World Journal of Advanced Engineering Technology and Sciences

eISSN: 2582-8266

Cross Ref DOI: 10.30574/wjaets

Journal homepage: https://wjaets.com/

WJAETS
World Journal of Advanced Engineering Technology and Sciences

(Research Article)

Check for updates

A simplex approach for binary linear programming

Subhendu Das *

24300 Abbeywood Drive, Los Angeles, California, 91307, USA.

World Journal of Advanced Engineering Technology and Sciences, 2024, 11(02), 508–521

Publication history: Received on 22 February 2024; revised on 09 April 2024; accepted on 12 April 2024

Article DOI: https://doi.org/10.30574/wjaets.2024.11.2.0119

Abstract

This article is a research, design, and development effort, on an algorithm for Binary Linear Programming (BLP). The idea of the algorithm is based on the concepts borrowed from the Simplex method of Linear Programming (LP). The Simplex method has both column selection and row selection logics for generating a pivot. In this article we select both logics of LP to use as only the column selection logic for the BLP. The article provides the complete implementable C-language source code of this version of the BLP algorithm to show all the details of the logics. In addition, it provides a simple software requirement for embedded engineering products based on the laws of nature. It helps to design, modify, and maintain a simpler code for the algorithms. Therefore, the article uses a multidisciplinary approach. It integrates optimization theory, matrix algebra, computer programming, software engineering design requirements, social science, laws of nature, etc. This BLP algorithm will shed some light on the P-NP problem and its solution.

Keywords: Binary linear programming; Source code; Simplex method; Pseudo Inverse; Software requirements; Laws of nature; P-NP example.

1. Introduction

A Linear Programming (LP) problem can be defined [Luenberger, p11] using the following set of expressions (1-3):

Minimize the linear objective function:

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n = z \qquad (1)$$

Subject to the linear equality constraints

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned} \qquad (2)$$

And the non-negativity constraints

$$x_1 \geq 0, x_2 \geq 0, \cdots, x_n \geq 0 \qquad (3)$$

Here $\{a_{ij}: i = 1,..m; j = 1..n\}, \{b_i: i = 1,..m\}, \{c_j: j = 1,..n\}$, are given real constants and $\{x_j, j = 1, \ldots n\}$ are real numbers to be determined as solutions of the LP problem.

*Corresponding author: Subhendu Das, Subhendu11Das@gmail.com

A Binary Linear Programming (BLP) problem can be similarly defined except the last inequalities in (3) on x_js' are replaced by the following expression (4):

$$x_j = 0\ or\ 1;\ \ for\ all\ j = 1..n \tag{4}$$

That is x_js' take binary values {0, 1} only, instead of all nonnegative real values.

The BLP is considered as NP complete problem [Dasgupta, p. 243]. It seems then that this BLP algorithm presented here will resolve an important issue. Knapsack problem also falls on the same class, and the BLP algorithm [Das, 2014] has been effectively used to solve that problem also. The algorithm given here is a variant of the above one, it is simpler, and explained clearly, and comes with its source code providing all the details. The source code is also simpler, because it uses simpler software requirements. The code should be quite handy for engineers and mathematicians.

2. The simplex method for LP

In the Simplex method, the LP problem is solved by a sequence of tables [Luenberger]. Equations (1) and (2) are converted into tables in the form of matrices of columns and rows. In each table a pivot is selected and a Gaussian elimination method is used to solve for one variable of the equations. The Simplex method is an iterative process, meaning a variable once selected as solution may be replaced in another step by another variable or value.

There are two important steps in the Simplex method: (1) A column selection logic (CSL) and (2) A row selection logic.

- In the CSL the Simplex method selects the most negative element in the reduced cost row, as the next column in the optimization process. Let us say that j = s is the index for such a selected column.
- Also in the Simplex method, the row selection logic, computes the following ratios as described by (5)

$$r_{is} = \frac{b_i}{a_{is}} \quad for\ all\ i = 1..m \tag{5}$$

The one with the minimum value for this ratio is selected as the new row. Let k be this row index. Then the entry a_{ks} is selected as the pivot element for the next iteration.

3. BLP-CSL method

In this BLP (Binary Linear Programming) algorithm, we call the method as the Column Selection Logic of Luenberger (CSLL). It is actually inspired by the Simplex method. But there is a major difference: there is no row selection, and therefore no pivoting. It has one CSL for zero assignment and another CSL for one assignment. The CSLL process is not iterative; therefore it finishes in at most n steps, where n is the number of variables in the BLP program [Das, 2014].

3.1. CSL for Zero-Assignment

We compute the ratio

$$r_{ij} = \frac{b_i}{a_{ij}} \quad for\ all\ i = 1..m; and\ j = 1..n \tag{6}$$

That is, for every column, we compute the row selection logic (5) of the Simplex method. This matrix is called the ratio test matrix (RTM). The one element that gives the minimum value of the ratios is examined and analyzed. If this minimum ratio element is less than one then that column variable is assigned a zero value. Otherwise this step is ignored. This has some difference with [Das, 2014].

The reason for this is very simple, since $x_j = 1$, we can write in equation (7):

$$\left(\frac{b_i}{a_{ij}} < 1\right) \Rightarrow \ (a_{ij} > b_i) \Rightarrow (a_{ij}\,x_j > b_i) \tag{7}$$

This single element will invalidate the i-th constraint. Thus x_j cannot take a positive value of one and therefore this CSL criterion indicates that x_j must be assigned a zero value. Observe that at this stage, the optimal choice is the variable $x_j = 0$, and the choice cannot be $x_j = 1$.

3.2. CSLL for One-Assignment

Pseudo inverse is very intimately related to the BLP problem. In [Das, 2014] article dual Pseudo Inverse matrices were used, to create a correlation matrix, for column selection logic, for the assignment of the unity value for the x-variables. In this article we use another well known method of pseudo inverse matrix for the Simplex method for the BLP program. We use the notations and the logic of [Luenberger] for the Simplex method. This method can be found in many other textbooks also.

The steps here then will be (a) Find an estimate x^e using the pseudo inverse (b) Multiply each component of x^e by the corresponding component of the cost vector c^T. (c) Choose the index j for the maximum of these products, and (d) Assign x(j) = 1 for that variable. Except the last step, this is clearly related to the Simplex method described earlier. Notice that we do not need to perform any pivoting; this happens because we use binary values only for the BLP.

In this subsection section we show why and how this BLP algorithm has some similarities with the Simplex method for the standard LP algorithm. Following [Luenberger, p55], we can decompose the BLP as:

$$A = [A_0\,, A_1], \qquad x = [x_0, x_1], \qquad c = [c_0, c_1] \qquad (8)$$

Where, 0 and 1 subscripts represent two partitions of the quantities corresponding to 0-values and 1-values of the x-variables. A is an mxn matrix, with m < n. We can write the BLP as

Maximize $\quad c_0^T x_0 + c_1^T x_1$

Subject to $\quad A_0 x_0 + A_1 x_1 = b,\ x_0 = 0,\ x_1 = 1$

Multiplying both sides by A_1^T gives us

$$A_0 A_1^T x_0 + A_1 A_1^T x_1 = A_1^T b, \Rightarrow A_1 A_1^T x_1 = A_1^T b - A_0 A_1^T x_0, \Rightarrow A_1 A_1^T x_1 = A_1^T b$$

The last expression happens since in BLP, $x_0 = 0$. Therefore we can estimate x_1 by:

$$x_1 = A_1^T (A_1 A_1^T)^{-1} b \qquad (9)$$

For the reduced cost we can then write $c_1^T x_1 = c_1^T A_1^T (A_1 A_1^T)^{-1} b$. Omitting the subscripts, we can simplify this CSLL (9) as:

Maximize

$$c^T x = c^T A^T (A A^T)^{-1} b \qquad (10)$$

In all iterations we compute the above formula (10) for new matrixes, and then choose the maximum element, as the column selection logic for the algorithm, to assign a value of one. Note, in below x_j is a real number, a pseudo estimate.

$$c^T x = \sum_{j=1}^{n} c_j * x_j \qquad (11)$$

We calculate each product term in (11), and then select the one term that has the maximum value, and the variable with that index is assigned a value of one. Each term is called the reduced cost row element in the Simplex method.

Various column selection logics can be used and all are very meaningful in the context of the Simplex method. The article [Das 2014] uses product of two pseudo estimates as correlation in the weighted least square idea. In this article we use only one estimate. We can also use the A matrix as the weight $y^T A x$ for the reduced cost row. This A-matrix weight idea comes from the duality theory of the LP. It seems it will be possible to implement this algorithm using the multi-stage

Dynamic Programming technique. We leave that for another research. We could not show all the examples here, because of the lack of space, too much of which is consumed by the source code listing.

4. Source code requirements

The guiding principle behind these requirements is - know the laws of nature, and then find the ways to obey them. The laws of nature can never harm us. We are using them for billions and trillions of years. The laws of nature are the only truths.

4.1. Dynamic Memory should not be used

The DMA (Dynamic Memory Allocation) is a very common feature of C language [Press], but it should be avoided. The memory should not be dynamically allocated and then de-allocated in the software. In the universe, the memory of every object remains existent for eternity. Thus very high quality seer yogis will see the entire history of the universe for eternity, over billions or trillions of years, depending on their quality or the scale of their yogic power. We can see the proof of that now through the UFO-ETs [Das, 2023]. Such stories are written in Vedas and therefore can be found also in the ancient books of Judaism and Christianity. Thus the DMA is a very harmful mechanism for software based products; it introduces uncertainty in the performance of systems.

4.2. Privacy vs. Democracy

Privacy and democracy are opposite concepts. They cannot survive together. You cannot have any secret with your spouse, for example. You will never be able to talk freely and openly if you have some secrets or privacy on some subjects between the two of you. This is true for every object in the universe, including nations, cultures, governments [Das, 2020], etc. Universe has the highest level of democracy. As mentioned before, the entire memory history of all objects, for billions of years, is eternally existent for all times, and is visible to every other object. If you can acquire the skills of the third eye vision, also called the divine vision, then you will be able to watch this memory as a 3D video. This video keeps even the emotions and feelings for every moment of your life. Through this memory history every soul wants to teach their life experiences to us, so that we can be emancipated.

Thus in nature, you cannot hide anything from anybody. Therefore in the software also you should follow that rule. In C++ language the class structure attempts to hide variables and functions from global access. This process introduces complexities and wrong philosophy in our thought process. They violate the definition of truth, a fundamental concept required for every human. Software is nothing but our thoughts. More democratic you make, more honest becomes our life. This will improve the security of our software code. In this BLP source code, global variables, and only the simple features of C language have been used. Clearly, the code can still be improved.

4.3. No Pointers and Jumping Around

There are no jumps in the laws of nature. Everything is so well planned and so deterministic that you will always obey the rules of straight sequential logic. There is no if-then-else logic in the universe; there is no freewill. Compilers can always be changed to eliminate pointers and make designs robust, lives simpler, and reliable. The laws of nature should always be our references; they cannot harm us. The nature has perfected itself over eternity; it is periodic, it dies, and reincarnates. There are two features in every object, one that is visible by natural eyes, which dies; and there are invisible parts, like the subtle bodies, which live for eternity.

4.4. OS and Embedded Engineering

Nature does not have any kind of consistency and uniformity. Every soul has complete freedom to create the body it wants. It is correct that, all of us normally have five fingers, but look at them, they are all different; and that never bothered us. Imposing rules on humanity and on our activities cause lot of pain and suffering. OS attempts to create such rules.

In nature there is no operating system for all humans, all animals, or all stars, etc. Even though the destiny is the highest level law, but the destiny of every individual object is customized. It is the destiny that creates the physical system, not the other way around. The destinies of any two humans are unique, just like their facial features.

5. The BLP-source code

These are not the exact copy of the computer screen. They have been edited for better display, readability, and understandability of the computer program. Also not all files are included in this document. For complete display the interested readers should check the actual code files in the Google Drive; the link is provided here.

The following is the main program for this BLP-CSLL algorithm. This function calls two parts of the algorithm: (a) Assign Value0 part and the (b) Assign Value1 part. After each assignment the code updates the data base by calling Update Database function.

```
//===============================================================================
int main(void) { // in file BLPmain.cpp

BLPinitializationProcess();

//nCols = columns of A matrix; nColsN is used to exit this loop
for (itern = 0; itern < nCols; itern++) {

    RTMmain();          //RatioTest - assignValue0 to solution-X[j]
    UpdateDatabase();  //Create new AN-matrix

    CSLL_Rc_Main();    //ReducedCost - assignValue1 to solution-X[j]
    UpdateDatabase(); //Create new AN-matrix
    }

_fcloseall(); //close all open files
}
```

The following section implements Assign Vlue0 part. It creates the ratio test matrix and then test if any element of the matrix is less than one.

```
//===============================================================================
void RTMmain (void) { //in file RTMmain.cpp

     CreateRTM();//Create the Ratio Test Matrix

     TestRTM();  //Test for the ratio bi/aij LE 1
}
```

This code snippet initializes the ratio test matrix to zero values. This is needed, for the code is repeated in each iteration of the algorithm. In the next double loop the code fills the matrix element with the required ratios for the new database.

```
//===============================================================================
void CreateRTM (void) {   //Ratio Test Matrix

 for (i = 0; i < mRows; i++) {
 for (j = 0; j < nColsN; j++) //number of columns changes every iteration
          RTM[i][j] = 0.0;   //initialize to a null matrix
 }
      for (i = 0; i < mRows; i++) {
      for (j = 0; j < nColsN; j++) {

           if (AN[i][j] != 0)
           RTM[i][j] = bColVectorN[i] / AN[i][j]; //Create the ratio
}}}
```

```
//==============================================================================
void TestRTM (void)
{ //test the ratio for LT 1 and NE 0
  //RTM[1][1] = 0.8; //for testing AssignValue0

rtmFlag = 1;
rtmMin = RTM[0][0];
       //Find the minimum ratio
       for (rtmC = 0; rtmC < nColsN; rtmC++){
       for (rtmR = 0; rtmR < mRows; rtmR++){
           if (RTM[rtmR][rtmC] < rtmMin){  // 0<r<1
              rtmMin = RTM[rtmR][rtmC];
              rtmC0 = rtmC; //save the index of minimum
}}}
       if(rtmMin < 1){  //when r<1, or aij > bj; assign o to xj
       rtmFlag = 0;     //detected a column
       AssignValue0(); //set the variable Xj = 0;
}
       if (rtmFlag == 1) {
       if (fptrOutput != 0) {
          fprintf(fptrOutput, "%s\n",
          "  TestRTM: No variable was set to zero.");
          fprintf(fptrOutput, "\n");
}}}

//==============================================================================
void CSLL_Rc_Main (void) {   //in file CSLL-RcMax.cpp
     CSL_LuenbergerMethod()  //finds reduced cost row
     FindRCmax();            //finds index-[j] of maximum cost
     SubtractAcolumn();      //perform b-a[j]
     AssignValue1();         //assign X[j] = 1
}

//==============================================================================
void CSL_LuenbergerMethod(void) {//CSL=column selection logic
      ComputePinvAN();//pseudoInverse for different sizes of AN
      CSLL_Find_CTX();//row vector Cost[i]*X[i] = CSLLcTx[i];
      CopyVectorF2L(CSLLcTx, nColsN, rC);//cTx is rC, the reduced cost
}

//==============================================================================
void CSLL_Find_CTX(void) {
     MultiplyMatrixByVector (CSLL_PinvAN, nColsN, mRows, bColVectorN,
           mRows, CSLLx, nColsN); //CSLLx = PinvAN * bN, x-estimate

     for (i = 0; i < nColsN; i++)
          CSLLcTx[i] = CostRowN[i] * CSLLx[i];
          // cTx = array of C[i]*X[i]= CSLLcTx[i];
}

//==============================================================================
void UpdateDatabase(void) {   //called when CostRow[j] becomes zero
       FindNewColumnNumbers();//compute from iCostRow
       Assign_mRn(); //assign: m<n, m=n, m>n
       CreateDataAN();//remove zero elements from XN, AN, CostRowN
}

//==============================================================================
```

6. Google-drive link

The complete listing of the BLP source code can be found in the BLP-CSLL Google drive at the link below. The directory structure is shown below. The link for the Google Drive is –

https://drive.google.com/drive/folders/1MXDdJVivT-Rll4P2cPprEjegODS4PK7Y?usp=drive_link

6.1. File names in the Project Directory

"C:\.vs"	"C:\CSLL-RcMax.cpp"	"C:\CSLL-Article.vcxproj.user"
"C:\CSLL-Article"	"C:\DatabaseUpdate.cpp"	"C:\InputText.txt"
"C:\x64"	"C:\GaussJordan.cpp"	"C:\OutputText.txt"
"C:\allExternal.h"	"C:\PinverseTests.cpp"	"C:\CSLL-Article.vcxproj"
"C:\utility.h"	"C:\RTMmain.cpp"	"C:\CSLL-Article.vcxproj.filters"
"C:\BLPmain.cpp"	"C:\Utility.cpp"	
"C:\CSL.cpp"	"C:\Utility.zip"	"C:\CSLL-Article.sln"

A few things you have to do to make the software work on your computer. (a) You must have 2022 version of MS Visual Studio installed in your computer. It comes free with MS windows. (b) Copy all the files from the Google drive to a directory of your choice in your computer. (c) Modify the directory address of the input and output files to your file installation directory name. You may search the program for InputText name to find the existing location in the program. If you are a windows programmer this should be an easy task. (d) After it is correctly installed just build and then execute the program using the visual studio. It should run and create the OutputText file in your named directory.

7. InputText.txt file

You should eliminate the formatting from this file, before you use it. Use the file from the Google Drive. This section has been edited to increase readability.

Row Column size of the A-Matrix

- 3 5

The input data for the A-Matrix

- 0.8138079236 0.2112777074 0.02508563094 0.5125428155
- 0.6545084972 0.09549150281 0.09549150281 0.6545084972 1.000000000
- 0.4748286925 0.7354202041 0.8784358579 0.9569245129 1.000000000

The b-Column Vector

- 2.351436370 2.404508497 3.167173409

The C-cost row vector

- 1.034693911 0.8100068609 0.8731983764 1.358575743 1.717000249

Actual Known solution vector X0

- 11011

The Optimal Cost value Z0

- 4.920276764

8. OutputText.txt file

This is a small extraction of the output text file. Complete file is provided in the Google Drive link address.

Amount of printed output can be controlled by these flags.

- ReadBLPproblemInputDdata: Change Print flags to reveal all print data
- ReadBLPproblemInputDdata: begin from BLP input file
- DisplayBLPproblemInputData: Display original A Matrix sizes : mRows, nCols

mRows = 3   nCols = 5

DisplayBLPproblemInputData: Display original A Matrix: 3 x 5

- 1.00E+00   8.14E-01   2.11E-01   2.51E-02   5.13E-01
- 6.55E-01   9.55E-02   9.55E-02   6.55E-01   1.00E+00
- 4.75E-01   7.35E-01   8.78E-01   9.57E-01   1.00E+00

DisplayBLPproblemInputData: Display original right hand side:bColVector: 3

- 2.352.403.17

DisplayBLPproblemInputData: Display Intput Cost Vector :CostRow: 5

- 1.03 0.81 0.87 1.36 1.72

DisplayBLPproblemInputData: Display known solution Vector: X0: 5

- 1   1   0   1   1

DisplayBLPproblemInputData:Display known optimal cost

- Z0 = 4.920277

DisplayBLPproblemInputData: End - Reading original data from BLP input file

InitializeBLPiterationData: Display cost row vector with zeros, iCostRow: 5

- 1.03 0.81 0.87 1.36 1.72

InitializeBLPiterationData: End - Initilization

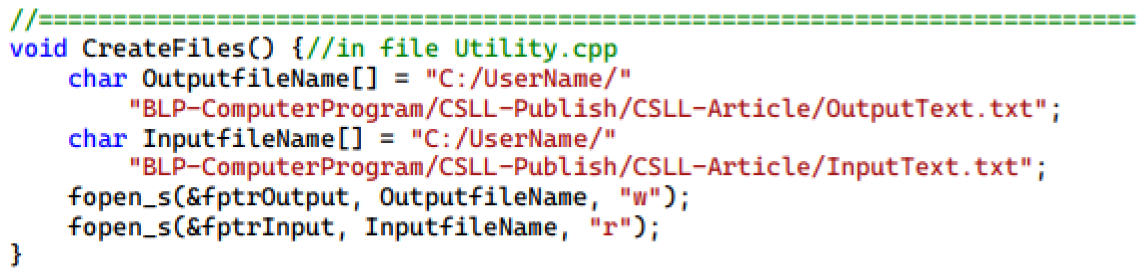
 ******Begin itern number =0*************

RTM Matrix =bi/aij, assign0 when 0<r<1,itern =0

- 2.35E+00   2.89E+00   1.11E+01   9.37E+01   4.59E+00
- 3.67E+00   2.52E+01   2.52E+01   3.67E+00   2.40E+00
- 6.67E+00   4.31E+00   3.61E+00   3.31E+00   3.17E+00

9. I/O file path names

User should modify the directory names in this following code snippet to make the I/O files work.



```
//==============================================================================
void CreateFiles() {//in file Utility.cpp
    char OutputfileName[] = "C:/UserName/"
        "BLP-ComputerProgram/CSLL-Publish/CSLL-Article/OutputText.txt";
    char InputfileName[] = "C:/UserName/"
        "BLP-ComputerProgram/CSLL-Publish/CSLL-Article/InputText.txt";
    fopen_s(&fptrOutput, OutputfileName, "w");
    fopen_s(&fptrInput, InputfileName, "r");
}
```

10. Pseudo inverse properties

10.1. Projection of a Vector

One vector can be projected onto another vector using the simple formula of inner product. In the figure-1, a, and x are two given vectors; p is the projection of x onto a; and the dashed line q, is the orthogonal component of x. They are defined using inner product formula (12) as:

$$\langle a, x\rangle = |a||x|Cos(\theta)$$

$$P(x) = p = \frac{\langle a, x\rangle}{\langle a, a\rangle} a = \hat{a}\ |a||x|Cos(\theta)\ \ and\ q = x - p \qquad (12)$$

It is easy to see that $\quad x = P(x) + (x - P(x)) = p + q$

The last expression is the orthogonal decomposition of x into the sum of one component parallel to a, and another component orthogonal to a. Notice that $(\langle a, x\rangle / \langle a, a\rangle)$ is a scalar quantity. Thus each component of a, is multiplied by the same scalar factor [Carrell, p196] in expression (12), to get the projection vector.

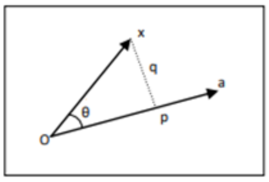


Figure 1 Vector Projection

10.2. Projection over a Subspace

The above concept, described in Figure-1, has been extended over to any subspace of a vector space. More specifically we want to decompose a given vector x into orthogonal components over a subspace, and find how the process generates the pseudo inverse [Carrell, p300].

Let $A = (w_1 \dots w_m)$be the matrix, n x m, of the basis vectors of W, which is a subspace of R^n. A is a tall matrix, n>m, n is the number of rows and m is the number of columns. Then W is the column space of A and we can write, as the requirements of normality:

$$w = Av\ and\ A^T(x - w) = 0$$

Here w is a linear combination of vectors in W, and therefore is the projection; we need to find w. The component (x-w) will be orthogonal to subspace W, since the rows of A^T span W, therefore the inner products must be zeros. See the diagram above.

$$A^T x = A^T w = A^T A v$$

$v = (A^T A)^{-1} A^T x\ therefore\ w = Av = A(A^T A)^{-1} A^T x$ - is the projection.

x-w = $x - A(A^T A)^{-1} A^T x = [I - A(A^T A)^{-1} A^T]x$- is the orthogonal component.

The matrix $(A^T A)^{-1} A^T$ is called the pseudo inverse, A^- of the tall matrix A. It satisfies the four conditions of the pseudo inverse [CalTech], the source code tests these four properties. For any matrix ∊ R^{nxm} , we can see that AA^- is an orthogonal projection matrix, onto the column space of A.

Since the matrix A has full row rank, the above inverse always exists. We can see the role played by the pseudo Inverse of a matrix in the BLP algorithm. Note that the inverses are different, see [CalTech].

For a long matrix, case m < n,

$$A^{-} = A^{T}(AA^{T})^{-1} \qquad (13)$$

And for a tall matrix, case m > n,

$$A^{-} = (A^{T}A)^{-1}\, A^{T} \qquad (14)$$

Here m and n are probably creating confusions. But the context should help. Tall and long are, maybe, better notations.

11. Theoretical analysis

Some theoretical results are presented below to show the soundness of the algorithm and the computational methods. They are all available in the existing literature. We just collect them appropriately. More definitions and details can be found in [Das, 2014].

Theorem: The CSLL is a polynomial time algorithm [Das, 2014]

Proof The computationally involved most complicated step in the CSLL logic is the evaluation of pseudo inverse of A-matrices at each iteration. In the simplest case of pseudo inverse, the process will evaluate the pseudo matrix directly from the relation $A^{-} = A^{T}(AA^{T})^{-1}$. This step takes two matrix multiplications and one matrix inversion. It is well known [Trahan] that they are polynomial time algorithms. Thus an iteration of the CSLL logic involves only polynomial time computations. Therefore the entire BLP algorithm presented is a polynomial time process.

Theorem: At every step the CSLL is an optimal step.

Proof It is clear from the CSLL algorithm that it makes a variable either 0 or 1. There are no other choices. A variable cannot take any real number value and also cannot remain undecided. Thus when it takes a 0 value, the optimality does not change, whatever was the value of the objective function that stays there in the present step. When it assigns a value of 1 then the objective increases, and always gives a higher optimality.

Theorem: CSLL finds the optimal solution

Proof Let us assume that u and v are two bit patterns or corner points of the cube claiming to be the optimal solutions. Without loss of generality we can assume that the cost coefficients are arranged in a decreasing order. Let k be the first bit position where u and v differ, that is, u has a 0 and at that same location v has a 1. This means v gives a higher value for the objective function. Now, this cannot happen since it is proven already that the CSLL gives the optimal value at each step. Therefore u must be same as v.

12. Test problems

One of the important requirements of the test problems is that it must have a proven known optimal solution; so that the results of the algorithm can be compared. And the other important factor is that the matrix A must be of full row rank, i.e., rank (A) = m, where m is the number of rows of A-Matrix.

Definition The set of functions $\{g_1(t), g_2(t), \ldots, g_n(t),\}$ is linearly dependent on an interval I = [a, b] if there are scalars $c_1, c_2, \ldots, c_n$ not all zero such that

$$c_1 g_1(t) + c_2 g_2(t) + \cdots + c_n g_n(t) = 0$$

for all t in I. Otherwise the set is linearly independent [Farlow, p.179].

$$\begin{array}{ll} g_1(t) = 1 + \mathrm{Sin}(2\,\pi\, f_0\, t) & g_2(t) = 1 + \mathrm{Cos}(2\,\pi\, f_0\, t) \\ g_3(t) = 1 - e^{-3000\,t} & g_4(t) = 1 + \mathrm{Cos}(4\,\pi\, f_0\, t) \end{array}$$

Figure 2 Examples of independent functions

To create such an A-matrix of any given size with m independent rows, we selected m independent functions, and then sampled those n times to produce the elements of the m x n matrices [Das, 2012]. All these functions are defined over a fixed time interval. Thus, as n increases the sample interval decreases. Some such functions are shown in Figure-2. It is clear that there are many choices of independent functions. Observe that all functions produced only nonnegative numbers. The characteristics of the problem-sets are quite dependent on the types of functions and the way they are sampled.

We have used random number generator for creating arbitrary binary solution vector x. To create an n-bit binary sequence, we used an integer of size 2^n. Then we generated a random integer number within that size, starting with a given seed value for the random generator. Finally we converted the random integer to a binary number with n-digits. We multiply A-matrix by these binary digits of x to get the b vector. So our BLP algorithm takes these A and b matrices and solves for the x. The solution will be correct only when it matches the known random x binary vector.

To ensure optimality, we used another random sequence of m real numbers for the dual variable y for the standard linear programming problem. Then we used $A^T y = c$ to generate the cost vector c. This will automatically ensure that the x vector is the optimal solution for the LP, because the variables so selected will satisfy the optimality condition:

$$y^T b = c^T x = y^T A x = z$$

It is clear that we are using a sufficient condition and not a necessary condition. For some of the problems, we verified the results using direct enumeration of all binary vectors [Das, 2014]. Observe that some random noise can be added in the data, but we avoided that in this algorithm. Our objective was to establish the algorithms and not their robustness.

It should be realized that this method of problem generation follows an important law of nature, as pointed out by [Dantzig, p. 33]. That is the law of conservation. This law is same as: material balance or law of conservation of mass, energy, force etc. It is also same as Kirchoff's current and voltage laws of electric circuit theory [Das, 2012a]. Thus a BLP problem example may be hypothetical or unrealistic or impractical if it violates the law of conservation. This law is also reflected in the relationship between the primal and the dual variables. Thus the coefficients in A, b, and c cannot be arbitrarily changed; they are related by this law.

We have chosen the non-negativity of elements of A to keep the computer program simpler. Our objective was not to venture for creating a commercially robust program. The focus is to provide a high level concept of the algorithm by eliminating complexities arising from negativities of A, b, and c parameters. The algorithm, once understood, can be easily customized for all such special cases. As we have seen that the solution of BLP does not depend on the nature of the feasible region and its corner points, which are defined by the A, b, and c matrices. The solution depends on the existence of the corner points of the unit cube inside the feasible region.

13. CSLL pseudo inverse code

In this CSLL algorithm the matrix sizes reduce from long matrixes to a square matrix and then to tall matrixes, as the variables are assigned 0-1 values, one step at a time. Thus three different inverse methods must be used. This part of the code performs that activity. The code also checks the conditions for the correct output of the inverses. The entire code is not copied here, but is provided in the Google drive.

The following four expressions are called Moore-Penrose conditions. When a matrix P satisfies all four of them then P is unique, and is called the pseudo inverse of A [CalTech]. This source code tests all four of them.

$$\bullet\ APA = A;\ \bullet\ PAP = P;\ \bullet\ (AP)^T = AP \quad \bullet\ (PA)^T = PA \qquad (15)$$

```
//===========================================================================
void ComputePinvAN (void) {
//if (mRows < nColsN) mRn = mLTn;  //1
//if (mRows == nColsN) mRn = mEQn; //2
//if (mRows > nColsN) mRn = mGTn;  //3

switch (mRn) {// m-n-relations
case mLTn: {//long matrix, case m < n,result is a tall matrix
//PinvAN = A'(AA')-inverse
mLTn_FindPinv();
break;
}//mLTn Pinv of a long matrix

case mEQn: {//mEQn square matrix
mEQn_FindPinv();
break;
}

case mGTn:{//PinvA = (A'A)inverse * A' ;
//result is a long matrix
mGTn_FindPiv();
break;
}//mGTn Pinv of a tall matrix

default: {
if (fptrOutput != 0) {
fprintf(fptrOutput, "  %s %i\n","CSL_LuenbergerMethod: case - default:"
                    "in itern = ", itern);
fprintf(fptrOutput, "  %s %i %i\n","CSL_LuenbergerMethod: case - default: "
        "mRows, nColsN = ", mRows, nColsN);
}
break;
}}}


  //===========================================================================
  void mGTn_FindPiv(void) {//Pseudo inverse of tall matrix
  TransposeOfMatrix(AN, mRows, nColsN, ANT);  //ANT = new AT
  MultiplyBCD(ANT, nColsN, mRows,
              AN, mRows, nColsN, ANTAN, nColsN, nColsN);//ANT*AN = ANTAN

  CopyMatrixF2L(ANTAN, nColsN, nColsN, GJX);
  GJrows = nColsN;
  GJmain();//call the GJ-inverse function = ANTANinv
  CopyMatrixF2L(GJXinv, nColsN, nColsN, ANTANinv);

  MultiplyBCD(ANTANinv, nColsN, nColsN, ANT, nColsN, mRows,
             CSLL_PinvAN, nColsN, mRows); //Pinv = ANTANinv * ANT
  Print_mGTn();

  Prows = mRows; Pcols = nColsN;
  CopyMatrixF2L(AN, mRows, nColsN, PA);
  CopyMatrixF2L(CSLL_PinvAN, nColsN, mRows, PinvA);

  TestPseudoInverse();//Test the four properties of the Pseudo Inverse
  }
```

```
//==========================================================================
void mLTn_FindPinv(void) { //Pseudo inverse of long matrix
TransposeOfMatrix(AN, mRows, nColsN, ANT);  //ANT = new AT
MultiplyBCD(AN, mRows, nColsN, ANT, nColsN, mRows,
            ANANT, mRows, mRows); // compute AN*ANT = ANANT

CopyMatrixF2L(ANANT, mRows, mRows, GJX);
GJrows = mRows;
GJmain(); //call the GJ-inverse for ANANT
CopyMatrixF2L(GJXinv, mRows, mRows, ANANTinv);//extract ANANTinv
MultiplyBCD(ANT, nColsN, mRows, ANANTinv,
            mRows, mRows, CSLL_PinvAN, nColsN, mRows);
            //Compute Pinv = ANT * ANANTinv
Print_mLTn();
Prows = mRows; Pcols = nColsN;
CopyMatrixF2L(AN, mRows, nColsN, PA);
CopyMatrixF2L(CSLL_PinvAN, nColsN, mRows, PinvA);

TestPseudoInverse(); //Test the four properties of the Pseudo Inverse
}
```

```
//==========================================================================
void mEQn_FindPinv (void) {//Pseudo inverse = inverse of a square matrix
CopyMatrixF2L(AN, mRows, mRows, GJX);// find the inverse of AN
GJrows = mRows;
GJmain(); //call the GJ-inverse function
CopyMatrixF2L(GJXinv, mRows, mRows, ANinv);
MultiplyMatrixByVector(ANinv, mRows, mRows,
            bColVectorN, mRows, CSLLx, mRows); // x = Ainv * b
Print_mEQn();
Prows = mRows; Pcols = nColsN;
CopyMatrixF2L(AN, mRows, nColsN, PA);
CopyMatrixF2L(ANinv, nColsN, mRows, PinvA);

TestPseudoInverse();//Test the four properties of the Pseudo Inverse
}
```

14. Conclusions

Someone who is using Microsoft Visual Studio 2022 to implement scientific programming work will have the environment already working in his Windows PC desktop computer. It should not be difficult to download the files for this program from the Google Drive, install this BLP software, and make it run.

If successful, then he will be able to apply this algorithm and the code for his own research projects on BLP or its applications. The theory behind this article is simple and is quite straight forward. It simplifies both the zero-assignment logic and one-assignment logic. Hope this article will help to solve the NP problem, as all NP problems are known to be equivalent. The algorithm has been tested on the Knapsack examples also.

The source code is complete and is self contained, in the sense that it includes Gauss-Jordan matrix inversion code, Pseudo inverse generation and testing code. The software has become simpler also to understand because of the new software requirements, with all global variables, and no hidden features.

References

[1] CalTech. 2024. The Moore-Penrose Pseudo Inverse, 3pages, available free from http://robotics.caltech.edu/~jwb/courses/ME115/handouts/pseudo.pdf , Accessed on 28March2024

[2] Carrell, J.B. 2005. Fundamentals of linear algebra, James B. Carrell, carrell@math.ubc.ca, July, 2005.

[3] Dantzig, G.B. 1963. Linear programming and extensions, R-366-PR, Part 1, Rand Corporation, California, 219 Pages. Available free from: http://www.rand.org/pubs/reports/R366.html

[4] Das, S. 2023. The Yogic Power of UFO-ETs. International Journal of Scientific Research and Engineering Development, 6, No. 2, Mar-Apr.

[5] Das, S. 2020. Truth and MLE – the only way for Eternal Global Peace on Earth. International Journal of Scientific and Research Publications, 10, No. 7.

[6] Das, S. 2014. "An Algorithm for Binary Linear Programming." Journal of Applied Mathematics and Bioinformatics, 4, No.3: 29-63.

[7] Das, S. 2012. Binary solutions for overdetermined systems of linear equations, AMO, v14, n1.

[8] Das, S. 2012a. Conservation laws of nature. Journal of Applied Global Research, 5(12). Available: https://s3-eu-west-1.amazonaws.com/pfigshare-u-files/1470570/LawsOfPhysicsJAGR.pdf (Accessed 8th March 2020).

[9] Dasgupta, S.,Papadimitriou,C.H., and Vazirani,U.V. 2006. Algorithms, UC Berkeley, California, 318 pages, available free from: http://www.cs.berkeley.edu/~vazirani/algorithms/

[10] Farlow, J. et al., 2002. Differential equations and linear algebra, Prentice hall, New Jersey, US.

[11] Luenberger, D.G., and Ye, Y. Linear and Nonlinear Programming, Third Edition, Stanford University, 551 Pages, available free from: http://grapr.files.wordpress.com/2011/09/luenberger-linear-and-nonlinear-programming-3e-springer-2008.pdf

[12] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P. 1992. Numerical Recipes in C, The Art of Scientific Computing, Second Edition. Cambridge university press, 1992. EBook.

[13] Trahan, J., Kaw, A., and Martin, K. Computational Time for Finding the Inverse of a Matrix: LU Decomposition vs. Naive Gaussian Elimination, University of South Florida, USA, Access it from: http://mathforcollege.com/nm/simulations/nbm/04sle/nbm_sle_sim_inversecomptime.pdf