International Journal of Pure and Applied Mathematics

Volume 4 No. 1 2003, 53-80

AN ALGORITHM FOR SIMPLEX TABLEAU REDUCTION WITH NUMERICAL COMPARISON

H. Arsham[1] §, P. Baloh[2], T. Damij[3], J. Grad[4]

[1]MIS Division
University of Baltimore
Baltimore, Maryland 21201, USA
e-mail: harsham@ubmail.ubalt.edu

[2,3]Faculty of Economics
University of Ljubljana
1000 Ljubljana, SLOVENIA
[2]e-mail: peter.baloh@uni-lj.si
[3]e-mail: talib.damij@uni-lj.si

[4]School of Public Administration
University of Ljubljana
1000 Ljubljana, SLOVENIA
e-mail: janez.grad@vus.uni-lj.si

Abstract: The simplex algorithm requires artificial variables for solving linear programs, which lack primal feasibility at the origin point. We present a new general-purpose solution algorithm, called Push-and-Pull, which obviates the use of artificial variables. The algorithm consists of preliminaries for setting up the initialization followed by two main phases. The Push Phase develops a basic variable set (BVS), which may or may not be feasible. Unlike simplex and dual simplex, this approach starts with an incomplete BVS initially, and then variables are brought into the basis, one by one. If the BVS is complete, but the optimality condition is not satisfied, then Push Phase pushes until this condition is satisfied, using the rules similar to the ordinary simplex. Since the proposed strategic solution process pushes towards an optimal solution, it may generate an infeasible BVS. The Pull Phase pulls the solution back to feasibility using pivoting rules similar to the dual simplex method. All phases use the usual Gauss pivoting row operation and it is shown to terminate successfully or indicates unboundedness or infeasibility of the problem. A computer imple-

Received: November 18, 2002

§Correspondence author

mentation, which further reduces the size of simplex tableau to the dimensions of the original decision variables, is provided. This enhances considerably the storage space, and the computational complexity of the proposed solution algorithm. A comparison analysis to test the efficiency of Push-and-Pull algorithm comparing to ordinary simplex is accomplished. Illustrative numerical examples are also presented.

AMS Subject Classification: 90C05, 90B03
Key Words: linear programming, basic variable set, artifical variable, advanced basis, simplex tableau reduction

1. Introduction

Since the creation of the simplex solution algorithm for linear programs (LP) problems in 1947 by Dantzig [7], this topic has enjoyed a considerable and growing interest by researchers and students of many fields. However, experience shows there are still computational difficulties in solving LP problems, in which some constraints are in ($\geq$) form with the right-hand side (RHS) non-negative, or in (=) form.

One version of the simplex known as the two-phase method introduces an artificial objective function, which is the sum of artificial variables, while the other version adds the penalty terms, which is the sum of artificial variables with very large positive coefficients. The latter approach is known as the Big-M method.

Most practical LP problems such as Transportation Problem [10], and the Finite Element Modeling [11] have many equality, and greater-than-or equal constraints. There have been some attempts to avoid the use of artificial variables in the context of simplex method. Arsham [3] uses a greedy method, which is applicable to small size problem. Paparrizos [12] introduced an algorithm to avoid artificial variables through a tedious evaluation of a series of extraneous objective functions besides the original objective function. At each iteration this algorithm must check both optimality and feasibility conditions simultaneously. An algebraic method is also introduced in [2], which is based on combinatorial method of finding all basic solutions even those, which are not feasible. Recently noniterative methods [8, 10], are proposed, which belong to heuristic family algorithms.

The aim of this paper is to test the efficiency of an algorithm to solve LP problems called Push-and-Pull, developed by Arsham [3], comparing to a well-known and widely used ordinary simplex. For this purpose, a comparison

analysis between the two mentioned algorithms has been accomplished. The result of this analysis shows that Push-and-Pull algorithm is more effective and faster than ordinary simplex.

Push-and-Pull is a new general solution algorithm, which is easy to understand, is free from any extraneous artificial variables, artificial objective function, extraneous objective functions, or artificial constraint. The algorithm consists of two phases. In Phase I a feasible segment of the boundary hyper– using rules similar to the ordinary simplex search plane (a face of feasible region or an intersection of several faces). Each successive iteration augments BVS (which is initially empty) by including another hyper-plane and/or shifting an existing one parallel to itself toward the feasible region (reducing its slack/surplus value), until the BVS specifies a feasible vertex. In this phase the movements are on faces of the feasible region rather than from a vertex to a vertices. This phase (if needed) is moving from this vertex to a vertex that satisfied the optimality condition. If the obtained vertex is not feasible, then the second phase generates an optimal solution (if exists), using the rule similar to the dual simplex method.

The inclusion of the ordinary simplex, and the dual simplex methods as part of the proposed algorithm unifies both methods by first augmenting the BVS, which is (partially) empty initially, rather than replacing variables.

The Push-and-Pull algorithm is motivated by the fact that, in the case of (=) and ($\geq$) constraints the simple method has to iterate through many infeasible verities to reach the optimal vertex. Moreover, it is well known that the initial basic solution by the simplex method could be far away from the optimal solution [13]. Therefore, the Push-and-Pull algorithm starts with a BVS, which is completely empty, then we fill it up with "good" variables, i.e. having large C_j, while maintaining feasibility. As a result, the simplex phase has a *warm-start*, which is a vertex in the neighborhood of optimal vertex.

The algorithm working space is the space of the original variables with a nice geometric interpretation of its strategic process. Our goal is to obviate the use of artificial variables, and unification with the ordinary simplex with the dual simplex in a most *natural and efficient* manner. The proposed approach has smaller tableau to work with since there are no artificial columns, penalty terms in the objective function, and also the row operations are performed directly on C_J's and there is no need for any extra computation to generate an extraneous row $C_j - Z_j$.

In Section 2, we present the strategic process of the new solution algorithm with the proof that it converges successfully if an optimal solution exists. In Section 3, we present a computer implementation of the algorithm with fur-

ther reduction in the computer storage, and the computational complexity by removing the slack/surplus variables. Applying the computer implementation technique to a numerical example is given in the next section. In Section 4, we present a comparison analysis between the Push-and-Pull and ordinary simplex algorithms using 15 examples. The last section contains some useful remarks.

2. The New Solution Algorithm

Consider any LP problem in the following standard form:

$$\text{Max} \quad \sum_{j=1}^{n} C_j X_j \,,$$

subject to

$$AX(\leq, =, \geq)\, b\,; \;\; X \;\geq\; 0 \;\text{ and }\; b \;\geq\; 0\,. \tag{1}$$

A is the respective matrix of constraint coefficients, and b is the respective RHS vectors (all with appropriate dimensions). This LP problem can be rewritten in the form

$$\text{Max } CX\,,$$

subject to

$$\sum_{j=1}^{n} X_j\, p^j = p^0\,; \quad X_j \geq 0\,. \tag{2}$$

Without loss of generality we assume all the RHS elements are non-negative. We will not deal with the trivial cases, such as when $A = 0$, (no constraints) or $b = 0$ (all boundaries pass through the origin point). The customary notation is used: C_j for the objective function coefficients (known as cost coefficients), and $X = \{X_j\}$ for the decision vector. Throughout this paper the word "constraint" means any constraint other than the non-negativity conditions. To arrive at this standard form some of the following preliminaries may be necessary.

Preliminaries: To start the algorithm the LP problem must be converted into the following standard form:

(a) The problem must be a maximization problem: If the problem is a minimization problem convert it to maximization by multiplying the objective function by -1. By this operation the optimal solution remains the same, then

use the original objective function to get the optimal value by substituting the optimal solution.

(b) RHS must be non-negative: Any negative RHS can be converted by multiplying it by -1 and changing its direction if it is an inequality, optimal solution remains unchanged. To avoid the occurrence of a possible degeneracy, convert all inequality constraints with RHS = 0 into 0 by multiplying each one of them by -1.

(c) All variables must be non-negative: Convert any *unrestricted variable* X_j to two non-negative variables by substituting $y - X_j$ for every X_j everywhere. This increases the dimensionality of the problem by one only (introduce one y variable) irrespective of how many variables are unrestricted. If there are any equality constraints, one may eliminate the *unrestricted variable (s)* by using these equality constraints. This reduces dimensionality in both number of constraints as well as number of variables. If there are no unrestricted variables do remove the equality constraints by substitution, this may create infeasibility. If any X_j variable is restricted to be non-positive, substitute $-X_j$ for X_j everywhere.

It is assumed that after adding slack and surplus variables to resource constraints (i.e.,$\leq$) and production constraints (i.e.,$\geq$) respectively, the matrix of coefficients is a full row rank matrix (having no redundant constraint) with m rows and n columns. Note that if all elements in any row in any simplex tableau are zero, we have one of two special cases. If the RHS element is non-zero then the problem is infeasible. If the RHS is zero this row represents a redundant constraint. Delete this row and proceed.

2.1. Strategic Process for the New Solution Algorithm

Solving LP problems in which some constraints are in ($\geq$) form, with the right-hand side (RHS) non-negative, or in (=) form, has been difficult since the beginning of LP (see, e.g. [6, 7, 14]). One version of the simplex method, known as the two-phase method, introduces an artificial objective function, which is the sum of the artificial variables. The other version is the Big-M method [6, 13], which adds a penalty term, which is the sum of artificial variables with very large positive coefficients. Using the dual simplex method has its own difficulties. For example, when some coefficients in the objective function are not dual feasible, one must introduce an artificial constraint. Also handling (=) constraints is very tedious.

The algorithm consists of preliminaries for setting up the initialization followed by two main phases: **Push** and **Pull** phases. The Push Phase develops

a basic variable set (BVS), which may or may not be feasible. Unlike simplex and dual simplex, this approach starts with an incomplete BVS initially, and then variables are brought into the basis one by one. If this process can not generate a complete BVS or the BVS is complete, but the optimality condition is not satisfied, then Push Phase pushes until this condition is satisfied. This strategy pushes towards an optimal solution. Since some solutions generated may be infeasible, the next step, if needed, the Pull Phase pulls the solution back to feasibility. The Push Phase satisfies the optimality condition, and the Pull Phase obtains a feasible and optimal basis. All phases use the usual Gauss pivoting row operation.

The initial tableau may be empty, partially empty, or contain a full basic variable set (BVS). The proposed scheme consists of the following two strategic phases:

Push Phase: Fill-up the BVS completely by pushing it toward the optimal corner point, while trying to maintain feasibility. If the BVS is complete, but the optimality condition is not satisfied, then push phase continues until this condition is satisfied.

Pull Phase: If pushed too far in Phase I, pull back toward the optimal corner point (if any). If the BVS is complete, and the optimality condition is satisfied but infeasible, then pull back to the optimal corner point, i.e., a dual simplex approach.

Not all LP problems must go through the Push and Pull sequence of steps, as shown in the numerical examples.

To start the algorithm the LP problem must be converted into the following standard form:

1. Must be a maximization problem.
2. RHS must be non-negative.
3. All variables must be non-negative.
4. Convert all inequality constraints (except the non-negativity conditions) into equalities by introducing slack/surplus variables.

The following two phases describe how the algorithm works. It terminates successfully for any type of LP problems since there is no loop-back between the phases.

The Push Phase:

Step 1. By introducing slack or surplus variables convert all inequalities (except non-negativity) constraints into equalities. The coefficient matrix must

have full row rank, since otherwise either no solution exists or there are redundant equations.

Step 2. Construct the initial tableau containing all slack variables as basic variables.

Step 3. Generate a complete basic variable set (BVS), not necessarily feasible, as follows:

1. Incoming variable is X_j with the largest C_j (the coefficient of X_j in the objective function, it could be negative).

2. Choose the smallest non-negative C/R if possible. If there are alternatives, break the ties arbitrarily.

3. If the smallest non-negative C/R is in already occupied BV row then choose the next largest X_j not used yet within the current iteration, and go to (1).
   If the BVS is complete or all possible incoming variables X_j have been already used then continue with step 4, otherwise generate the next tableau and go to (1).

Step 4. If all $C_j \leq 0$ then continue with Step 5.

1. Identify incoming variable (having largest positive C_j).

2. Identify outgoing variable (with the smallest non-negative C/R). If more than one, choose any one, this may be the sign of degeneracy. If Step 4 fails, then the solution is unbounded. However, to prevent a false sign for the unbound solution, introduce a new constraint $\sum X_i + S = M$ to the curent tableau, with S as a basic variable. Where M is an unspecified sufficiently, large positive number, and X_i's are variables with a positive C_j in the current tableau. Enter the variable with largest C_j and exit S. Generate the next tableau, (this makes all $C_j \leq 0$), then go to Step 5.

Step 5. If all $RHS \geq 0$ and all $C_j \leq 0$ in the current tableau then this is the optimal solution, find out all multiple solutions if they exist (the necessary condition is that the number of $C_j = 0$ is larger than the size of the BVS). If some $C_j > 0$, then go to Step 4, otherwise continue with the Pull phase.

The Pull Phase:

Step 6. Use the dual simplex pivoting rules to identify the outgoing variable (smallest RHS). Identify the incoming variable having negative coefficient in the

pivot row in the current tableau, if there are alternatives choose the one with the largest positive row ratio (R/R), (that is, a new row with elements: row C_j/pivot row); if otherwise generate the next tableau and go to Step 5. If Step 6 fails, then the problem is infeasible. Stop.

For numerical examples see [5].

2.2. The Theoretical Foundation of the Proposed Solution Algorithm

One of the advantages of this simplex-based method over another methods is that final tableau generated by these algorithms contains all of the information needed to perform the LP sensitivity analysis. Moreover, the proposed algorithm operates in the space of the original variables and has a geometric interpretation of its strategic process. The geometric interpretation of the algorithm is interesting when compared to the geometry behind the ordinary simplex method. The simplex method is a vertex-searching method. It starts at the origin, which is far away from the optimal solution. It then moves along the intersection of the boundary hyper-planes of the constraints, hopping from one vertex to the neighboring vertex, until an optimal vertex is reached in two phases. It requires adding artificial variables since it lacks feasibility at the origin. In the first phase, starting at the origin, the simplex hops from one vertex to the next vertex to reach a feasible one. Upon reaching a feasible vertex, i.e., upon removal of all artificial variables from the basis, the simplex moves along the edge of the feasible region to reach an optimal vertex, improving the objective value in the process. Hence, the first phase of simplex method tries to reach feasibility, and the second phase of simplex method strives for optimality. In contrast, the proposed algorithm strives to create a full basic variable set (BVS), i.e., the intersection of m constraint hyper-planes, which provides a vertex. The initialization phase provides the starting segment of a few intersecting hyper-planes and yields an initial BVS with some open rows. The algorithmic strategic process is to *arrive at* the feasible part of the boundary of the feasible region. In the Push Phase, the algorithm pushes towards an optimal vertex, unlike the simplex, which only strives, for a feasible vertex. Occupying an open row means arriving on the face of the hyper-plane of that constraint. The Push Phase iterations augment the BVS by bringing-in another hyper-plane in the current intersection. By restricting incoming variables to open rows only, this phase ensures movement in the space of intersection of hyper-planes selected in the initialization phase only until another hyper-plane is hit. Recall that no replacement of variables is done in this phase. At each iteration the dimension-

ally of the working region is reduced until the BVS is filled, indicating a vertex. This phase is free from pivotal degeneracy. The selection of an incoming variable with the largest C_j helps push toward an optimal vertex. As a result, the next phase starts with a vertex.

At the end of the Push-Further phase the BVS is complete, indicating a vertex which is in the neighborhood of an optimal vertex. If feasible, this is an optimal solution. If this basic solution is not feasible, it indicates that the push has been excessive. Note that, in contrast to the first phase of the simplex, this infeasible vertex is on the other side of the optimal vertex. Like the dual simplex, now the Pull Phase moves from vertex to vertex to retrieve feasibility while maintaining optimality; it is free from pivotal degeneracy since it removes any negative, non-zero RHS elements.

Theorem 1. *By following Steps 3(1) and 3(2) a complete BV set can always be generated which may not be feasible.*

Proof. Proof of the first part follows by contradiction from the fact that there are no redundant constraints. The second part indicates that by pushing toward the optimal corner point we may have passed it.
Note that if all elements in any row are zero, we have one of two special cases. If the RHS element is non-zero then the problem is infeasible. If the RHS is zero this row represents a redundant constraint. Delete this row and proceed. □

Theorem 2. *The Pull Phase and the initial part of the Push Phase are free from pivotal degeneracy that may cause cycling.*

Proof. It is well known that whenever a RHS element is zero in any simplex tableau (except the final tableau), the subsequent iteration may be pivotal degenerate when applying the ordinary simplex method, which may cause cycling. In the Push phase, we do not replace any variables. Rather, we expand the basic variable set (BVS) by bringing in new variables to the open rows marked with "?". The Pull Phase uses the customary dual simplex rule to determine what variable goes out. This phase is also free from pivotal degeneracy since its aim is to replace any negative, non- zero RHS entries. Unlike the usual simplex algorithms, the proposed solution algorithm is almost free from degeneracy. The initial phase brings variables in for the BVS, and the last phase uses the customary dual simplex, thus avoiding any degeneracy, which may produce cycling. When the BVS is complete, however, degeneracy may occur using the usual simplex rule. If such a rare case occurs, then the algorithm calls for Degeneracy Subroutine as described in the computer implementation section. □

Theorem 3. *The solution algorithm terminates successfully in a finite number of iterations.*

Proof. The proposed algorithm converges successfully since the path through the Push, Push-Further and Pull Phases does not contain any loops. Therefore, it suffices to show that each phase of the algorithm terminates successfully. The Set-up Phase uses the structure of the problem to fill-up the BVS as much as possible without requiring GJP iterations. The initial part of the Push Phase constructs a complete BVS. The number of iterations is finite since the size of the BVS is finite. Push Phase uses the usual simplex rule. At the end of this phase, a basic solution exists that may not be feasible. The Pull Phase terminates successfully by the well-known theory of dual simplex. □

3. Implementation and Reduction Algorithm

The description given up to now of the proposed solution algorithm aimed at clarifying the underlying strategies and concepts. A second important aspect that we consider now is efficient computer implementation and data structures.

Practical LP problems may have hundreds or thousands of constraints and even more variables. A large percentage of variables are either slack or surplus variables. Our proposed method as described is not efficient for computer solution of large-scale problems, since it carries slack/surplus variables of the tableau at each iteration. Some useful modifications of the proposed algorithm, as well as its efficient implementation will reduce the number of computations and the amount of computer memory needed. We are concerned with computational efficiency, storage, and accuracy, as well as ease of data entry. This is achieved by removing the need for storing the slack/surplus variables. This fact also means faster iterations.

3.1. The Main Algorithm

The Push Phase:

Step 1: State $A = [p^1, ..., p^n]$ in (2) as $A = [F, D, B]$, where F is the matrix of input vectors p^j, $j = 1, ..., s$, D is the matrix of slack vectors p^j, $j = s+1, ..., d$, and B is the (unity) identity matrix composed of basis vectors, i.e. slack vectors p^j, $j = d+1, ..., n$. For the elements d_{ij} of D we have $d_{ii} = -1$, and $d_{ij} = 0$, for $i = 1, ..., m$ and $j = s+1, ..., d$. This means that the column vectors composing D and B and the corresponding components c_j of the objective function are

prescribed in advance and we may confine ourselves to the LP solving process carried out only on matrix F. Namely, matrix B is only needed for defining the initial basis, which could be empty or partial empty. Regarding matrix D, at each iteration, if necessary we define one of its vectors, say p^y, transform it and store it in F to replace p^k that enters the basis. p^y may enter the basis at some later stage of the iteration process. From the computational point of view this means less computer storage for storing the simplex tableau and less computer time for its transformation. Therefore in the algorithm A, i.e. F, is defined as an $(m+1) \times s$ matrix. The $(m+1)$-th row is needed for storing of C.

Step 2:

(i) read m, s, $(c_j,\ j = 1, ..., s)$, $(p_i^0,\ i = 1, ..., m)$,
$((a_{ij},\ i = 1, ..., m),\ j = 1, ..., s)$, where($p_i^0$ is RHS).

(ii) Build a vector q of order m with components q_i = type of the i-th constraint of (1), i.e. "$\geq$", "=" or "$\leq$": read $(q_i,\ i = 1, 2, ..., m)$.

(iii) Compute the values of d:
$d = s$;
for $i = 1$ to m
    if ($q_i =$ " $\geq$ " or $q_i =$ " $\leq$") then
        $d = d + 1$;
    end if
end for

(iv) Create vectors v, c and a_{m+1} involving subscripts of the input and slack vectors within the simplex tableau, and the corresponding coefficients of the objective function, respectively:

for $i = 1$ to s
    $v_i = i$;
    $a_{m+1,i} = c_i$;
end for
$w = 0$;

for $j = s + 1$ to d
  $c_j = 0$;
  for $i = w + 1$ to m
    if($q_i =$ " $\leq$ " or $q_i =$ " $\geq$ ") then
      $v_j = j$;
      $w = i$;
      exit for
    end if
  end for
end for

(v) Create vectors v^B and c^B involving subscripts of the basic vectors within the simplex tableau and the corresponding coefficients of the objective function, respectively:
ii = 0;
for $i = 1$ to m
  $v_i^B = 0$;
  $c_i^B = 0$;
  if ($q_i =$ " $\leq$ ") then
    $ii = ii + 1$;
    $v_i^B = s + ii$;
    $v_{s+ii} = -i$;
  else if ($q_i =$ " $\geq$ ") then
    $ii = ii + 1$;
  end if
end for

Step 3:

Do while (v^B is not complete)

(1) Define k of vector p^k that enters the basis

a)
$$p^k = \max_{1 \leq j \leq s} a_{m+1,j}$$

where p^j is not in the basis and $a_{m+1,j}$ has not been chosen yet in the current iteration.

b)

for $i = 1$ to d
  if $(k = v_i)$ then
    $h = i$;
    exit for
  end if
end for
Subscript i of $v_i = k$ shows the real location of the pivot column [see, e.g., Ref.4] within A.

(2) Definition of p^r that leaves the existent basis
Subscript r defines the pivot row [see, e.g., Ref.4] within A.
Search for subscript r satisfying

$$\frac{p_r^0}{a_{rk}} = \min_{1 \le i \le m} \frac{p_i^0}{a_{ik}}, \quad \text{for} \quad \frac{p_r^0}{a_{rk}} > 0 .$$

Four possibilities occur:

(i) v_r^B is already occupied and not all $a_{m+1,j}$ have been chosen yet in the current iteration then go to 1 to choose the next largest $a_{m+1,j}$, otherwise exit Do while loop.

(ii) There exists one and only one value of r among i, where $1 \le i \le m$. Then $u = v_r^B$ shows the real location of p^r $(=p^u)$ that leaves the basis, and r shows the pivot row;
$u = v_r^B$
Include transformation procedure;
Include storing procedure.

(iii) There exists no $a_{ik} > 0$, for $i = 1, 2, ..., m$. In this case the problem is unbounded and the process is stopped.

(iv) There are more than one value of r, then the case of degeneracy occurs;
Choose any one.
Include Transformation Procedure.
Include Storing Procedure.

End do

Step 4:

Do while (there are $a_{m+1,j} > 0$; $j = 1, ..., s$)

(1) Define k of vector p^k that enters the basis

$$p^k = \max_{1 \le j \le s} a_{m+1,j} \ , \quad \text{for} \quad a_{m+1,j} > 0$$

Include Step 3 (1.b).

(2) Definition of p^r that leaves the existent basis

$$\frac{p_r^0}{a_{rk}} = \min_{1 \le i \le m} \frac{p_i^0}{a_{ik}} \ , \quad \text{for} \quad \frac{p_r^0}{a_{rk}} > 0 \ .$$

Three possibilities occur:

(i) There exists one and only one value of r among i, where $1 \le i \le m$. Then $u = v_r^B$ shows the real location of p^r $(=p^u)$ that leaves the basis, and r shows the pivot row;
$u = v_r^B$
Include transformation procedure;
Include storing procedure.

(ii) There exists no $a_{ik} > 0$, for $i = 1, 2, ..., m$. In this case the problem is unbounded and the process is stopped.

(iii) There are more than one value of r, then the case of degeneracy occurs;
Choose any one.
Include Transformation Procedure.
Include Storing Procedure.

End do

Step 5:

If all $p_i^0 \ge 0$, $i = 1, ..., m$, then the current solution is optimal. Otherwise, continue with Step 6.

The Pull Phase:

Step 6:

(1) Definition of value θ.

$$\theta = \min_{1 \le i \le m} p_i^0 \ , \quad \text{for} \quad p_i^0 < 0$$

(2) Definition of p^k that enters and p^r that leaves the basis.

$$\frac{p_r^0}{a_{rk}} = \max_{1 \le j \le s} \frac{a_{m+1,j}}{a_{ij}} \ ,$$

where $i = 1, ..., m$, $p_i^0 = \theta$, $a_{m+1,j} < 0$, and $\frac{a_{m+1,j}}{a_{ij}} > 0$.
Include Step 3: (1.b).
Two possibilities occur:

(i) There exists one and only one value of r.
$u = v_r^B$
Include transformation procedure;
Include storing procedure;
Go to Step 5.

(ii) There exists no $\frac{a_{m+1,j}}{a_{ij}} > 0$, where $j = 1, ..., s$, $i = 1, ..., m$, and $p_i^0 = \theta$; then the problem is infeasible. Stop.

3.2. Subroutine Iteration and Transformation

Transformation of vector p^0 and matrix A, i.e. vectors p^j, $j = 1, ..., s$.
Compute

(i) $p_r^0 = \frac{p_r^0}{a_{rk}}$

(ii) for $j = 1, ..., s$, $j \neq k$
$a_{rj} = \frac{a_{rj}}{a_{rk}}$

(iii) for $i = 1, ..., m$, $i \neq r$
$p_i^0 = p_i^0 - p_r^0 \cdot a_{ik}$
and
for $i = 1, ..., m+1$, $i \neq r$
$a_{ij} = a_{ij} - a_{rj} \cdot a_{ik}$, for $j = 1, ..., s$, $j \neq k$

Storing: Storing p^r into p^k in A for possible further use in the iteration process.

Set

$$v_r^B = h\ ; \qquad c_r^B = c_h\ ; \qquad v_h = -r\ ;$$

The value $-r$ in v_h shows column r with the basis vector p^h in it.

Now the exchange and replacement of column vectors in A is taking place.

At the beginning of the problem solving process, matrix A has the same structure as matrix F, i.e. it consists of vectors p^j corresponding to the input variables x_j, $j = 1, ..., s$. At each iteration some defined exchange and replacement of vectors of A occurs. This is done in order to attain the following objectives:

- to exploit the storage capacity within A taken by p^k, after p^k has been transformed into a form with $p_i^k = 0$, $i = 1, ..., m$, $i \neq k$, and $p_k^k = 1$
- to save the necessary characteristics of the slack vector p^r that are needed for its possible later use as a pivot column vector p^k
- to save the necessary characteristics of vector p^r that has left the basis.

The process of replacement of p^k by p^r is as follows:

(i) if ($q_r =$ " $\geq$ " and $u = 0$) then

COMMENT. In this case u-th row in the basis is not occupied. A slack vector p^{s+r} is built up to the position of an auxiliary vector p, then transformed in accordance with the particular iteration and stored in the k-th column of A.

a) Build up p^{s+r} within an auxiliary vector p, where $p_i = 0$, $i = 1, ..., m+1$, $i \neq r$, and $p_r = -1$.

b) Transform components of p accordingly: $p_r = \frac{p_r}{a_{rk}}$
$p_i = p_i - p_r \cdot a_{ik}$, $i = 1, ..., m+1$, and $i \neq r$.

c) Store p into the k-th column of A:
$a_{ik} = p_i$, $i = 1, ..., m+1$.

d) Mark up the location of p^{s+r} by storing value k into the $(r+s)$-th component of vector v:
$v_{r+s} = k$.

(ii) else if ($q_r =$ " $=$ " and $u = 0$) then

COMMENT. u-th row in the basis is not occupied and there is no slack vector to be stored in the k-th column of A
$a_{rk} = 1$, $a_{ik} = 0$, $i = 1, ..., m+1$, $i \neq r$.

(iii) else

COMMENT. In this case u shows the real location (column) of p^r ($=p^u$) that leaves the basis. p^u is a basis vector with its components $p_i^u = 0$, $i = 1, ..., m$, $i \neq r$, and $p_r^u = 1$. In addition, p^u can again enter the basis in later stages of the iteration process. Therefore it is transformed accordingly and saved in the location of p^k within A.

a) Build up p^u within an auxiliary vector p, where $p_i = 0$, $i = 1, ..., m+1$, $i \neq r$, and $p_r = 1$.

b) Transform components of p accordingly:
$p_r = \frac{p_r}{a_{rk}}$,
$p_i = p_i - p_r \cdot a_{ik}$, $i = 1, ..., m+1$, and $i \neq r$.

c) Store p in the k-th column of A:
$a_{ik} = p_i$, $i = 1, ..., m+1$.

d) Mark up the location of p^u by storing k in the u-th component of vector v:
$v_u = k$.

end if

4. Numerical Illustrative Example for the Computer Implementation

In this section we are applying the computer implementation technique to a numerical example.

Numerical Example 1

Min $X_1 + 3X_2 + 4X_3 + 10X_4$
subject to

$$\begin{aligned} X_1 \quad &+ X_3 + X_4 \geq 10 \\ X_2 \; &+ 2X_3 + 2X_4 \geq 25 \\ X_1 + 2X_2 \quad &+ X_4 \geq 20 \end{aligned}$$, where $X_j \geq 0$, $j = 1, ..., 4$.

Step 1

$$\begin{aligned} X_1 \quad &+ X_3 + X_4 - X_5 \quad = 10 \\ X_2 \; &+ 2X_3 + 2X_4 - X_6 \quad = 25 \\ X_1 + 2X_2 \quad &+ X_4 - X_7 = 20 \end{aligned}$$

Step 2

(i) Read the input data:
m (= 3), s (= 4), c, p^0 (= RHS), A

(ii) Read q, i.e., $q = [" \geq ", " \geq ", " \geq "]$

(iii) Compute
$d = 7$

(iv) Create
$v = [1, 2, 3, 4, 5, 6, 7]$
$c = [2, 6, 8, 5, 0, 0, 0]$
$a_{m+1} = [-1, -3, -4, -10]$

(v) Create
$v^B = [0, 0, 0]$
$c^B = [0, 0, 0]$

Step 3

(1) Definition of vector p^k that enters the basis

v^B	c^B	p^0		p^1	p^2	p^3	p^4
0	0	10	A	1	0	1	1
0	0	25		0	1	2	2
0	0	20		1	2	0	1
				-1	-3	-4	-10

v	1	2	3	4	5	6	7
c	-1	-3	-4	-10	0	0	0

$k = 1$, $h = 1$

(2) Definition of vector p^r that leaves the basis

$$10/1, \quad r = 1, \quad u = v_1^B = 0$$

Transformation of p^0 and p^j, $j = 1, ..., s$

	p^0		p^1	p^2	p^3	p^4
p^1	10	A	1	0	1	1
0	25		0	1	2	2
0	10		0	2	-1	0
			-1	-3	-3	-9

Storing p^r and p^k

Set

$v_1^B = 1$
$c_1^B = -1$
$v_1 = -1$

Since $q_1 =$ " $\geq$ " and $u = 0$, the u-th row in the basis is not occupied. A slack vector $p^{r+s} = p^5$ is created, transformed and stored in the k-th (=1) column of A.

Iteration 2

Step 3

(1) Definition of vector p^k that enters the basis

v^B	c^B
1	-1
0	0
0	0

p^0
10
25
10

A

p^5	p^2	p^3	p^4
-1	0	1	1
0	1	2	2
1	2	-1	0
-1	-3	-3	-9

v	-1	2	3	4	1	6	7
c	-1	-3	-4	-10	0	0	0

$k = 1$, $h = 5$

(2) Definition of vector p^r that leaves the basis

$$10/1, \quad r = 3, \quad u = v_3^B = 0$$

Transformation of p^0 and p^j, $j = 1, ..., s$

	p^0
p^1	20
0	25
p^5	10

A

p^5	p^2	p^3	p^4
0	2	0	1
0	1	2	2
1	2	-1	0
0	-1	-4	-9

Storing p^r and p^k

$v_3^B = 5$

$c_3^B = 0$

$v_5 = -3$

Since $q_3 =$ " $\geq$ " and $u = 0$, the u-th row in the basis is not occupied. A slack vector $p^{r+s} = p^7$ is created, transformed and stored in the k-th (=1) column of A.

Iteration 3

Step 3

(1) Defininion of vector p^k that enters the basis

v^B	c^B
1	-1
0	0
5	0

p^0
20
25
10

A

p^7	p^2	p^3	p^4
-1	2	0	1
0	1	2	2
-1	2	-1	0
-1	-1	-4	-9

v	-1	2	3	4	-3	6	1
c	-1	-3	-4	-10	0	0	0

$k = 3$, $h = 3$

(2) Definition of vector p^r that leaves the basis

$$25/2, \quad r = 2, \quad u = v_2^B = 0$$

Transformation of p^0 and p^j, $j = 1, ..., s$

	p^0
p^1	20
p^3	12.5
p^5	22.5

A

p^7	p^2	p^3	p^4
-1	2	0	1
0	0.5	1	1
-1	2.5	0	1
-1	1	0	-5

Storing p^r and p^k

$v_2^B = 3$

$c_2^B = -4$

$v_3 = -2$

Since $q_2 =$ " $\geq$ " and $u = 0$, the u-th row in the basis is not occupied. A slack vector $p^{r+s} = p^6$ is created, transformed and stored in the k-th (=3) column of A.

Iteration 4

Step 3

v^B	c^B	p^0
1	-1	20
3	-4	12.5
5	0	22.5

A

p^7	p^2	p^6	p^4
-1	2	0	1
0	0.5	-0.5	1
-1	2.5	-0.5	1
-1	1	-2	-5

v	-1	2	-2	4	-3	3	1
c	-1	-3	-4	-10	0	0	0

The basis is completed. Therefore continue with Step 4.

Step 4

(1) Definition of vector p^k that enters the basis
$k = 2$, $h = 2$

(2) Definition of vector p^r that leaves the basis

$$22.5/2.5, \quad r = 3, \quad u = v_3^B = 5$$

Transformation of p^0 and p^j, $j = 1, ..., s$

	p^0
p^1	2
p^3	8
p^2	9

A

p^7	p^2	p^6	p^4
-0.2	0	0.4	0.2
0.2	0	-0.4	0.8
-0.4	1	-0.2	0.4
-0.6	0	-1.8	-5.4

Storing p^r and p^k

$v_3^B = 2$

$c_3^B = -3$

$v_2 = -3$

Since $q_3 =$ " $\geq$ " and $u = 5$, a vector $p^u (= p^5)$ is created, transformed and stored in the k-th (=2) column of A.

v^B	c^B
1	2
3	8
2	9

	p^7	p^5	p^6	p^4
A	-0.2	-0.8	0.4	0.2
	0.2	-0.2	-0.4	0.8
	-0.4	0.4	-0.2	0.4
	-0.6	-0.4	-1.8	-5.4

v	-1	-3	-2	4	2	3	1
c	-1	-3	-4	-10	0	0	0

All RHS are non-negative. This solution is optimal.

5. Comparison Analysis

In this section we deal with determining the efficiency of the Push-and-Pull algorithm comparing to the ordinary simplex. For this purpose both algorithms were tested using the following 15 examples:

Example 01

$$\begin{aligned} \text{Max} \quad & 2X_1 + 6X_2 + 8X_3 + 5X_4 \\ \text{subject to} \quad & \\ & 4X_1 + X_2 + 2X_3 + 2X_4 \geq 80 \\ & 2X_1 + 5X_2 + 4X_4 \leq 40 \\ & 2X_2 + 4X_3 + X_4 = 120 \end{aligned}$$

Example 02

$$\begin{aligned} \text{Max} \quad & 2X_1 + 3X_2 - 6X_3 \\ \text{subject to} \quad & \\ & X_1 + 3X_2 + 2X_3 \leq 3 \\ & -2X_2 + X_3 \leq 1 \\ & X_1 - X_2 + X_3 = 2 \\ & 2X_1 + 2X_2 - 3X_3 = 0 \end{aligned}$$

Example 03

$$\begin{aligned} \text{Max} \quad & 20X_1 + 10X_2 + 40X_3 + 20X_4 + 15X_5 \\ \text{subject to} \quad & \\ & X_1 + 4X_3 + 2X_4 + 4X_5 \leq 120 \\ & 2X_1 + 5X_2 + 2X_3 + X_4 \leq 80 \\ & 4X_1 + X_2 + 5X_3 + 4X_5 \leq 240 \end{aligned}$$

Example 04

$$\begin{aligned} \text{Min} \quad & X_1 + 3X_2 + 4X_3 + 10X_4 \\ \text{subject to} \quad & \\ & X_1 + X_3 + X_4 \geq 10 \\ & X_2 + 2X_3 + 2X_4 \geq 25 \\ & X_1 + 2X_2 + X_4 \geq 20 \end{aligned}$$

Example 05

$$\begin{aligned} \text{Max} \quad & 100X_1 + 10X_2 + X_3 \\ \text{subject to} \quad & \\ & X_1 \leq 1 \\ & 20X_1 + X_2 \leq 100 \\ & 200X_1 + 20X_2 + X_3 \leq 100\ 000 \end{aligned}$$

Example 06

$$\begin{aligned} \text{Min} \quad & 8X_1 + 2X_2 + 5X_3 + 2X_4 + X_5 + 3X_6 + 7X_7 + 2X_8 + 6X_9 \\ \text{subject to} \quad & \\ & X_1 + X_2 + X_3 = 90 \\ & X_4 + X_5 + X_6 = 105 \\ & X_7 + X_8 + X_9 = 105 \\ & X_1 + X_4 + X_7 = 150 \\ & X_2 + X_5 + X_8 = 75 \\ & X_3 + X_6 + X_9 = 75 \end{aligned}$$

Example 07

$$\begin{aligned} \text{Max} \quad & 60X_1 + 70X_2 + 75X_3 \\ \text{subject to} \quad & \\ & 3X_1 + 6X_2 + 4X_3 \leq 2400 \\ & 5X_1 + 6X_2 + 7X_3 \leq 3600 \\ & 3X_1 + 4X_2 + 5X_3 \leq 2600 \end{aligned}$$

Example 08

$$\begin{aligned} \text{Max} \quad & 2X_1 + 2.5X_2 \\ \text{subject to} \quad & \\ & 3X_1 + 4X_2 \leq 20000 \\ & 0.75X_1 + 0.65X_2 \leq 4000 \end{aligned}$$

Example 09

$$\begin{aligned} \text{Max} \quad & 12X_1 + 18X_2 \\ \text{subject to} \quad & \\ & 2X_1 + 3X_2 \leq 33 \\ & X_1 + X_2 \leq 15 \\ & X_1 + 3X_2 \leq 27 \end{aligned}$$

Example 10

$$\begin{aligned} \text{Min} \quad & 6X_1 - 3X_2 - 4X_3 \\ \text{subject to} \quad & \\ & -2X_1 + X_2 \leq 0 \\ & -3X_1 + X_2 + X_3 \geq 0 \\ & X_2 + X_3 = 24 \end{aligned}$$

Example 11

$$\begin{aligned} \text{Max} \quad & 2X_1 + X_2 \\ \text{subject to} \quad & \\ & X_1 + 3X_2 \leq 12 \\ & X_1 + 2X_2 \leq 10 \\ & 2X_1 + 5X_2 \leq 30 \end{aligned}$$

Example 12

$$\begin{aligned} \text{Min} \quad & 4X_1 - 2X_2 + X_3 \\ \text{subject to} \quad & \\ & 2X_1 - X_2 + X_3 \leq 30 \\ & 3X_1 + X_2 - X_3 \leq 26 \\ & X_2 + X_3 \leq 13 \end{aligned}$$

Example 13

$$\begin{aligned} \text{Max} \quad & 12X_1 + 6X_2 + 4X_3 \\ \text{subject to} \quad & \\ & X_1 + 2X_2 \leq 6 \\ & -X_1 - X_2 + 2X_3 \leq 4 \\ & X_2 + X_3 \leq 2 \end{aligned}$$

Example 14

$$\begin{aligned} \text{Max} \quad & -X_1 + 21X_2 + 24X_3 + X_4 + 12X_5 \\ \text{subject to} \quad & \\ & 0.5X_1 + 6X_2 + 7X_3 - 2X_4 + X_5 \leq 64 \\ & -X_1 + 2X_2 + 4X_3 + 2X_4 + 5X_5 \leq 27 \\ & 2X_1 + 3X_2 - X_3 - 4X_4 + 3X_5 \leq 17 \\ & 3X_1 + 2X_2 + 4X_3 - X_4 - X_5 \leq 21 \end{aligned}$$

Example 15

$$\begin{aligned} \text{Min} \quad & 3X_1 + X_2 - X_3 \\ \text{subject to} \quad & \\ & 2X_1 + X_2 - X_3 = 10 \\ & -X_1 + X_3 \leq 6 \\ & 3X_1 - 4X_3 \leq 8 \end{aligned}$$

In our comparison analysis we used 3 parameters; these are: number of iterations, number of Add/Sub operations, and number of Mul/Div operations.

Table 1 shows the results of the analysis. Concerning the first parameter, number of iterations, the reader may see that Push-and-Pull algorithm needed less iterations to achieve the optimal solutions in 3 examples; these are 2,4 and 15. In all other examples both algorithms needed the same number of iterations to achieve optimal solutions. Concerning the second and third parameters, arithmetic operations, we can conclude that Push-and-Pull is far more effective than ordinary simplex. Push-and-Pull needed approximately only half of the number of arithmetic operations needed by ordinary simplex to achieve the optimal solutions.

Example	Push-Pull			Simplex		
	Iteration	Add/Sub	Mul/Div	Iteration	Add/Sub	Mul/Div
1	4	72	88	4	154	150
2	1	27	23	2	96	85
3	2	56	58	2	113	105
4	4	78	95	5	196	183
5	7	109	128	7	207	204
6	6	412	458	6	800	792
7	2	42	41	2	80	72
8	2	24	25	2	44	40
9	2	34	34	2	65	57
10	3	44	52	3	103	93
11	1	24	19	1	44	35
12	1	26	24	1	53	40
13	2	41	37	2	78	70
14	4	122	136	4	249	239
15	1	20	18	4	126	118

Table 1: Results of comparison analysis

6. Concluding Remarks

LP problems, which lack feasibility at the origin point, have raised difficulties since the beginning of LP. For such problems, the ordinary simplex algorithm *requires* additional variables (i.e., artificial variables) along with large penalty terms in the objective function. Unfortunately, adding these extraneous variables creates computational instability [8, 10] and increases the computational complexity [11]. Moreover, classroom experience shows that some students, particularly non-mathematical majors, have difficulty in understanding the intuitive notion of such requirement [3].

We proposed a new general solution algorithm, which is easy to understand, and it is free from any extraneous variables. The algorithm consists of two phases. The Push Phase uses the rules similar to the ordinary simplex in generating a *basic variable set* (BVS). The Pull Phase (if needed) generates a BVS, which is the optimal (if it exists). The algorithm working space is the space of the original variables with a geometric interpretation of its strategic process.

The proposed solution algorithm has the following additional features:

1- The algorithm's working space is the space of the original variables. The tableau is simpler since the row operations are performed directly on C_j's. Additionally, there is no need for the reduced cost row (known as $C_j - Z_j$). The proposed algorithm has simplicity and potential for wide *adaptation by instructors*. The entire variables are the *natural* parts of the LP problem: decision variables (X_j), and the slack/surplus variable (S_i) for resource/production types of constraints respectively, with significant *managerial and economical meanings and applications*.

2- The proposed solution algorithm is a general-purpose method for solving any LP type problems. More specialized methods exist that can be used for solving LP problems with specific structure. The new algorithm is most suitable for LP problems with a large number of equality (=) and/or production constraints ($\geq$), such as network models with lower bound on their arc capacities.

3- It is well known that the initial basic solution by the primal simplex could be far away from the optimal solution [13]. The BVS augmentation concept, which pushes toward *faces* of the feasible region, which may contain an optimal rather than jumping from a vertex to an improved adjacent one to provide a "warm-start".

4- Its computer implementation leads to further reduction of simplex tableau. This means a smaller amount of computer storage is needed to store the tableau and also a faster iteration. This fact could be very important particularly in solving large problems of linear programming.

A comparative study has been done, which compares Push-and-Pull to ordinary simplex. The result of this study shows that the new proposed aklgorithm is better and faster than ordinary simplex. At this stage, a convincing evaluation for the reader would be the application of this approach to a problem he/she solved by any other methods.

Acknowledgment

We are grateful for the referees' useful comment and suggestions. The National Science Foundation Grant CCR-9505732 supported this work.

References

[1] D.H. Ackeley, G.E. Hilton, T.J. Sejnovski, A learning algorithm for Boizmann machine, *Cognitive Science*, **62** (1985), 147-169.

[2] H. Arsham, Affine geometric method for linear programs, *Journal of Scientific Computing*, **12**, No. 3 (1997), 289-303.

[3] H. Arsham, Initialization of the simplex algorithm: An artifical-free approach, *SIAM Rewiew*, **39**, No. 4 (1997), 736-744.

[4] H. Arsham, Distribution-routes stability analysis of the transportation problem, *Optimization*, **43**, No. 1 (1998) 47-72.

[5] H. Arsham, T. Damij, J. Grad, An algorithm for simplex tableau reduction: the push-to-pull solution strategy, em Applied Mathematics and Computation, **137** (2003), 525-547.

[6] V. Chvatal, *Linear Programming*, Freeman and Co., New York (1993).

[7] G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, N.J. (1968).

[8] T. Gao, T. Li, J. Verschelde, M. Wu, *Balancing the lifting values to improve the numerical stability of polyhedral homotopy continuation methods*, *Applied Mathematics and Computation*, **114**, No. 2-3 (2000), 233-247.

[9] D.O. Hebb, *Organization of Behavior*, Wiley, New York (1949).

[10] V. Lakshmikantham, S. Sen, M. Jain, A. Ramful, $O(n^3)$ noniterative heuristic algorithm for linear programs with error-free implementation, *Applied Mathematics and Computation*, **110**, No. 1 (2000), 53-81.

[11] G. Mohr, Finite element modelling of distribution problems, *Applied Mathematics and Computation*, **105**, No. 1 (1999), 69-76.

[12] K. Papparrizos, The two-phase simplex without artificial variables, *Methods of Operations Research*, **61**, No. 1 (1990), 73-83.

[13] A. Ravindran, A comparison of the primal-simplex and complementary pivot methods for linear programming, *Naval Research Logistic Quarterly*, **20**, No. 1 (1973), 95-100.

[14] H. Taha, *Operations Research: An Introduction*, Macmillan, New York (1992).