$ operations, including $O(mn)$ calls to the *GCD-Oracle*. Second, we establish that all operations are performed on polynomial sized numbers. In Step 1, we augment the original matrix with a matrix with largest entry Δ, which is a subdeterminant value of A and hence polynomial in the size of the entries of A; additionally, the size of values produced in the Gaussian elimination are polynomially bounded in the size of the input. Clearly, then, $\bar{A}$ enters Step 2 with elements polynomial in the size of the entries of A. In Step 2, the matrix returned by the *GCD-Oracle* is polynomial in the size of its inputs; since (3) ensures that our matrix entries are bounded in magnitude by Δ, the entries of $\bar{A}$ remain polynomially bounded in size. Finally, during Step 3, it can be seen by induction that the values of the entries in B remain bounded, in magnitude, by Δ^i; hence, the magnitudes of the entries in B never exceed Δ^m, which is polynomial in the size of the input. ■

We can now prove the main result of this paper.

Theorem 2.2 *Suppose that we are given a GCD-Oracle. Then, we can find the set of integer solutions to the system $A\mathbf{x} = \mathbf{b}$, where A and $\mathbf{b}$ have rational entries, in strongly polynomial time. The algorithm performs $O(n^3)$ operations from the set $\{+,-,\times,/,\mathsf{mod},<\}$ and makes $O(n^2)$ calls to the GCD-Oracle.*

Proof: First of all, we can perform Gaussian elimination on the system of equations to produce a system of full row rank in strongly polynomial time ($O(m^2n)$ operations). Secondly, by multiplying each equation by the product of the denominators in the equation (or, by multiplying each equation by the *least common multiple* of the denominators, which can be found in strongly polynomial time by dividing the product of the denominators by the *gcd* of the denominators), we can convert the system of equations to contain only integer valued entries using $O(mn)$ operations. So, without any loss in generality, we can assume that A is an integer matrix of full row rank and $\mathbf{b}$ is an integer column vector. Now, by Lemma 2.1 and Theorem 2.1, we have shown that with a *GCD-Oracle* we can determine the set of integer solutions to the system of equations, $A\mathbf{x} = \mathbf{b}$, in strongly polynomial time; since algorithm *HNF* is called with a $n \times n$ matrix (see Lemma 2.1), the running time of our algorithm is $O(n^3)$, including $O(n^2)$ calls to the *GCD-Oracle*. ■

The above theorem, along with Theorem 1.1, establishes that the problem of finding the set of solutions to a system of linear diophantine equations is strongly polynomial equivalent to the problem solved by the *GCD-Oracle*.

3 Additional Remarks

We make a few observations regarding the results in this paper:

1. In Lemma 2.1, we show how to find the unimodular multiplier matrix U, such that $AU = HNF(A)$: we augment the $m \times n$ matrix, A, and find the Hermite Normal Form of an $n \times n$ matrix (requiring $O(n^3)$ operations including $O(n^2)$ calls to the *GCD-Oracle*), from which U can then be calculated. Observe that U is a unimodular matrix that captures the column operations performed in transforming A to its Hermite Normal Form; as such, it is also possible to build up U during the execution of algorithm *HNF*, so that we only need to calculate the Hermite Normal Form of A (requiring only $O(m^2n)$ operations including $O(mn)$ calls to the *GCD-Oracle*), rather than the augmented $n \times n$ matrix described in the proof of Lemma 2.1.

2. We have shown that the problem of finding the set of solutions to a system of linear diophantine equations is strongly polynomial time equivalent to the problem of finding the *gcd* of two arbitrary integers, a and b, along with multipliers, x_0 and y_0, such that $ax_0 + by_0 = \gcd(a, b)$. We have assumed that our set of operations is $\{+,-,\times,/,\mathsf{mod},<\}$, so that the strongly polynomial equivalence of the two problems is with respect to this set of operations. Notice that the only place that we have made use of the mod operation is in algorithm *HNF*; hence, exhibiting an algorithm to find the Hermite Normal Form that does *not* make use of the mod operation would imply that the two problems are strongly polynomial reducible to each other with respect to the reduced set of operations, $\{+,-,\times,/,<\}$. Indeed, Theorem 1.1 and Lemma 2.1 remain valid with respect to this reduced set of operations.

3. Our oracle is required to find both the *gcd* of two integers and the multipliers (x_0, y_0). It would be nice to either omit the requirement that the oracle return the multipliers, or, show that we cannot solve a system of linear diophantine equations in strongly polynomial

time with an oracle that only returns the *gcd*. A necessary and sufficient condition for the latter would be to show that we cannot find the multipliers, with an oracle that returns only the *gcd*, in strongly polynomial time.

4. Corollary 1.1, using Theorem 1.1, asserts that no strongly polynomial algorithm exists for the problem of finding the *set* of solutions to a system of linear diophantine equations. We can actually make the stronger claim:

   Theorem 3.1 *No strongly polynomial algorithm exists for the problem of finding any (single) solution to a system of linear diophantine equations.*

   Proof: It is a well known result that, given two integers a and b, the equation

   $$ax + by = 1$$

   has an integer valued solution if, and only if, a and b are coprime. Hence, a strongly polynomial algorithm for finding a solution to a system of linear diophantine equations could be used to determine, in strongly polynomial time, whether or not two arbitrary integers are coprime, which we know (see [MST91]) is impossible. ■

   In light of Theorem 3.1, it is clear that no *strongly* polynomial algorithm exists for the problem of finding an integer solution to a system of linear inequalities, even if the number of variables is fixed (note that Theorem 3.1 holds even if the number of variables is fixed at 2). Lenstra has shown that, for a fixed number of variables, it *is* possible to find an integer solution (if one exists) to a set of linear inequalities in polynomial time (see [Len83]). Perhaps it is possible, however, to give a strongly polynomial algorithm for finding an integer solution to a set of linear inequalities, with fixed number of variables, with the help of something like a *GCD-Oracle*. (Note that a strongly polynomial (linear time) algorithm does exist for the continuous version of this problem (see [Meg84]).)

5. As we have seen, the fact that no strongly polynomial algorithm exists for coprimality can be used to establish that no strongly polynomial algorithm exists for other problems. We give another such example.

   Consider the following two-variable integer programming problem, which we will refer to as a 2-*dimensional knapsack* problem:

   $$\begin{array}{lll} \text{maximize} & c_1x_1 + c_2x_2 & \\ \text{s.t.} a_{i1} & x_1 + a_{i2}x_2 \leq b_i, & i = 1, 2, \ldots, n \\ & x_1, x_2 \geq 0, & \text{integers} \ , \end{array}$$

   where a_{ij}, c_j, and b_i are nonnegative integers. A polynomial algorithm for this problem when there is only a single constraint ($n = 1$) is given in [HW76]; a polynomial algorithm for general n is given in [Kan80].

   We show that *coprimality* is strongly polynomial reducible to an instance of 2-*dimensional knapsack* with a single constraint (see Appendix A). Hence, 2 *dimensional knapsack*, even

when there is a single constraint, cannot be solved in strongly polynomial time. (The algorithm in [Kan80] has strong resemblance to the Euclidean *gcd* algorithm; it would be interesting to see if the availability of something like a *gcd* oracle would allow for a strongly polynomial algorithm for *2-dimensional knapsack*.)

6. Theorem 1.1 shows that a strongly polynomial algorithm for finding the *set* of solutions to a system of linear diophantine equations implies a strongly polynomial algorithm for finding the *gcd* of two integers. Suppose, however, one is given an oracle for finding a *single* solution to a system of linear diophantine equations. Is it still possible to find, with this oracle, the *gcd* of two arbitrary integers in strongly polynomial time?

   Along these lines, one can ask many similar questions. For instance, is it possible to find the *gcd* of two arbitrary integers in strongly polynomial time with the use of an oracle that recognizes coprimality?

Appendix

A Lower Bound for 2-Dimensional Knapsack

We show that, as a consequence of the lower bound provided for the coprimality problem in [MST91], no strongly polynomial algorithm exists for the problem of finding the optimal solution to a 2-*dimensional knapsack problem*, in a complexity model allowing the operations $\{+,-,\times,/,\mathsf{mod},<\}$. We prove this result by providing a strongly polynomial reduction of the comprimality decision problem to the 2-*dimensional knapsack* optimization problem.

Let a and b be positive integers, and let k be a nonnegative integer.

Let *(P1)* and *(P2)* be defined as:

$$\begin{array}{lll} \textit{(P1)} & \max & ax+by \\ & \text{such that} & ax+by \leq 2^k \\ & & x,y \text{ integer} \end{array}$$

$$\begin{array}{lll} \textit{(P2)} & \max & ax+by \\ & \text{such that} & ax+by \leq 2^k \\ & & x,y \geq 0 \text{ integer} \end{array}$$

In the next three lemmas, it is assumed that $\gcd(a,b) \neq 2$ (i.e a and b are not both even).

Lemma A.1 *$ax+by=2^k$ has an integer solution if and only if a and b are coprime.*

Proof: Follows from the fact that $\gcd(a,b) \neq 2$, and that there is a solution if and only if $\gcd(a,b)$ divides into 2^k. ■

Lemma A.2 *(P2) has optimal objective value 2^k if and only if a and b are coprime.*

Proof: Follows immediately from Lemma A.1. ■

We next show that, for k large enough, we can ensure that there exists an optimal solution to *(P2)*.

Lemma A.3 *Let $k=\lceil \log(ab) \rceil$. Then, (P2) has optimal objective value 2^k if and only if a and b are coprime.*

Proof: From the previous lemma, we know that if a and b are not coprime then the optimal objective value of *(P2)* is less than 2^k.

Suppose that a and b are coprime. Then $ax+by=2^k$ has a set of solutions of the form,

$$\{(x,y)|(x,y)=(x_0-tb,y_0+ta),t \text{ integer}\} .$$

This describes the set of optimal solutions for *(P1)*.

Now, we can choose t so that we get a solution (x_1, y_1) with $0 \leq x_1 < b$. If we choose k so that $2^k \geq ab$, it follows that for $ax + by = 2^k$ to be satisfied, we must have $y_1 \geq 0$. Therefore, if we let $k = \lceil \log(ab) \rceil$, it follows that for a and b coprime, *(P2)* has an optimal objective value of 2^k.

Note also that $|2^k| = k \leq \lceil \log a \rceil + \lceil \log b \rceil$, and is therefore of polynomial size. ■

We can now establish the following theorem.

Theorem A.1 *No strongly polynomial algorithm exist for the* 2*-dimensional knapsack problem.*

Proof: We show that the *comprimality* decision problem can be reduced to an instance of 2*-dimensional knapsack* in strongly polynomial time. The theorem will then follow from the fact there is no strongly polynomial algorithm for determining whether two integers are coprime.

Suppose we wish to determine if two positive integers a and b are coprime. If a and b are both even, then we are done as the answer is *"no"*). Otherwise, consider *(P2)* with $k = \lceil \log(ab) \rceil$. From the previous lemma, we know that this problem has optimal solution 2^k if and only if a and b are coprime. ■

References

[DKT87] P.D. Domich, R. Kannan, and L.E. Trotter. Hermite Normal Form Computation using Modulo Determinant Arithmetic. *Mathematics of Operations Research*, 12:50–59, 1987.

[Edm67] J. Edmonds. Systems of Distinct Representatives and Linear Algebra. *Journal of Research of the National Bureau of Standards (B)*, 71:241–245, 1967.

[HW76] D. S. Hirschberg and C. K. Wong. A Polynomial Alogirhtm for the Knapsack Problem in Two Variables. *Journal of the Association for Computing Machinery*, 23(1):147–154, January 1976.

[Kan80] R. Kannan. A Polynomial Algorithm for the Two-Variable Integer Programming Problem. *Journal of the Association for Computing Machinery*, 27(1):118–122, January 1980.

[Knu81] D. E. Knuth. *The Art of Computer Programming*, volume 2 (Seminumerical Algorithms). Addison-Wesley, Inc., 2'nd edition, 1981.

[Len83] H. W. Lenstra, Jr. Integer Programming with a Fixed Number of Variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[Meg84] N. Megiddo. Linear Programming in Linear Time when the Dimension is Fixed. *Journal of the Association for Computing Machinery*, 31:114–127, 1984.

[MST91] Y. Mansour, B. Shieber, and P. Tiwari. A Lower Bound for Integer Greatest Common Divisor Computations. *Journal of the Association for Computing Machinery*, 38(2):453–471, April 1991.

[Sch87] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1987.

[Sto76] L. Stockmeyer. Arithmetic versus Boolean Operations in Idealized Register Machines. Technical Report RC 5954, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., April 1976.