

A NEW COMPUTER ORIENTED TECHNIQUE FOR SOLVING LINEAR PROGRAMMING PROBLEM USING BENDER'S DECOMPOSITION METHOD

Md. Anwarul Islam Bhuiyan^{1*} and Shek Ahmed¹

¹*Department of Mathematics, University of Barisal, Barisal 8200, Bangladesh*

Abstract

Linear Programming (LP) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships. Linear programming is a special case of mathematical programming. Decomposition technique is one of the most commonly used technique for solving Linear Programming Problems (LPP). There are many existing techniques for solving LPP. If the number of decision variables and constraints for LPP is large then it will be very difficult to solve manually. The purpose of this paper is to develop computer oriented decomposition technique for solving LPP using benders decomposition principle. By using this decomposition technique one can solve more complicated LPP by dividing original problem into two easier problems, namely Master problem and Sub problem. We demonstrate our technique by solving a relatively complicated LPP. For this purpose we have also developed computer code using A Mathematical Programming Language (AMPL) and present a comparison of results of manual output and programming output.

Keywords: Linear Programming, Linear Programming Problems, Benders Decomposition, A Mathematical Programming Language, Master Problem.

Introduction

Linear Programming Problems (LPP) in general are concerned with the use or allocation of scarce resource-laborers, materials, machines, and capital-in the best possible manner so that costs are minimized or profits are maximized. In using the term best it is implied that some choices or a set of alternative courses of actions is available for making the decision. In general, the best decision is found by solving a mathematical problem.

*Corresponding author's e-mail: anwar_1803@yahoo.com (M. A. I. Bhuiyan)

The development of linear programming (LP) is the most scientific advances in the mid 20th century. LP involves the planning of activities to obtain an optimal result which reaches the specialized goal best among all feasible alternatives. Numerous algorithms for solving LP problem have been developed in the past.

Benders Decomposition is a popular technique for solving certain classes of difficult problems such as stochastic programming problems and mixed-integer linear programming problems. Benders Decomposition is a technique in mathematical programming that allows the solution of very large linear programming problems that have a special block structure. This structure often occurs in applications such as stochastic programming. As it process towards a solution, Benders decomposition adds new constraints, So the approach is called “row generation”. In contrast, Dantzig-Wolfe decomposition uses “Column generation”.

A Mathematical Programming Language (AMPL) is an algebraic modeling language to describe and solve high-complexity problems for large-scale mathematical computing (i.e., large-scale optimization and scheduling-type problems). It was developed by Robert Fourier, David Gay, and Brian Kernighan at Bell Laboratories. One advantage of AMPL is the similarity of its syntax to the mathematical notation of optimization problems. AMPL supports a wide range of problem types such as Linear Programming(LP), Quadratic programming, Nonlinear programming, Mixed-integer programming, Mixed-integer quadratic programming with or without convex quadratic constraints, Mixed-integer nonlinear programming etc.

Objective of this paper is to develop a technique to solve more complicated LPP using Bender’s decomposition method by converting LPP into two easier problems namely Master problem and Sub problem. Finally our aim is to develop a computer code using A Mathematical Programming Language (AMPL).

1. Preliminaries

In this section we discuss some basic definitions and techniques relevant to our work.

1.1 Linear Programming Problem (LPP)

The Linear Programming Problem (LPP) is to find the decision variables x_1, x_2, \dots, x_n which is optimizing (minimizing or maximizing) the objective function.

$$z = c_1x_1 + c_2x_2 \dots \dots \dots + c_nx_n$$

Subject to the constraints,

$$a_{11}x_1 + a_{12}x_2 \dots \dots \dots + a_{1n}x_n (\geq, =, \leq) b_1$$

$$a_{21}x_1 + a_{22}x_2 \dots \dots \dots + a_{2n}x_n (\geq, =, \leq) b_2$$

.....

$$a_{m1}x_1 + a_{m2}x_2 \dots \dots \dots + a_{mn}x_n (\geq, =, \leq) b_m$$

and $x_j \geq 0; j = 1, 2, 3, \dots, n.$

The coefficients $c_j (j = 1, 2, 3, \dots, n)$ are called the cost coefficients. The constants $b_i (i = 1, 2, 3, \dots, m)$ in the constraints conditions are called stipulations and the constants $a_{ij} (i = 1, 2, 3, \dots, m, j = 1, 2, 3, \dots, n)$ are called the structural coefficients.

1.2 Duality in Linear Programming (LP)

Every linear programming problem whether it is of maximization or minimization is associated with its mirror image problem based on the same data. The original problem is often termed as primal problem while its image problem is called as its dual problem. However, in general either problem can be considered as primal and the remaining as the dual problem. Moreover, a solution to the primal problem also gives a solution to the dual problem and vice versa. Duality is an extremely important and interesting feature of LP.

1.3 Benders Decomposition

Benders Decomposition is a technique in mathematical programming that allows the solution of very large LPP that have a special block structure.

Benders Decomposition Principle for Linear Programming (LP)

Original Problem		Sub-problem	
(P)	Maximize	$cx + qy$	Primal Sub-problem
	subject to	$Ax \leq b$	Maximize qy
		$Tx + Hy \leq b$	subject to $Hy \leq b - Tx, \text{ depends on } x.$
		$x, y \geq 0$	$y \geq 0$
Master problem		Dual Sub-problem	
(M)	Maximize	$cx + v(S(x))$	(S(x)) Minimize $\lambda(b - Tx)$
	subject to	$Ax \leq b$	subject to $\lambda H \geq q$
		$x \geq 0$	$\lambda \geq 0$

2. Existing Techniques of Solving LPP

In this section, we have discussed some existing techniques briefly for solving LPP.

2.1 One Phase Simplex Method

Simplex Method (SM), also called simplex technique or simplex algorithm was developed by Dantzig (1947). The SM is an iterative procedure which either solve an LPP in a finite number of steps or gives an indication that there is an unbounded solution to the LPP. To solve any LPP by SM, the existence of an initial basic feasible solution is always assumed.

The steps for the computation of an optimum solution are as follows:

Step 1: Check whether the objective function of the given LPP is to be maximize or minimized. If it is to be minimized then we convert it into a problem of maximization by using the result

$$\text{Minimum } z = -\text{Maximum } (-z).$$

Step 2: Check whether all $b_i, (i = 1, 2, \dots, m)$ are non-negative. If any one of b_i is negative, then multiply the corresponding inequation of the constraints by -1, so as to get all $b_i, (i = 1, 2, \dots, m)$ non-negative.

Step 3: Convert all the inequalities of the constraints into equations by introducing slack and/or surplus variables in the constraints. Put the costs of these variables equal to zero.

Step 4: Obtain an initial basic feasible solution to the problem in the form

$$x_B = B^{-1}b$$

and put it in the first column of the simplex table.

Step 5: Compute the net evaluations $z_j - c_j (j = 1, 2, \dots, u)$ by using the relation $z_j - c_j = c_B y_j - c_j$. Examine the sign of $z_j - c_j$.

(i) If all $(z_j - c_j) \geq 0$ then the initial basic feasible solution x_B is an optimum basic feasible solution.

(ii) If at least one $(z_j - c_j) < 0$, proceed on to the next step.

Step 6: If there are more than one negative $(z_j - c_j)$, then choose the most negative of them. Let it be $(z_r - c_r)$ for some $j = r$.

(i) If all $y_{ir} \leq 0 (i = 1, 2, \dots, m)$, then there is an unbounded solution to the given problem.

(ii) If at least one $y_{ir} < 0, (i = 1, 2, \dots, m)$, then the corresponding vector y_{ir} enters in the basis y_B .

Step 7: Compute the ratios $\left\{\frac{x_{ni}}{y_{ir}}, y_{ir} > 0, i = 1, 2, \dots, m\right\}$ and choose the minimum of them. Let the minimum of these ratios be $\frac{x_{Bk}}{y_{kr}}$. Then the vector y_k will leave the basis y_B . The common element y_{kr} which is in the k th row and the r th column is known as the leading element (or pivot element) of the table.

Step 8: Convert the leading element to unity by dividing its row by the leading element itself and all other elements in its column to zeros by making use of the relations:

$$\frac{A}{y_{kj}} = \frac{y_{kj}}{y_{kr}}, j = 0, 1, 2, \dots, n \text{ and}$$

$$\frac{A}{y_{ij}} = y_{ij} - \frac{y_{kj}}{y_{kr}} y_{ir}, i = 1, 2, \dots, m; i \neq k.$$

Step 9: Go to step 5 and repeat the computational procedure until either an optimum solution is obtained or there is an indication of an unbounded solution.

This is easily achieved by the elementary row operations:

$$R' = R_k / y_{kr}$$

$$\text{and } R_i' = R_i - R_i' y_{ir}, i \neq k.$$

2.2 Big M Method

A standard method for handling artificial variables within the simplex method is the Big-M Method. To solve a LPP involving artificial variables, a method developed by Charnes is called Charnes penalty method or penalty method or Charnes M-method or Big-M method.

Step 1: Modify the constraints so that the right hand side (rhs) of each constraint is nonnegative. Identify each constraint that is now an = or \geq constraint.

Step 2: Convert each inequality constraint to standard form (add a slack variable for \leq constraints, add an excess variable for \geq constraints).

Step 3: For each \geq or = constraint, add artificial variables. Add sign restriction $a_i \geq 0$.

Step 4: Let M denote a very large positive number. Add (for each artificial variable) Max to min problem objective functions or -Min to max problem objective functions.

Step 5: Since each artificial variable will be in the starting basis, all artificial variables must be eliminated from row 0 before beginning the simplex. Remembering M represents a very large number, solve the transformed problem by the simplex.

2.3 Two Phase Simplex Method

Two phase simplex method is another form of Big-M method to solve a LPP involving one or more artificial variables including two phases. Although they seem to be different, they are essentially identical. However, methodologically the 2-Phase method is much superior.

Phase I:

The simplex method is applied to remove all artificial variables added to the constraints. Finally we may conclude that the LPP has feasible solution or not.

Phase-II:

Leads from the basic feasible solution determined by phase –I without artificial variables.

3. Proposed Technique for solving LPPs:

In this section we present Benders Decomposition Algorithm for solving LPP. We demonstrate the algorithm by the following steps:

Original Problem P:

$$\max cx + qy, Ax \leq b, Tx + Hy \leq b, x, y \geq 0$$

(variables are x and y, Hy may be a difficult, nonlinear or integer)

Master problem $M(\lambda^k)$:

$$\max cx + z, Ax \leq b, \lambda^k Tx + z \leq \lambda^k b,$$

$$k = 1, 2, \dots, K, x \geq 0, z \text{ free}$$

(variables are x and z (a scalar), λ^k is fixed)

Sub problem $S(x^k)$:

$$\min \lambda(b - T x^k), \lambda H \geq q, \lambda \geq 0 \text{ (variables are } \lambda, x^k \text{ is fixed)}$$

Initialize: Set $k = 1$, pick an x^k (perhaps from $\max cx, Ax \leq b, x \geq 0$)

Step 1: Solve $S(x^k)$: $\min \lambda(b - T x^k), \lambda H \geq q, \lambda \geq 0$.

Get λ^k .

Optimal if $v(S(x^k)) = v(M(\lambda^k))$.

Step 2: Solve $M(\lambda^k)$: $\max cx + z, Ax \leq b, \lambda^k Tx + z \leq \lambda^k b, k = 1, 2, \dots, K$,

$x \geq 0, z$ free . Get new x^k .

Step 3: Let: $k = k+1$ and go to Step 1.

4. Optimal solution of a numerical example by using proposed technique

In this section we solve a numerical example using proposed technique manually.

$$\begin{array}{ll}
 \text{Maximize} & 7x_1 + 6x_2 - 3x_3 - 5x_4 \\
 \text{Subject to} & 5x_1 + 3x_2 \leq 12 \\
 & 5x_1 + 9x_2 \leq 18 \\
 & 4x_1 - x_2 - 2x_3 - x_4 \leq 4 \\
 & 2x_1 + 3x_2 + x_3 - 2x_4 \leq 2 \\
 & -2x_1 + x_2 + 2x_3 - x_4 \leq 5 \\
 & x_1, x_2, x_3, x_4 \geq 0
 \end{array}$$

Solution:

Iteration-1:

Master problem

$$\begin{array}{ll}
 \text{Maximize} & 7x_1 + 6x_2 \\
 \text{Subject to} & 5x_1 + 3x_2 \leq 12 \\
 & 5x_1 + 9x_2 \leq 18 \\
 & x_1, x_2 \geq 0
 \end{array}$$

Master problem solution: $x_1 = 1.8, x_2 = 1$

Master value: 18.6

Primal Sub-problem

$$\begin{array}{ll}
 \text{Maximize} & -3x_3 - 5x_4 \\
 \text{Subject to} & 4x_1 - 2x_3 - x_4 \leq 4 - 4x_1 + x_2 \\
 & x_3 - 2x_4 \leq 2 - 2x_1 - 3x_2 \\
 & 2x_3 - x_4 \leq 5 + 2x_1 - x_2 \\
 & x_3, x_4 \geq 0
 \end{array}$$

Dual Sub-problem

$$\begin{aligned}
 \text{Minimize} \quad & \lambda_1(4 - 4x_1 + x_2) + \lambda_2(2 - 2x_1 - 3x_2) + \lambda_3(5 + 2x_1 - x_2) \\
 & = \lambda_1(4 - 4 * 1.8 + 1) + \lambda_2(2 - 2 * 1.8 - 3 * 1) + \lambda_3(5 + 2 * 1.8 - 1) \\
 & = -2.2 \lambda_1 - 4.6 \lambda_2 + 7.6 \lambda_3 \\
 \text{Subject to} \quad & -2\lambda_1 + \lambda_2 + 2\lambda_3 \geq -3 \\
 & -2\lambda_2 - \lambda_3 \geq -5 \\
 & \lambda_1, \lambda_2 \geq 0
 \end{aligned}$$

Sub problem solution: $\lambda_1 = 0, \lambda_2 = 2.5, \lambda_3 = 0$

Sub problem value: -11.5

Iteration-2:

Master problem solution: $x_1 = 2.4, x_2 = 0$ and $z = -7$

Master value: 9.8

Sub problem solution: $\lambda_1 = 2.2, \lambda_2 = 1.4, \lambda_3 = 0$

Sub problem value: -16.24

Iteration-3:

Master problem solution: $x_1 = 1.81395, x_2 = 0.976744$

and $z = -11.3953$

Master value: 7.16279

Sub problem solution: $\lambda_1 = 2.2, \lambda_2 = 1.4, \lambda_3 = 0$

Sub problem value: -11.3953.

Since, At iteration 3 the value of z and sub problem value are same so optimal solution is obtained.

Optimal Solution: $x_1 = 1.81395, x_2 = 0.976744, x_3 = 0$ and $x_4 = 2.27907$

[x_3 and x_4 are obtained from primal subproblem.]

5. Computer Code

In this section, we develop a computer code of our proposed technique. We have used a mathematical programming language AMPL. Our code consists of AMPL model file, AMPL data file and AMPL run file. But only model file and data have been presented in this paper. If readers are interested then they may contact with the authors.

AMPL model file:

```
# BENDERS DECOMPOSITION
param k>=1 default 1;      # iteration
### VARIABLE DECLARATION FOR MASTER ###
param nvm;                # no. of variables in master
param c {1..nvm};         # coefficients of objective function
param ncm;                # no. of constraints in master
param d {1..ncm,1..nvm};  # coefficients of variables in constraints.
param b {1..ncm};         # right hand constants
var xm {1..nvm}>=0;        # variables in master

### VARIABLE DECLARATION FOR SUB-PROGRAM ###
param nvs;                # no. of variables in subprogram
param a {1..nvs};         # coefficients of objective function
param nrs;                # no. of constraints in subprogram
param f {1..nrs,1..nvs};  # coefficients of variables in constraints
param e {1..nrs};         # right hand constants
var xs {1..nvs}>=0;        # variables of subprograms

### VARIABLE DECLARATION FOR PRIMAL SUB -PROGRAM ###
param nvp;                # no. of variables in primal sub problem
param g {1..nvp};         # coefficients of objective function
param ncp;                # no. of constraints in primal sub problem
param h {1..ncp,1..nvp};  # coefficients of variables in constraints
param r {1..ncp};         # right hand constants
var xp {1..nvp}>=0;        # variables in primal sub problem

### MASTER ( FOR FIRST ITERATION ) ###
maximize Master_1: sum {j in 1..nvm} c[j]*xm[j];
subject to const_master1 {i in 1..ncm}: sum {j in 1..nvm} d[i,j]*xm[j] <= b[i];
```

```

#### MASTER (FOR HIGHER ITERATION) ####
var z;
maximize Master_2: sum {j in 1..nvm-1} c[j]*xm[j] +c[nvm]*z;
subject to const_master2 {i in 1..ncm}: sum {j in 1..nvm-1} d[i,j]*xm[j]+d[i,nvm]*z <=
b[i];

#### SUB PROBLEM ####
minimize Sub_v: sum {j in 1..nvs} a[j]*xs[j];
subject to const_sub {i in 1..nrs}: sum {j in 1..nvs} f[i,j]*xs[j] >= e[i];

#### PRIMAL SUB PROBLEM ####
maximize primal_v: sum {j in 1..nvp} g[j]*xp[j];
subject to const_primal {i in 1..ncp}: sum {j in 1..nvp} h[i,j]*xp[j] <= r[i];

```

AMPL data file:

```

param nvm := 2;
param c:=
    1  7
    2  6;
param ncm := 2;
param d: 1 2:=
    1  5  3
    2  5  9;
param b := 9
    1  12
    2  18;
param nvs := 3;
param nrs := 2;
param f: 1 2 3:=
    1  -2  1  2
    2  -1  -2  -1;

param e :=
    1  -3
    2  -5;
param nvp := 2;
param g:=
    1  -3
    2  -5;
param ncp := 3;
param h: 1 2:=
    1  -2  -1
    2  1  -2
    3  2  -1;

```

AMPL Output System

Like other software such as FORTRAN, MATHEMATICA, MATLAB, LINDO etc. AMPL has an intrinsic system to run code. In AMPL model file and data file have to

write in different text files. Then one can generate a run file and have to call model and data file in that run file. AMPL has different solvers.

6. Optimal solution using Computer Code

We solve a numerical problem which is given in section 5 by our developed AMPL code. AMPL output has shown below:

6.1 Output of the numerical example

In this section we give a computer output by using AMPL.

Table 1. Output using computer code.

<pre> iteration = 1 MINOS 5.5: optimal solution found. 1 iterations, objective 15 xm [*] := 1.8 1; Solve Sub problem: 2 iterations, objective -42 xs [*] := 0 2.5 0; iteration = 2 Solve Master: MINOS 5.5: optimal solution found. 1 iterations, objective -12 xm[1] = 2.4 xm[2] = 0 z = -7; Solve Sub problem: MINOS 5.5: optimal solution found. 2 iterations, objective -24 xs [*] := 1 2.2 2 1.4 3 0; </pre>	<pre> iteration = 3 Solve Master: MINOS 5.5: optimal solution found. xm[1] =1.81395 xm[2] =0.976744 z = -11.3953 Solve Sub problem: MINOS 5.5: optimal solution found. 1 iterations, objective -21 xs [*] := 1 2.2 2 1.4 0; 2 iterations, objective -22.5 xp [*] := 1 0 2.27907 Optimal Solution: os [*] := 1 1.81395 2 0.976744 3 0 4 2.27907; ov = 7.16279 </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7. Result and discussion

In this section we compare the results between manual output and program output.

Table 2. Comparison between manual output and program output

Iteration number	Manual Output	Program Output
1	<i>Master problem solution:</i> $x_1 = 1.8, x_2 = 1$ <i>Master value:</i> $M_1(v) = 18.6$	<i>Master problem solution:</i> $x_1 = 1.8, x_2 = 1$ <i>Master value :</i> $M_1(v) = 18.6$
	<i>Subproblem solution:</i> $\lambda_1 = 0, \lambda_2 = 2.5$ and $\lambda_3 = 0$ <i>Subproblem value:</i> $S_1(v) = -11.5$	<i>Subproblem solution:</i> $\lambda_1 = 0, \lambda_2 = 2.5$ and $\lambda_3 = 0$ <i>Subproblem value:</i> $S_1(v) = -11.5$
	<i>Master problem solution:</i> $x_1 = 2.4, x_2 = 0$ and $z = -7$ <i>Master problem value</i> $M_2(v) = 9.8$	<i>Master problem solution:</i> $x_1 = 2.4, x_2 = 0$ and $z = -7$ <i>Master problem value</i> $M_2(v) = 9.8$
2	<i>Subproblem solution:</i> $\lambda_1 = 2.2, \lambda_2 = 1.4$ and $\lambda_3 = 0$ <i>Subproblem value:</i> $S_2(v) = -16.24$	<i>Subproblem solution:</i> $\lambda_1 = 2.2, \lambda_2 = 1.4$ and $\lambda_3 = 0$ <i>Subproblem value:</i> $S_2(v) = -16.24$
	<i>Master solution:</i> $x_1 = 1.81395,$ $x_2 = 0.976744$ and $z = -11.3953$ <i>Master problem value :</i> $M_3(v) = 7.16279$	<i>Master solution:</i> $x_1 = 1.81395,$ $x_2 = 0.976744$ and $z = -11.3953$ <i>Master value:</i> $M_3(v) = 7.162790698$
	<i>Subproblem solution:</i> $\lambda_1 = 2.2, \lambda_2 = 1.4$ and $\lambda_3 = 0$ <i>Subproblem value:</i> $S_3(v) = -11.3953$	<i>Subproblem solution:</i> $\lambda_1 = 2.2, \lambda_2 = 1.4$ and $\lambda_3 = 0$ <i>Subproblem value:</i> $S_3(v) = -11.395344884$

Form the above table we can say that our computer code in Section 6 gives as much as the same results as we have in Section 5. Moreover, the difference of getting the different results in just a few cases are caused by the tolerance and internal difference of carrying operations between Mathematical Coding and AMPL.

8. Graphical Representation of the Convergence of the Master and Sub problem Value

<<Graphics` Multiple List Plot`

`a={{1,0},{2,-7},{3,-11.3953}};`

`s={{1,-11.5},{2,-16.24},{3,-11.395344}};`

Multiple List Plot [a, s, Axes Label→ {"Iteration No. ","Objective Value"}, Plot Joined→ True, Plot Legend→ {value z, Sub}];

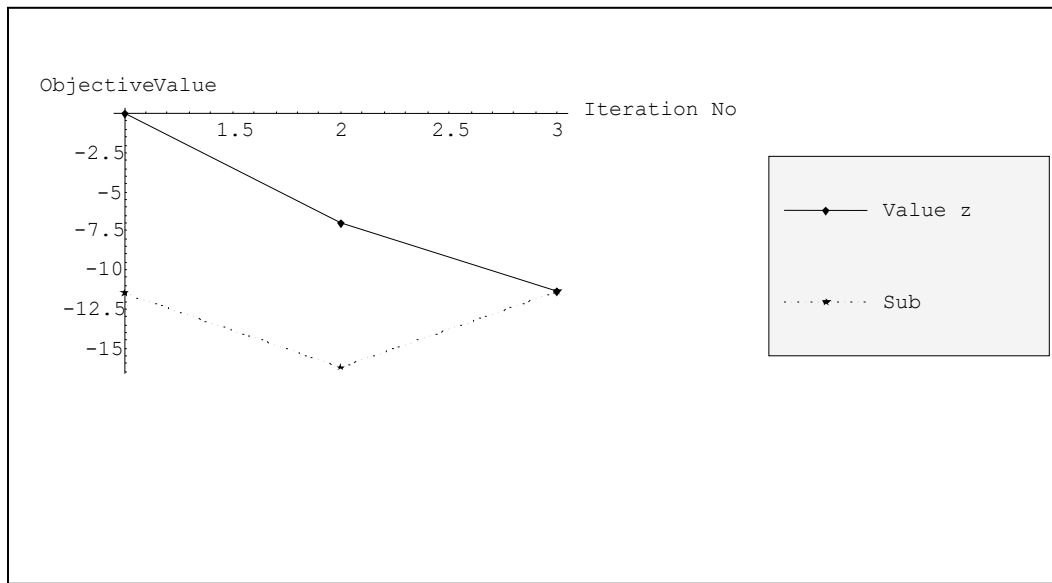


Fig 1. Convergence of the sub-problem and master-problem values.

9. Conclusion

In this paper we presented a new technique for solving more complicated LPPs. To develop this technique we used the idea of benders decomposition method. We also developed a computer code using AMPL for solving LPPs and so that we can easily solve LPPs using it. Graphical representations also illustrated to show the convergence of the master and the sub-problem values. using MATHEMATICA. This improved technique will be extended to solve the Integer Programming Hence we can conclude that our decomposition algorithm can be used as an effective tool for solving LPs to avoid the laborious calculations using row generation.

References

- Dantzig, G. B. and P. Wolfe. 1961. The decomposition algorithm for linear programming. *Econometrica*. **29**:767-778.
- Dantzig, G.B. 1963. *Linear Programming and Extensions*, Princeton University Press, Princeton. U.S.A.
- Don, E. 2000. *Theory and Problems of Mathematica*, Schaum's Outline Series, McGRAW-HILL.
- Fisher, M. L. 1979. The lagrangean relaxation method for solving integer programming problem. *Management Science*. **27**:75-83.
- Fourier, Robert; Gay, David M. 2002. Extending an algebraic modeling language to support constraint programming. *INFORMS Journal on Computing*. **14**:322-344.
- Geofrion, A. M. 1974. Lagrange relaxation for integer program. *Mathematical programming study*. **26**:82-114.
- Hasan, M. B. and J. F. Raffensperger. 2007. A decomposition based pricing model for solving a large-scale MILP model for an integrated fishery. *Journal of Applied Mathematics and Decision Sciences*. Vol. 2007, Article ID 56404, 10 pages.
- Held, M. and R. M. Karp. 1970. The travelling salesman Problem and minimum spanning tree. *Operations Research*. **18**:1138-162.
- Held, M. and R.M. Karp 1971. The travelling salesman problem and minimum spanning tree: Part II. *Mathematical Programming I*. **23**:6-25.
- R. Fourier, and Brian W. K. 2002. *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press.
- Sweeny, D. J. and R. A. Murphy. 1979. A method of decomposition for integer programs. *Operations Research*. **27**:1128-1141.
- Winston, W. L. 1994. *Linear Programming: Applications and Algorithm*, Duxbury press, Belmont, California, U.S.A.