Solving Integer Programs from Dependence and Synchronization Problems

Jaspal Subhlok

March 1993
CMU-CS-93-130

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a method to determine whether a set of equations has a non-negative integer solution. The method is designed for the particular occurence of this problem in the context of compiler analysis of parallel programs. The system of equations is first transformed to Smith normal form to determine if any integer solutions exist. In case of multiple integer solutions, a parameterized solution space representing all non-negative solutions is obtained. Fourier-Motzkin elimination is employed to determine if the real solution space is empty. If the solution space is not empty, either the existence of an integer solution is readily verified, or a simplified convex region is obtained such that the original system of equations has a solution if and only if this convex region contains an integer point. The main result of the paper is a set of new heuristic search procedures that are used to identify an integer solution in a convex region, or to prove that no integer solution exists. These are based on the geometrical properties of convex regions that are not empty but do not contain integer points. This method is an exact and efficient way of solving integer programming problems arising in dependence and synchronization analysis of parallel programs.

This research was sponsored by The Defense Advanced Research Projects Agency, Information Science and Technology Office, under the title "Research on Parallel Computing", ARPA Order No. 7330, issued by DARPA/CMO under Contract MDA972-90-C-0035.

Keywords: Exact dependence testing, integer programming, parallelizing compilers, parallel program analysis, synchronization analysis

Solving Integer Programs from Dependence and Synchronization Problems

CMU-CS-93-130

Jaspal Subhlok

March 1993

We present a method to determine whether a set of equations has a non-negative integer solution. The method is designed for the particular occurence of this problem in the context of compiler analysis of parallel programs. The system of equations is first transformed to Smith normal form to determine if any integer solutions exist. In case of multiple integer solutions, a parameterized solution space representing all non-negative solutions is obtained. Fourier-Motzkin elimination is employed to determine if the real solution space is empty. If the solution space is not empty, either the existence of an integer solution is readily verified, or a simplified convex region is obtained such that the original system of equations has a solution if and only if this convex region contains an integer point. The main result of the paper is a set of new heuristic search procedures that are used to identify an integer solution in a convex region, or to prove that no integer solution exists. These are based on the geometrical properties of convex regions that are not empty but do not contain integer points. This method is an exact and efficient way of solving integer programming problems arising in dependence and synchronization analysis of parallel programs.

Keywords: EXACT DEPENDENCE TESTING, INTEGER PROGRAMMING, PARALLELIZING COMPILERS, PARALLEL PROGRAM ANALYSIS, SYNCHRONIZATION ANALYSIS

(18 pages)

Contents

1 Introduction

Several mathematical problems that arise in the development of compilers for parallel languages can be transformed to integer programming problems. Determining whether a data dependence exists between two array references can be transformed to the problem of determining whether an integer solution to a set of equalities and inequalities exists [1, 10]. Similarly, in an event variable synchronization model, determining whether the synchronization present in a program is sufficient to protect a dependence can be transformed to the problem of determining whether a system of linear equations has a non-negative integer solution [2, 9]. Both these problems require integer programming which is known to be NP-complete. In this report, we present a practical solution technique for determining whether a non-negative integer solution for a system of equations exists. This is equivalent to the transformed synchronization analysis problem mentioned above, and the solution methodology can also be used to solve the data dependence analysis problem. Both these problems have characteristics that make it possible to develop methods that work well in practice, even though the problems are NP-complete. We state the characteristics of systems of equations that are constructed from the synchronization analysis problem:

- The number of variables corresponds to the number of synchronization operations relevant to a specific data dependence, and is expected to be small.
- The number of equations corresponds to the number of subscript positions in the array references causing a data dependence, and is expected to be small.
- The coefficients of terms are synchronization distance vector components and are usually small integers, very often one or zero.

Systems of equations constructed from the dependence analysis proble have similiar characteristics. The point is that these equation systems are fairly small and can be solved efficiently in practice.

We outline the main steps in the solution method. We first solve the equation system for all (not necessarily non-negative) integer solutions. This is often referred to as solving linear diophantine equations. If such a solution is unique or does not exist, we determine this fact and do not need to proceed any further. If there are multiple solutions, we obtain parametric equations that describe the solution space. Next we determine the non-negative solution space and check whether it is empty. If it is not empty, we use heuristic search procedures to determine if it contains an integer solution.

2 Solving Systems of Linear Diophantine Equations

The theory of solving linear diophantine equations has been discussed in many texts addressing number theory and integer programming [5, 4]. Our contribution is to present a solution method in a way that can be easily programmed on a computer and analyze the complexity. We also discuss why we chose the specific method that we present.

The problem can be stated as follows:

Find all integer solutions of

$$[A]_{m\times n}\cdot[x]_{n\times 1} = [b]_{m\times 1}$$

We first present the necessary mathematical concepts, and then an algorithm to solve the problem.

2.1 Mathematical Concepts

Following are the *elementary* column operations on a matrix:

1. exchanging two columns
2. multiplying a column by -1
3. adding an integral multiple of one column to another column

Corresponding elementary row operations are similarly defined. We shall use elementary operations to transform a matrix to a form desirable for solving the equations. In the rest of this section, we introduce two kinds of matrices that describe specific elementary operation sequences that we shall use.

A *transposition* matrix $[P]_{n\times n}$ is a square matrix of 1s and 0s in which:

1. each row and each column has exactly one 1.

2. all the 1's except two are along the main diagonal.

The matrix shown below, P_{25} is a transposition matrix of order 5. The subscript 25 signifies that the 1s in rows(and columns) 2 and 5 are out of place relative to a unit matrix.

$$P_{25} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Theorem 1 *Premultiplying a matrix A with a transposition matrix P_{ij} is equivalent to interchanging rows i and j of A, and postmultiplying A with P_{ij} is equivalent to interchanging columns i and j of A.*

We show this with an example where premultiplying A with P_{25} yields A with rows 2 and 5 permuted.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{51} & a_{52} & a_{53} & a_{54} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}$$

A *Subtraction* matrix $[S_{ij,\alpha}]_{n \times n}$ is a square matrix in which:

1. all items along the main diagonal are 1.

2. $S[i, j] = -\alpha$.

3. all other elements are 0.

Following is an order 5 subtraction matrix $S_{13,\alpha}$

$$S_{13,\alpha} = \begin{bmatrix} 1 & 0 & -\alpha & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Theorem 2 *Premultiplying a matrix A with a subtraction matrix $S_{ij,\alpha}$ has the effect of reducing the ith row of A by α times the jth row of A. Postmultiplying a matrix A with a subtraction matrix $S_{ij,\alpha}$ has the effect of reducing the jth column of A by α times the ith column of A.*

We show this with an example where postmultiplying A with $S_{13,\alpha}$ yields A with every element of column 1 reduced by α times the corresponding row element in column 3.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \end{bmatrix} \begin{bmatrix} 1 & 0 & -\alpha & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} - \alpha a_{13} & a_{12} & a_{13} & a_{14} \\ a_{21} - \alpha a_{23} & a_{22} & a_{23} & a_{24} \\ a_{31} - \alpha a_{33} & a_{32} & a_{33} & a_{34} \\ a_{41} - \alpha a_{33} & a_{42} & a_{43} & a_{44} \\ a_{51} - \alpha a_{43} & a_{52} & a_{53} & a_{54} \end{bmatrix}$$

Multiplication of two subtraction matrices, say S_{ij,α_1} and S_{ik,α_2} yields a matrix with 1s along the diagonal, α_1 and α_2 at locations S_{ij} and S_{ik} respectively, and 0s elsewhere. The operation of subtracting different multiples of a column of a matrix from other columns, which can be accomplished by successive multiplication by different subtraction matrices, can be achieved by a multiplication with a *composite subtraction matrix* constructed as described above.

Transposition matrices and subtraction matrices are examples of a more general class of matrices called *regular unimodular matrices*. A regular unimodular matrix is a square matrix whose determinant has the value $(+1)$ or (-1). We state some of the properties of regular unimodular matrices that are relevant for our purposes:

- the product of two regular unimodular matrices is a regular unimodular matrix.
- the inverse of a regular unimodular matrix formed of all integers always exists and is a regular unimodular matrix.

2.2 Smith Normal Form

In this section we define a matrix form called the *Smith normal form* [8] and how a matrix can be transformed to Smith normal form using operations defined by transposition and subtraction matrices. The theory of transforming matrices to Smith normal form and solving systems of equations in integers using it can be found in [5, 4]. Here we present an algorithmic description of the method and argue that it is a suitable first step for solving the systems of equations we expect to encounter.

The Smith normal form of a matrix $[A]_{m\times n}$ of rank r is a matrix D of the following form:

$$[D]_{m\times n} = \begin{bmatrix} d_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & d_r & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}$$

where $d_k \neq 0$, $k = 1, 2, \cdots r$.

Theorem 3 *For every matrix $[A]_{m\times n}$, there exist regular unimodular matrices $[U]_{m\times m}$ and $[V]_{n\times n}$ such that*

$$[U]_{m\times m}.[A]_{m\times n}.[V]_{n\times n} = [D]_{m\times n}$$

where $[D]_{m\times n}$ is in Smith normal form

We present a procedure to compute the Smith normal form of a matrix and the corresponding unimodular multiplier matrices.

algorithm

Input: Matrix $[A]_{m\times n}$

Output: Matrices $[D]_{m\times n}$, $[U]_{m\times m}$ and $[V]_{n\times n}$ such that

$$[U]_{m\times m}.[A]_{m\times n}.[V]_{n\times n} = [D]_{m\times n}$$

where U and V are regular unimodular matrices and D is in Smith normal form.

Initialization: $D = A$, U and V are unit square matrices of dimensions m and n respectively.

Method: We use the procedure FindSmith stated in Figure 1, which converts a matrix of the form D_r to D_{r+1} shown below in each step.

$$D_r = \begin{bmatrix} d_1 & 0 & \cdots & 0 & \\ 0 & d_2 & \cdots & 0 & \\ \vdots & \vdots & \ddots & \vdots & [0]_{r\times n-r} \\ 0 & 0 & \cdots & d_r & \\ [0]_{m-r\times r} & & & & [D'_r]_{m-r\times n-r} \end{bmatrix}$$

$$D_{r+1} = \begin{bmatrix} d_1 & 0 & \cdots & 0 & 0 & \\ 0 & d_2 & \cdots & 0 & 0 & \\ \vdots & \vdots & \ddots & \vdots & \vdots & [0]_{r+1\times n-r-1} \\ 0 & 0 & \cdots & d_r & 0 & \\ 0 & 0 & \cdots & 0 & d_{r+1} & \\ [0]_{m-r-1\times r+1} & & & & & [D'_{r+1}]_{m-r-1\times n-r-1} \end{bmatrix}$$

The procedure stops when D is transformed to a Smith normal form.

procedure FindSmith(D)

$r = 1$
U and V are unit matrices of rank m and n respectively.
$size(D)$ is the minimum of numbers of rows and number of columns in D.
We refer to the submatrix of D formed of elements with row and column number greater than r as D'_r.

1. If $r > size(D)$ or the matrix D'_r has all 0s *STOP*.
   /* D is already in Smith normal form */

2. Determine
   minrow, *mincol* and *minval* - the location and value of the smallest nonzero number in D'_r.

3. Interchange row r with *minrow* and column r with *mincol*.
   /* Now location D_{rr} has value *minval*. */
   Premultiply V and postmultiply U with permutation matrices that reflect this transformation.
   $V = P_{minrow,r}.V$ and $U = U.P_{mincol,r}$

4. Reduce row and column r with appropriate subtraction matrices:
   For $i = r+1, n$:
   $\alpha_i = D_{ri}$ div *minval*
   $D_{ri} = D_{ri}$ mod *minval*
   The relevant composite subtraction
   matrix S is an $n \times n$ unit matrix except that $\alpha_{r+1}, \alpha_{r+2}, \cdots \alpha_n$ are the entries in row r, columns $r+1$ through m, respectively. (see earlier discussion).
   $V = S.V$
   Repeat the subtraction procedure for column r and postmultiply U accordingly.

5. If row and column r of D contain all 0s, $r = r + 1$ *GOTO* step 2. Else *GOTO* step 1.

end of procedure

Figure 1: Conversion of a Matrix to Smith normal form

Complexity: We analyze the complexity of procedure FindSmith shown in Figure 1. Steps 1 and 2 search a submatrix of $[D]_{m \times n}$ in $O(mn)$ time. Step 3 interchanges one pair of rows and one pair of columns and multiplies corresponding permutation matrices to unimodular matrices U and V, which is again equivalent to a row pair and a column pair interchange. The time taken is $O(m+n)$. Step 4 performs addition of a multiple of a column to the remaining columns in a submatrix of D and a similar operation with rows of D. A similar operation is performed on unimodular matrices U and V, when they are multiplied by the subtraction matrix computed above. All of these operations take $O(mn)$ time. Thus the overall complexity of one execution of steps 1-4 is $O(mn)$.

In step 5, the control goes back to step 2 if row and column r do not have all 0s, except in location D_{rr}. If D_{rr} became 1, the above condition must be met in the next step. Also D_{rr} is reduced in every step, in a manner similar to computing *gcd* of a set of numbers and is expected to reach 1 in $log_2(value)$ steps, where *value* is the smallest element in the submatrix D'_r at the beginning of this reduction step. Finally, the loop from 1-5 is executed R times, where R is the rank of the original matrix A.

Thus the complexity of computing the Smith normal form and the corresponding unimodular matrices for matrix $[A]_{m \times n}$ of rank R is $O(mnR\log(value))$ where *value* can be approximated by the median of non-zero values in matrix A. Thus if *size* is the higher dimension of A, the complexity of the operation is bounded by $O((size)^3 \log(value))$.

end of algorithm

Systems of equations in integers can also be solved by transforming a matrix to a triangular matrix form in which only row or column operations are used in the transformation procedure. However, in those methods a new lowest element in pivot position($[D]_{rr}$ in earlier discussion) can be chosen only from a particular row or column. In computing Smith normal form, the new pivot can be chosen from anywhere in the unprocessed section of the matrix, hence there is a greater chance of finding a pivot that would zero out the corresponding row and column.

2.3 Solving Equation Systems

We show how transformation to Smith normal form is used to solve a system of equations in integers.

Let

$$[A]_{m\times n}.[x]_{n\times 1} = [b]_{m\times 1} \tag{1}$$

be a system of linear equations for which we need to determine the integer solution set.

Let

$$[D]_{m\times n} = [U]_{m\times m}.[A]_{m\times n}.[V]_{m\times n} \tag{2}$$

where D is in Smith normal form and U and V are unimodular matrices.

We rewrite from equation 1

$$[U]_{m\times m}.[A]_{m\times n}.[V]_{n\times n}.[V^{-1}]_{n\times n}.[x]_{n\times 1} = [U]_{m\times m}.[b]_{m\times 1} \tag{3}$$

$$[D]_{m\times n}.[V]^{-1}_{n\times n}.[x]_{n\times 1} = [U]_{m\times m}[b]_{m\times 1} \tag{4}$$

We define

$$[t]_{n\times 1} = ([V^{-1}].[x])_{n\times 1} \tag{5}$$

From equation 4 we get

$$[D]_{m\times n}.[t]_{n\times 1} = [U]_{m\times m}[b]_{m\times 1} \tag{6}$$

If r is the rank of matrix A, and hence of matrix D, then $[D_{m\times n}]$ is of the form $\begin{bmatrix} D_{r\times r} & 0 \\ 0 & 0 \end{bmatrix}$. We rewrite equation 6 as two equations representing the top r rows and bottom $m-r$ rows of the column matrices being equated.

$$[D^r_r]_{r\times r}.[t^r_r]_{r\times 1} = [([U].[b])^r_r]_{r\times 1} \tag{7}$$

$$[0]_{(m-r)\times 1} = [([U].[b])^{m-r}_{m-r}]_{(m-r)\times 1} \tag{8}$$

Thus, for the original system of equations to have an integer solution

$$[t^r_r]_{r\times 1} = [D^r_r]^{-1}_{r\times r}[([U].[b])^r_r]_{r\times 1} \tag{9}$$

must be formed of integers and

$$[([U].[b])^{m-r}_{m-r}]_{(m-r)\times 1} = [0]_{(m-r)\times 1} \tag{10}$$

If an integer solution exists, it can be represented by rewriting from equation 4.

$$[x]_{n\times 1} = [V]_{n\times n}.[t]_{n\times 1} \tag{11}$$

We demonstrate the complete procedure of solving systems of equations in integers with an example. Consider the following system of equations:

$$\begin{array}{rcrcrcrcr} 2x_1 & + & 3x_2 & + & 4x_3 & + & 5x_4 & = & 3 \\ x_1 & - & x_2 & + & x_3 & + & 2x_4 & = & 5 \\ 2x_1 & + & 0x_2 & + & 2x_3 & + & 5x_4 & = & -3 \end{array}$$

We restate the system as:

$$\begin{bmatrix} 2 & 3 & 4 & 5 \\ 1 & -1 & 1 & 2 \\ 2 & 0 & 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ -3 \end{bmatrix}$$

We have

$$A = D^0 = \begin{bmatrix} 2 & 3 & 4 & 5 \\ 1 & -1 & 1 & 2 \\ 2 & 0 & 2 & 5 \end{bmatrix} U^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We shall transform these matrices such that

$$D^k = U^k.A.V^k$$

remains an invariant.

First we interchange rows 1 and 2 of D to obtain the smallest non-zero element of the matrix in the top left corner. In the above equation we premultiply D and U by permutation matrix P_{12}. We have

$$D^1 = \begin{bmatrix} 1 & -1 & 1 & 2 \\ 2 & 3 & 4 & 5 \\ 2 & 0 & 2 & 5 \end{bmatrix} U^1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^1 = V^0$$

To get the smallest possible values in 1st column of D, we premultiply by composite subtraction matrix $S_{21,2}.S_{31,2}$. For the same effect on first row, we postmultiply by $S_{12,-1}.S_{13,1}.S_{14,2}$ We have

$$S^1_{PRE} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} S^1_{POST} = \begin{bmatrix} 1 & 1 & -1 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After corresponding multiplications we get

$$D^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 5 & 2 & 1 \\ 0 & 2 & 0 & 1 \end{bmatrix} U^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -2 & 0 \\ 0 & -2 & 1 \end{bmatrix} V^2 = \begin{bmatrix} 1 & 1 & -1 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Again to move the lowest nonzero element of D to D_{22} we interchange columns 2 and 4 of D. We postmultiply D by permutation matrix P_{24}. We get

$$D^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 5 \\ 0 & 1 & 0 & 2 \end{bmatrix} U^3 = U^2 \quad V^3 = \begin{bmatrix} 1 & -2 & -1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

To reduce the row and column 2 of matrix D, we premultiply by subtraction matrix $S_{32,1}$ and postmultiply by composite subtraction matrix $S_{23,2}.S_{24,5}$ and obtain

$$D^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -3 \end{bmatrix} U^4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -2 & 0 \\ -1 & 0 & 1 \end{bmatrix} V^4 = \begin{bmatrix} 1 & -2 & 3 & 11 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & -2 & -5 \end{bmatrix}$$

Now the lowest nonzero element in the part of D not in Smith normal form is already in the desired position. To reduce the remaining one element, we postmultiply by subtraction matrix $S_{34,1}$ and get:

$$D^5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -1 \end{bmatrix} U^5 = U^4 \quad V^5 = \begin{bmatrix} 1 & -2 & 3 & 8 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & -2 & -3 \end{bmatrix}$$

We interchange columns 3 and 4 to get the smallest element in the desired position by postmultiplying by permutation matrix P_{34} and get:

$$P^5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -2 \end{bmatrix} \quad U^6 = U^5 \quad V^6 = \begin{bmatrix} 1 & -2 & 8 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & -3 & -2 \end{bmatrix}$$

Postmultiplying by subtraction matrix $S_{34,2}$ brings D in Smith normal form and we get the final forms:

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad U = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -2 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad V = \begin{bmatrix} 1 & -2 & 8 & -13 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & -3 & 4 \end{bmatrix}$$

And we verify that

$$D = U.A.V$$

The top left square submatrix of D with number of rows equal to rank r of D is the following matrix:

$$[D_{rr}]_{3\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Its inverse can be computed easily since it is a diagonal matrix.

$$[D_{rr}]^{-1}_{3\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

From equation 9 we have

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & -2 & 0 \\ -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ -3 \end{bmatrix} = \begin{bmatrix} 5 \\ -7 \\ 6 \end{bmatrix}$$

Since it evaluates to an integer column and equation 10 is trivially satisfied, the system must have integer solutions. The solutions of the system can be represented using equation 11

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 8 & -13 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & -3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ -7 \\ 6 \\ t_4 \end{bmatrix} = \begin{bmatrix} 67 - 13t_4 \\ 6 - 2t_4 \\ -6 + 3t_4 \\ -25 + 4t_4 \end{bmatrix}$$

We rearrange the solution and replace t_4 by t since there is only one parameter.

$$\begin{aligned} x_1 &= -13t + 67 \\ x_2 &= -2t + 6 \\ x_3 &= 3t - 6 \\ x_4 &= 4t - 25 \end{aligned}$$

2.4 Comparison with Generalized GCD test

Generalized GCD Test is a well known way of transforming a system of equality constraints to a system of inequality constraints. Both methods essentially use matrix elimination procedures in the domain of integer solutions. In our method, any matrix element can be chosen as pivot, and one row/column is eliminated when the pivot element is a factor of every element in the corresponding row and column (trivial if the absolute value of pivot is 1). In generalized GCD Test, only elementary row operations are used, hence the choice of a pivot is restricted to a single column. Thus, although we spend more time looking for a pivot element, we expect perform less steps to eliminate a row/column.

3 Finding Solutions in a Feasible Region

We first summarize the results of the solution procedure of the last section, in the context of the overall problem. We started with a system of m equations (and say r independent equations) in n variables. If r is greater than or equal to n, any integer solution that exists must be unique, and we would have already determined whether a non-negative integer solution exists. However, if r is less than n, then we have all possible solutions for the n variables parameterized by $n - r$ new variables which can take any integer value. We need to determine whether there are any non-negative integer solutions to the original problem. For the example in the last section, the problem is parameterized in terms of the new variable t, and the original problem has a non-negative integer solution if and only if the system:

$$\begin{aligned} -13t &+ 67 \geq 0 \\ -2t &+ 6 \geq 0 \\ 3t &- 6 \geq 0 \\ 4t &- 25 \geq 0 \end{aligned}$$

is feasible for some integer value of t.

This inequality system is particularly easy to solve since there is only one variable involved. The above system simplifies to:

$$t \leq 67/13 \quad (1)$$
$$t \leq 3 \quad (2)$$
$$t \geq 2 \quad (3)$$
$$t \geq 25/4 \quad (4)$$

The system is infeasible since (2) and (4) cannot be simultaneously satisfied.

3.1 Fourier-Motzkin Elimination Method

The number of variables in a system of inequalities can be successively reduced by using Fourier-Motzkin elimination method. Any system of inequalities can then be solved by successive reduction and back substitution. In this method, a variable is eliminated by creating a new inequality for each pair of inequalities in which the variable being eliminated has a different sign. This is an important step in the solution procedure, but our approach is as same as that taken in several other systems. However, for completeness, wee will illustrate this method with an example. A complete treatment can be found in [3]. Consider the following set of inequalities:

$$0x - y + 6 \geq 0 \quad (1)$$
$$x + 2y - 6 \geq 0 \quad (2)$$
$$-2x - 3y + 7 \geq 0 \quad (3)$$
$$-4x - 5y + 40 \geq 0 \quad (4)$$

Suppose we decide to eliminate x from this system of inequalities[1]. x has a positive sign in (2) above and a negative sign in (3) and (4) above. By combining inequality pairs (2)-(3) and (2)-(4) to eliminate x, we obtain the following system:

$$-y + 6 \geq 0 \quad (1)$$
$$y - 5 \geq 0 \quad from\,(2)\,and\,(3) \quad (2)$$
$$3y + 16 \geq 0 \quad from\,(2)\,and\,(4) \quad (3)$$

This simplifies to:

$$5 \leq y \leq 6$$

[1] Heuristics for choosing a variable so as to minimize the work are discussed in [3].

Thus a solution set to the inequalities does exist. We substitute the maximum or minimum possible value of y back in the original equations so as to obtain the range of values x can take. In the form in which our equations are, we substitute such that the y term has the minimum possible value. We get :

$$\begin{aligned} x+4 &\geq 0 \quad (1)\\ -2x-11 &\geq 0 \quad (2)\\ -4x+10 &\geq 0 \quad (3) \end{aligned}$$

These are equivalent to the following inequality:

$$-4 \leq x \leq -5/2$$

Solving systems of inequalities by elimination tells us whether a real solution exists to the system of inequalities. In case such a solution does exist, we can determine the range of the values that the variables may have in the solution space. It still needs to be determined whether an integer solution to the system exists.

Solving a system of inequalities by Fourier-Motzkin elimination techniques can potentially increase the number of inequalities from n to $(n/2)^2$ in every stage. Duffin [3] discusses ways to lower the computation effort needed to reach a solution. They include rules for selection of the variable to be eliminated at each stage, and identification and deletion of redundant inequalities. He further argues that with these modifications, Fourier analysis is a practical method to solve systems of inequalities. The number of variables and inequalitites that we expect in our problems is small and we expect the analysis to be fast and take only a small number of operations.

4 Identifying Integer Solutions

The system of inequalities obtained by solving diophantine equations defines a convex region in n dimensional space where n is the number of variables in the parametric equations. We are interested in determining the feasibility of an integer solution, if the Fourier-Motzkin analysis shows that the solution space is not empty. In Fourier-Motzkin elimination, we also determine the bounds in each dimension for the convex region defined. Furthermore these bounds are *tight* in the sense that at least one vertex of the convex region must be on each bounding hyperplane.

We define some terms here that we shall use in the rest of this section. An $n-rectangle$ is a closed convex region in n dimensions bounded by hyperplanes that are parallel to the axis hyperplanes (that is, defined by $X = constant$ for some co-ordinate axis X). An n-rectangle *tightly*, or *strictly* bounds a convex region if the the convex region is completely contained in the n-rectangle, and every hyperplane defining the n-rectangle intersects with the convex region. The *centroid* of an n-rectangle is the point with co-ordinates equal to the midpoint of the range of the n-rectangle along each axis. If $X_1, X_2, ...X_n$ are the co-ordinate axes and $(\alpha_1, \alpha_2, ...\alpha_n)$ and $(\beta_1, \beta_2, ...\beta_n)$ are two points in n space, a *line* in n dimensions between them is defined by:

$$\frac{x_1-\alpha_1}{\beta_1-\alpha_1} = \frac{x_2-\alpha_2}{\beta_2-\alpha_2} = \cdots \frac{x_n-\alpha_n}{\beta_n-\alpha_n}$$

4.1 The Case of Two Dimensions

If the system of inequalities obtained has only one parametric variable, the system can be trivially solved as shown earlier. We first describe a solution procedure when there are two parametric variables. Subsequently we will describe the solution method for three or more variables by solving a set of two variable problems. We expect that most practical situations would yield a system in two or fewer variables.

A system of inequalities in two dimensions describes a convex region in two dimensions. For the present we assume that the region is finite. Fourier-Motzkin elimination discussed in the last section would yield a tight bounding rectangle for the convex region. The bounding rectangle is known to contain integer points and we want to determine if the convex region itself has any integer points.

The following theorem holds for convex regions in two dimensions:

Theorem 4 *If a finite convex region in two dimensions is tightly bounded by a rectangle, then the centroid of the rectangle is always a part of the convex region.*

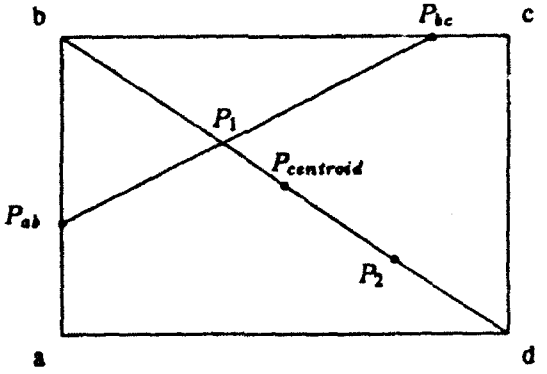


Figure 2: Illustration for Theorem 4

Proof:

Let C be a convex region in two dimensions and let R be the rectangle that bounds it tightly. Let $abcd$ be the vertices of the rectangle R as shown in Figure 2. Let $P_{centroid}$ be the centroid of R, which is also the point of intersection of the diagonals ac and bd. Since R bounds C tightly, each of the sides ab, bc, cd and da must contain at least one point that belongs to C. Let P_{ab} and P_{bc} be points on the sides ab and bc respectively, that belong to C. Since C is convex, the line segment $P_{ab}P_{bc}$ is contained in C. Also, the line segment $P_{ab}P_{bc}$ is inside the triangle abc and must intersect the diagonal bd at a point between b and $P_{centroid}$ (which is on the diagonal ac), say P_1. Thus there is one point on the diagonal bd between b and $P_{centroid}$ that is also in C. Similarly we can show that there is one point on the diagonal bd between d and $P_{centroid}$, say P_2, that is also in C. Therefore, since C is convex, $P_{centroid}$ is part of C.
End of Proof.

This theorem suggests that the neighborhood of the centroid is a good place to look for possible integer solutions. We use this fact in the algorithm presented later in this section. Furthermore, if none of the closest integer point neighbors of the centroid are in the convex region, the convex region can only be of a special shape. Specifically we have the following results:

Lemma 5 *Let R (vertices $r_1r_2r_3r_4$) and R' (vertices $r'_1r'_2r'_3r'_4$) be rectangles with parallel sides such that R' is enclosed inside R. Let p be a point inside R'. A set of line segments that connect a point on each side of R to p must intersect at least two sides of R'.*

Proof:

We will use Figure 3 for illustration. Any line segment connecting p to a point on a side of R must intersect at least one side of the rectangle R' since all of R is entirely outside R' and p is inside R'. Suppose $r'_4r'_3$ is the only side of R' that the set of line segments connecting p to each side of R intersect. Now p is a point between parallel lines passing through r_1r_2 and $r'_4r'_3$ and a line segment connecting a point on r_1r_2 to p cannot intersect the line through $r'_4r'_3$. Thus $r'_4r'_3$ cannot be the only side of R' that the set of line segments connecting p to each side of R intersect. Hence, there must be at least two such sides, and that proves the lemma.

End of Proof.

Theorem 6 *Let C be a convex region in two dimensions. Let R be the smallest rectangle with sides parallel to the axes that encloses C. Assume that the centroid of R is not an integer point and let R' be the smallest rectangle(square) with*

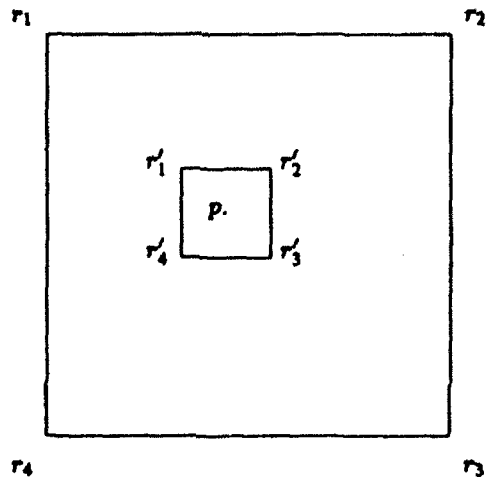


Figure 3: Illustration for Lemma 5

integer vertices that includes the centroid of R. If none of vertices of R' belong to C, and R is more than 4 integer units in length and width, then C intersects exactly two sides of R'.

Proof:

Let R be the rectangle $r_1r_2r_3r_4$ in Figure 4(i) that tightly bounds a convex region C (not shown in the figure). R' is the rectangle shown as $r'_1r'_2r'_3r'_4$ and the centroid of R is an interior point of R'. The sides of R' are extrapolated until they meet a side of R.

Since R tighly bounds C, there is at least one point belonging to C on each side of rectangle R. Also there is at least one point belonging to C inside R' (centroid of R). Since C is convex, there is a line segment from a point on each side of R to a point inside R' which is completely contained in R. From Lemma 5, C must intersect at least two sides of R'.

We now prove that C can intersect at most two sides of R'. Suppose C intersects three sides of R'. Without loss of generality we assume the sides are $r'_2r'_3$, $r'_3r'_4$ and $r'_4r'_1$. We represent this by drawing these sides with thick lines in Figure 4(ii). In Figure 4 we will convert every line segment that we prove has a point belonging to C, to thick lines. Every line segment which is proved *not* to contain any points in C is marked by dashed lines.

We have assumed that there is at least one point on the line segment $r'_4r'_1$ that belongs to C. If any point on the line segment r'_1a belongs to C, then the point r'_1 must belong to C, since C is convex. But r'_1 is a vertex of R' and does not belong to C. Thus we conclude *that the line segment r'_1a does not intersect C.* Similarly we can show that line segments r'_1a, r'_4f, r'_2b, r'_3e, r'_4g and r'_3d cannot intersect C. These inferences are shown in Figure 4(iii).

If there is any point of C inside the rectangle gr'_4fr_4, then a line segment from that point to a point inside R' must be contained in C. But such a line segment must intersect gr'_4 or r'_4f, and we have established that these line segments do not intersect C. Thus there can be no points inside the rectangle gr'_4fr_4 that belong to C. In particular, line segments fr_4 and r_4g do not intersect C. Similarly we can show that line segments dr_3 and r_3e do not contain any points belonging to C. This is shown in Figure 4(iv).

Since R tightly bounds C, there must be at least one point on r_3r_4 that is in C. Since we have proved that line segments r_3e and fr_4 do not contain any points belonging to C, the line segment ef must intersect C, as shown in Figure 4(v).

Now since the sides of R are more than 4 integer units (or 4 times the sides of R'), line segments gr'_4, hr'_1, r'_4f, r'_4e are all longer than one integer unit (or longer than sides of R'). Also sides of R' and line segments ef and gh are of unit length.

It can be seen that any line through a point on the line segment ef that does not intersect the line segment fr'_4 cannot intersect the line through points h and c at a point more than 1 unit left (towards h) of point r'_1, and hence cannot intersect the line segment gh (see pq in Figure 4(v)). Thus there cannot be points belonging to C on both line segments ef and

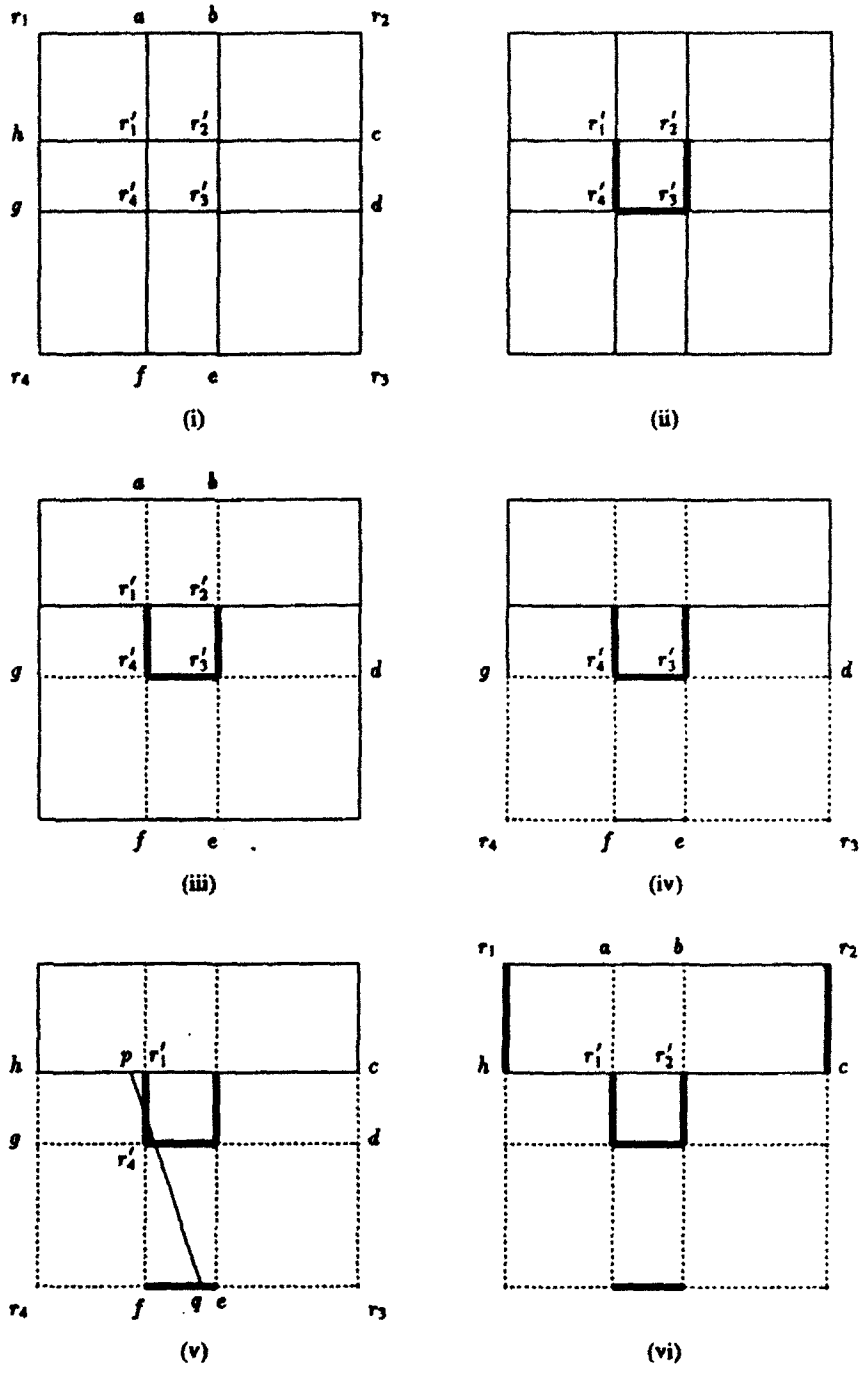


Figure 4: Intersection of a convex region and its enclosing rectangle

gh, since a line segment connecting those points would have to intersect the line segment fr'_4 and we have proved that the line segment fr'_4 does not intersect C. Since we have also proved that the line segment ef has at least one point belonging to C, the line segment gh cannot intersect C. Similarly we can show that the line segment cd cannot intersect C. Thus we reach the situation shown in Figure 4(v).

Now since the side r_4r_1 of rectangle R must have at least one point that is in C and the line segment r_4h has been shown to not have any points belonging to C, the line segment r_1h must intersect C. Similarly we can show that the line

segment r_2c must intersect C (see Figure 4(vi)). Thus there must be a line segment from a point on the line segment r_1h to a point on the line segment r_2c which is contained in C. But every such line segment must intersect the line segment ar_1' (and br_2') which we have proved does not intersect C. Therefore we have a contradiction.

Hence the convex region C cannot intersect three or more sides of R'. Since we have already shown that it must intersect at least two sides of R', the convex region C must intersect exactly two sides of R'.

End of Proof.

Furthermore, we find that a convex region that satisfies the requirements of this theorem must be located around one of the diagonals of the bounding rectangle. Specifically we have the following theorem:

Theorem 7 *Let C, R and R' be defined as in Theorem 6. Let R_1, R_2, R_3 and R_4 be the rectangles formed by the extensions of the sides of R' and the rectangle R as shown in Figure 5. Then under the conditions of Theorem 6, exactly two diagonally opposite rectangles R_i intersect with C.*

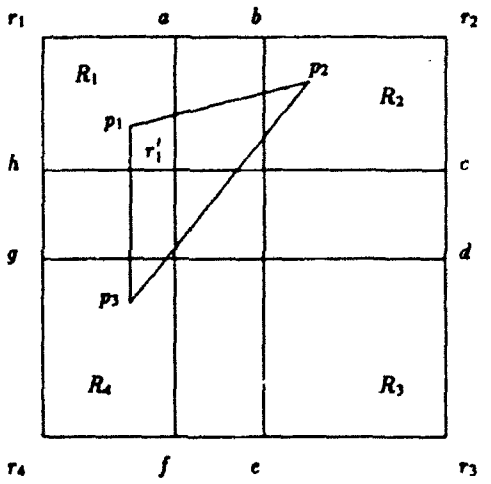


Figure 5: Illustration for Theorem 7

Proof:

We first show that at least two of the rectangles R_i intersect C. Suppose none of the rectangles R_i intersect C. Since R strictly bounds C, all sides of R intersect C. Thus C must intersect each of the segments ab, cd, ef and gh, since those are the only parts of the sides of R that do not belong to any of R_i. But a line segment connecting any point on ab to any point on bc must intersect R_1, which contradicts that none of the rectangles R_i intersects C. Hence there must be at least one rectangle R_i that intersects C.

Suppose R_1 is the only rectangle R_i that intersects C. Then C must intersect the line segments cd and ef since those are the only parts of the sides r_2r_3 and r_3r_4 respectively that do not belong to any of the R_i assumed not to intersect C. But a line segment connecting any point on cd to any point on ef must intersect the rectangle R_3, which contradicts that R_1 is the only rectangle that intersects C. We conclude that at least two of the rectangles R_i must intersect C.

We now show that not more than two of the rectangles R_i can intersect C. Suppose at least three of the rectangles R_i intersect C. Let the rectangles be R_1, R_2 and R_3. Let p_1, p_2 and p_3 be points belonging to C in the rectangles R_1, R_2 and R_3 respectively. It is easy to see from Figure 5 that point r_1' is inside the triangle $p_1p_2p_3$, and hence inside C which is a contradiction. Hence we can have at most two rectangles R_i that intersect C. We conclude that exactly two of the rectangles R_i intersect C.

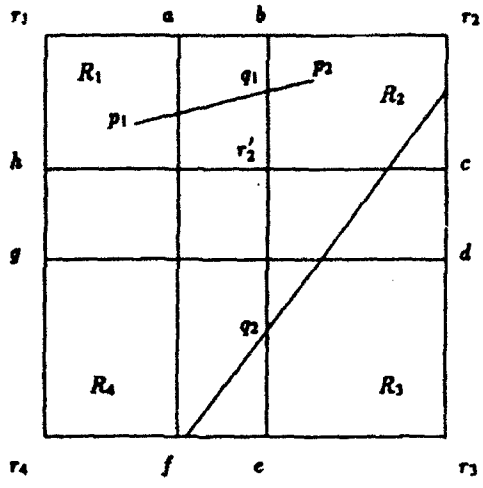


Figure 6: Illustration for Theorem 7

We now show that the two rectangles R_i that intersect C, must be diagonally opposite. Suppose two adjacent rectangles R_1 and R_2 intersect C. Then R_3 and R_4 cannot intersect C, since exactly two of R_i intersect C. The line segment ef must intersect C since that is the only part of the side r_3r_4 that is not in the rectangles R_3 or R_4. Let p_1 and p_2 be points in the rectangles R_1 and R_2 respectively that belong to C, as shown in the Figure 6. A line connecting p_1 and p_2 will intersect the segment br_2'. Let the point of intersection be point q_1. Also, since C is convex and tightly bounded by R, there must be a line segment contained in C that connects a point on the side r_2r_3 to a point on the line segment fe. Since the line segment ab is shorter than the line segment br_2, and the line segment er_2' is longer than the line segment $r_2'b$, any line segment connecting a point on the side r_2r_3 to a point on the line segment ef must intersect line segment $r_2'e$, say at point q_2. Since q_1 and q_2 belong to C and r_2' lies on the line connecting them, r_2' must belong to C. This is a contradiction to the conditions of this theorem. Hence two adjacent R_i cannot intersect C.

Hence, exactly two diagonally opposite R_i must intersect C.

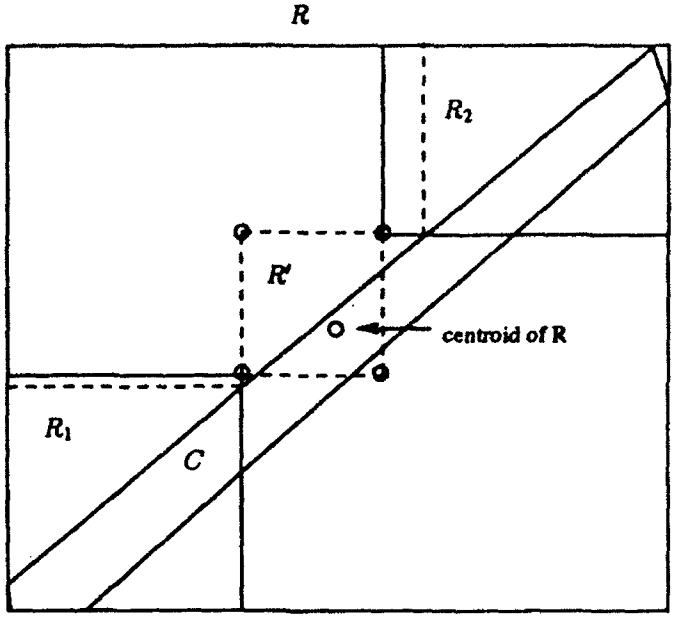
End of Proof.

We conclude that if a convex region (larger than a certain size) contains an integer point, then either an integer point close to the centroid of its bounding rectangle is inside the convex region, or the convex region region can be divided into two smaller regions, at least one of which must contain an integer point. Based on these results, we have the following procedure to determine if a given convex region contains any integer points.

procedure: FindIntegers(C, R)
input: Convex region C defined by inequalities in two variables, say x and y. Rectangle R tightly bounding C.
output: Boolean value representing whether an integer solution exists in the region.

1. If R covers more than 4 integers in both dimensions, GOTO step 2. Otherwise, without loss of generality, say x is the variable which can take at most 4 integer values. For each possible integer value of x, determine the range of values y can take. If an integer is found in the value range of y, return TRUE, else return FALSE. This is an exhaustive search in a small space.

2. Find the centroid of R. If it is an integer point, return TRUE. Let R' be the smallest square with integer points as corners enclosing the centroid of R. If any of the four corner points of R' is part of C, return TRUE. Otherwise GOTO step 3.



C : Convex region under consideration.

R : Smallest rectangle enclosing C

R' : Smallest integer square enclosing centroid of R

R_1, R_2 : Bounding rectangles for the new problem

Figure 7: Nature of Convex Regions with no points in the vicinity of the Centroid of the Bounding Rectangle

3. Find the two sides of R' that intersect C. Determine the two rectangles R_1 and R_2 as shown in Figure 7. Define C_1 and C_2 by adding the new boundaries of R_1 and R_2 respectively, to C as additional inequality constraints. Find the new strictly bounding rectangles R'_1 and R'_2 for C_1 and C_2 respectively using Fourier-Motzkin elimination. Return (FindIntegers(C_1, R'_1) *OR* FindIntegers(C_2, R'_2)).

complexity: This procedure divides the problem into two smaller problems that are half the size. The division stops at problem size 4. Thus the maximum number of times the above steps may have to be repeated is $n/4$, where n is the length of the bounding rectangle for the original convex region C.
end of procedure

4.2 Solution for n Dimensions

The results obtained for 2 dimensions cannot be directly extended to n dimensions. However, intuition suggests that the centroid of the bounding rectangle is still a good position to potentially establish an integer point. We use this to divide and shrink the region for analysis, until a solution is found or the problem is reduced to 2-dimensions.
procedure: NdimFindIntegers(n, C, R)
input: A set of inequalities with n variables representing a closed convex region C. Values X_i^{max} and X_i^{min} for $i = 1$ *to* n denoting an $n-$rectangle R strictly enclosing C.
output: Boolean value representing whether an integer solution exists in the region.

1. If range in any dimension does not contain an integer, no integer solutions can exist, so terminate returning FALSE.

If the product of the the ranges is a small integer, perform an exhaustive search for solutions in the integer space. If the number of dimensions is two, Return (FindIntegers(C, R)). Otherwise go to step 2.

2. Determine the centroid of the given n-rectangle R. Let R' be the smallest n-rectangle with all integer vertices that contains the centroid of R. If any of the integer vertices of R' belong to the convex region C, return TRUE, else GOTO step 3.

3. Let X_i be the dimension in which R contains the smallest number of integers. If the number of integers is greater than two, GOTO step 4. Otherwise obtain (at most) two new convex regions in $n-1$ dimensions by giving occurrences of X_i fixed integer values within this range. Suppose C_1 and C_2 are the new regions obtained. Find the smallest enclosing n-rectangles R_1 and R_2. Return (NdimFindIntegers($n-1, C_1, R_1$) *OR* NdimFindIntegers($n-1, C_2, R_2$)).

4. Let x_1 and x_2 be the two values in the center of the range of values with $x_1 < x_2$ that X_i can take. Add additional constraints $X_i <= x_1$ and $X_i >= x_2$ to C to get two new convex regions C_1 and C_2 respectively. Find their smallest bounding n-rectangles R_1 and R_2. Return (NdimFindIntegers(n, C_1, R_1) *OR* NdimFindIntegers(n, C_2, R_2)).

end of procedure

The steps of these solution procedures are organized so that most common cases can be handled quickly, but the less common cases are also handled. It is not possible to do an accurate complexity analysis of this solution procedure since it is dependent on numerical values. However, we expect the procedure to be efficient for problems with very few variables and small ranges, which is what we expect in our applications.

4.3 Handling Infinite Regions

The solution procedures so far assumed that the ranges for all variables are finite. This may not be true in general. An infinite convex region would include zero or infinite integer points. It is obvious that an infinite convex region bounded by diverging hyperplanes would include infinite integer points. A necessary condition for an infinite convex region to have no integer points is that there are parallel hyperplanes bounding the region (with an infinite thin strip with no integer points between them). We have the following lemma:

Lemma 8 *An infinite size convex region must have infinite integer points unless at least two of the hyperplanes defining it are parallel.*

For our analysis, we check if any of the hyperplanes defining the convex region are parallel. If no two hyperplanes are parallel, no further analysis is required. If parallel hyperplanes do exist, we generate heuristic values for bounds whenever actual values are not available. A heuristic value that we use for bounds is the largest coefficient in the system of inequalities. This is supported by the fact that the occurrence of integers inside infinite convex regions is usually cyclic along the axes in which it is infinite, and the lengths of cycles are determined by the magnitude of the integers in the inequality set. A more exact analysis could be performed to determine a better defined bound, but we do not have evidence that would suggest that our heuristic is inadequate.

5 Related Work

Exact data dependence analysis has been addressed by several research groups in the recent past. We compare our method to the work of Maydan et. al. [6], Omega test [7] and Power test [11]. The first step in our solution procedure, that is transformation of equality constraints to inequality constraints (which may yield a solution in some cases), is also used in the approaches stated above. The difference in the methodologies used is discussed in section 2.4. The second step that we apply is Fourier-Motzkin elimination to determine the existence of a real solution to the transformed system. All the other methods also use Fourier-Motzkin elimination when needed. Maydan et. al. examine the form of the equation system at this stage and check if any of a set of simple special tests is applicable, before employing Fourier-Motzkin elimination. In power test, a small approximation is introduced to enable an efficient solution methodology, and hence the results may be inexact. In Omega test, a modified version of Fourier-Motzkin elimination is used, which can prove the existence of an integer solution in many situations. We have not specifically addressed the best way of performing this step, but many optimizations are certainly possible..

The unique feature of our approach is the final step, which is reached if none of the above two steps are able to prove or disprove the existence of an integer solution. Maydan et. al. suggest using a branch and bound strategy at this stage. In Omega test, the final equation system is solved by successively eliminating variables by building a new system for every feasible value of the variable. We employ a sophisticated search strategy based on geometric properties of convex regions. In particular, the fact that an integer solution in a closed convex region is likely to be close to the centroid of the region, is used to guide the search procedure. If a step does not yield a solution, it generates a pair of two smaller, more constrained convex regions as the possible solution spaces.

6 Conclusions

We have presented a methodology for determining whether a non-negative integer solution for a system of equations exists. Our experience suggests that the method is suitable and sufficient for equation systems that need to be solved in a compiler for parallel languages. The research was motivated by the problem of determining if synchronization in a parallel program is sufficient to ensure sequentially consistent results [9]. It is particularly important to have an exact solution when examining race conditions in parallel programs, since the existence of an exact solution is used to guarantee deterministic parallel execution.

However, the approach is general and can be used for data dependence analysis and other small problems with similar characteristics. Although the problems involved are NP-hard, in practice we can solve almost all real instances without much trouble, because of practical limits on their sizes.

References

[1] R. Allen and K. Kennedy. Automatic translation of FORTRAN programs to vector form. *ACM Transactions on Programming Languages and Systems*, 9(4):491–542, October 1987.

[2] D. Callahan, K. Kennedy, and J. Subhlok. Analysis of event-synchronization in a parallel programming tool. In *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Seattle, WA, March 1990.

[3] R. J. Duffin. On fourier's analysis of linear inequalit, ystems. *Mathematical Programming Study*, 1:71–95, 1974.

[4] R. Garfinkel and G. Nemhauser. *Integer Programming*. John Wiley and Sons, New York, 1972.

[5] A. Kaufmann and A. Henry-Labordere. *Integer and Mixed Programming*. Academic Press, New York, 1977.

[6] D. Maydan, J. Hennessy, and M. Lam. Efficient and exact data dependence analysis. In *Proceedings of the ACM SIGPLAN 91' Conference on Program Language Design and Implementation*, pages 1–14, Toronto, Canada, June 1991.

[7] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of Supercomputing '91*, pages 4–13, Albuquerque, NM, November 1991.

[8] H. Smith. On systems of linear indeterminate equations and congruences. *Philosophical Transactions of the Royal Society of London*, A(151):293–326, 1861.

[9] J. Subhlok. *Analysis of synchronization in a parallel programming environment*. PhD thesis, Dept. of Computer Science, Rice University, August 1990.

[10] M. Wolfe. *Optimizing Supercompilers for Supercomputers*. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1982.

[11] M. Wolfe and C. Tseng. The power test for data dependence. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):591–601, September 1972.