A Computer Implementation of the Push-and-Pull Algorithm and Its Computational Comparison With LP Simplex method

H. Arsham[a], G. Cimperman[b], N. Damij[c], T. Damij[c], and J. Grad[d]

[a]University of Baltimore, MIS Division, Baltimore, Maryland 21201, USA
harsham@ubmail.ubalt.edu

[b] "Petrol Company ", 1000 Ljubljana, Slovenia

[c]University of Ljubljana, Faculty of Economics, 1000 Ljubljana, Slovenia
talib.damij@uni-lj.si

[d]University of Ljubljana, School of Public Administration, 1000 Ljubljana, Slovenia
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 implementation, which enables the user to run either Push-and-Pull or ordinary simplex algorithms, is provided. The fully coded version of the algorithm is available from the authors upon request. 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 Subj. Classification:
90C05: Linear programming
90B: Operations research and management science
Key Words and Phrases: 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 [8], 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 [12], and the Finite Element modeling [13] 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 [14] 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 [9, 12], 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 [15]. 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 $\underline{\text{extraneous}}$ 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 system, which enables the user to solve linear programming problems using either Push-and-Pull or ordinary simplex algorithms. The system provides a detailed analysis for each tested problem and also a comparison analysis if both algorithms are used. Applying the computer implementation system to a numerical example is given in the next section. The fully coded version of the algorithm is available from the authors upon request. In Section 5, 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:

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

subject to

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

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

$$Max\, CX$$

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 constraints 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.,[7, 8, 16]). 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 [7, 15] 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 [6].

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 dimensionally 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.

The fully coded version of the algorithm is available from the authors upon request.

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. Computer implementation system

In this section we represent a computer implementation system called Six-Pap, which is developed to enable the reader to run one or both algorithms, Push-and-Pull and ordinary simplex, for solving different linear programming problems. The fully coded version of the algorithm is available from the authors upon request. The system also creates a comparison analysis between the two algorithms. The system was designed and built with MS Visual Studio 6.0 using MS Visual Basic.

The system is object oriented, its functionality is based on the object structure, that consists of main functional objects, accompanied by useful supporting objects. The system is event-driven: the objects mostly communicate amongst themselves as well as the graphical user interface elements (user controls); thus by raising and watching for certain events. Objects and their structure are explained in detail in Section 4.

User can communicate with the system through a graphical user interface. It is a multi window interface consisting of three main windows with their own functional purpose and elements. Most of elements are User Controls: controls designed to fit the exact problem or deliver the exact functionality. That enhances the usability of the graphical user interface, and accompanied with appliance of error management using error handling routines, makes the system more stable and robust. User interface is explained in detail in Section 5.

There is an on-line help integrated in the system, which can be easily accessed and can provide useful information to the user about the system and how to use it.

3.1 Data organization and managenent

Data is organized in three groups: definition data, result summary data and result detailed data.

The definition and result summary data for each linear programming problem are stored in one plain text file. The first part of the file contains general data of the problem such as name, source and some comments. The second part contains data of the objective function, number of constraints, number of variables, equations and inequations and right hand side values.

The third and/or fourth parts of the file are created when one or both algorithms were executed. They represent the result summary data. The data consists of the algorithm's name, the status of the result, optimal value of objective function (if exists), basic variable set BVS values, objective function variables $x(j)$ values, degeneracy indicator, different efficiency counters (number of iterations, additions / subtractions, multiplications / divisions, loops and decisions), time stamp and the result detailed file name.

Detailed result shows a step-by-step algorithm execution by recording relevant data like basic variable sets, tableaus, outgoing, incoming variables and comments explaining the previous and/or next action for each iteration.

All data files and data operations are handled by a subsystem called the Repository. User communicates with the Repository subsystem through the graphical user interface, especially through the Repository window which is explained in detailed later. That means that users need not worrying about data handling operations, data structure or file locations - the Repository subsystem does all that, including the creation of all required subfolders if they do not exist already.

3.2 Implementation of algorithms

The two algorithms are implemented in separate modules (MS Visual Studio, MS Visual Basic). There are actually three modules. The first two main modules implement iterative loops, one for Push-and-Pull algorithm and the other for ordinary simplex algorithm. The third is a supporting module with Gauss pivoting row operations.

The system uses algorithm implementation modules by calling a single module procedure (subroutine), which acts as the module entry point. The parameter passed is a reference to the problem object, where the results are stored and returned to the system.

The advantage of the modular implementation lies in the possibility of using the DLL (Dynamic Link Library) technology. With little system redesign, user could write their own implementations and compile them to a DLL. That would enable them to test, analyze and use their own implementations without needing to apply any changes to the SixPap system.

3.3 Object structure

3.3.1. Main objects

A group of main objects is used to handle linear programming problems as a whole, from problem definition to the results given by each (Push-and-Pull as well as ordinary simplex) algorithm. Classes representing those objects are: Problem, Problem Definition and Problem Result.

The Problem class is a parent class for Problem Definition and Problem Result classes and represents a single problem on a higher level. It consists of methods that support operations like creating new problem object structure and its initiation, methods for working with reading and writing data from files as well as methods that cause different ways of displaying a problem. The class through its properties holds information about a problem object structure, by maintaining references to child objects, and links to user interface, by maintaining references to user controls that handle different ways of displaying a problem data.

The Problem Definition class is a child of the Problem Class. It handles problem definition data in detail. Its methods support operations like reading and writing the definition-part of problem data to file (in collaboration with its parent objects r/w methods) as well as definition data editing with data validation. Contained information consists of values defining a linear problem.

The Problem Result class is also a child of the Problem Class. It has two instances, for each linear programming algorithm one. It handles problem result data in detail through methods that support read/write operations as well as different display operations for the result-part data. The information the class keeps consists of algorithm result data - both summary and detailed.

3.3.2. Supporting Objects

Functionality of the system is enhanced by useful supporting objects such as various counters, sorted lists and result detailed file handlers represented by Counters, Output Detail File, Sorted List and Item in Sorted List Class.

The Counters Class is a part of each Result Class. Its function is to handle and store analytic-purpose counters: iteration, addition / subtraction, multiplication / division, loop and decision count.

The Output Detail File Class is a part of each Result Class. Its function is to handle write-to-file operations during the linear programming algorithm run. Output is a result detailed file.

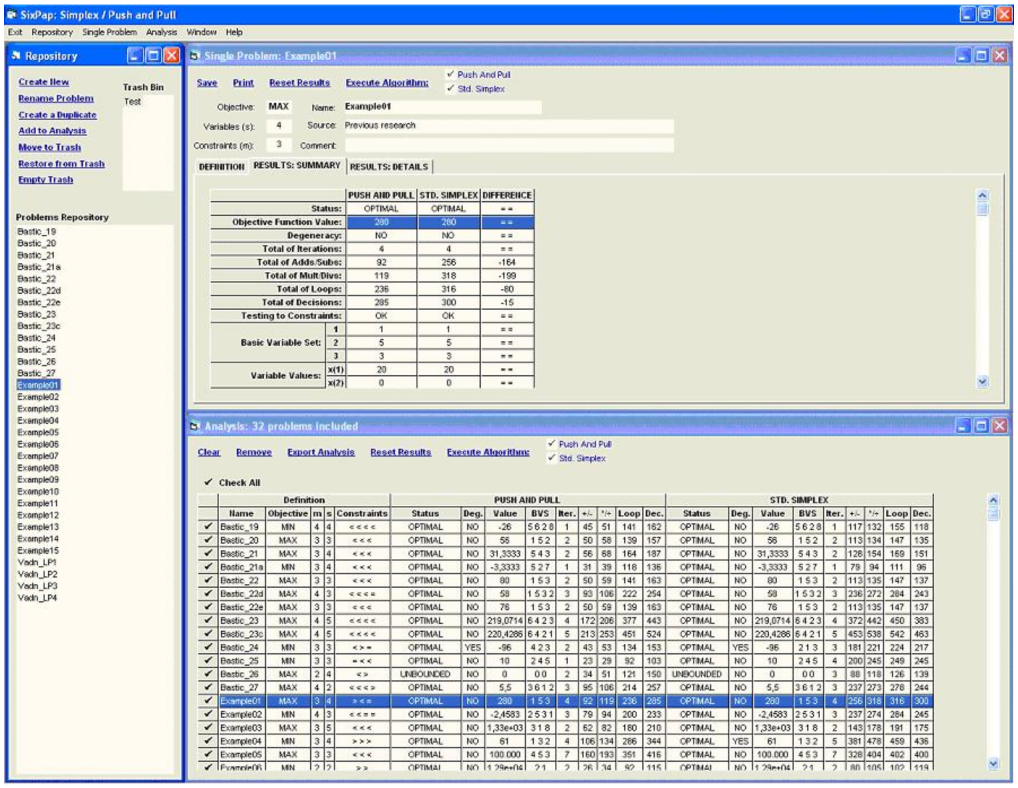
The Sorted List Class is involved with Push-and-Pull algorithm execution. It is a simple class representing a sorted list of elements. Its role in the System is to maintain and handle the C_j values as a sorted list. The elements of the list consist of the Item in Sorted List Class objects.

The Item in Sorted List Class is a part of the Sorted List Class. It represents a single element with its value and reference to the next element in the list.

3.4 User interface

The SixPap User Interface is a multi-document interface, using an MDI Form (the Main Window) as a container for three functional windows: the Repository Window, the Single Problem Window and the Analysis Window, as can be seen on Figure 1. All the features, mentioned in the following text, are explained in detail in the SixPap on-line help file.

Figure 1: The SixPap User Interface



SixPap: Simplex / Push and Pull

Exit Repository Single Problem Analysis Window Help

Repository

Create New
Rename Problem
Create a Duplicate
Add to Analysis
Move to Trash
Restore from Trash
Empty Trash

Trash Bin
Test

Problems Repository
Bastic_19
Bastic_20
Bastic_21
Bastic_21a
Bastic_22
Bastic_22d
Bastic_22e
Bastic_23
Bastic_23c
Bastic_24
Bastic_25
Bastic_26
Bastic_27
Example01
Example02
Example03
Example04
Example05
Example06
Example07
Example08
Example09
Example10
Example11
Example12
Example13
Example14
Example15
Vadn_LP1
Vadn_LP2
Vadn_LP3
Vadn_LP4

Single Problem: Example01

Save Print Reset Results Execute Algorithm: ✓ Push And Pull ✓ Std. Simplex

Objective: MAX Name: Example01
Variables (s): 4 Source: Previous research
Constraints (m): 3 Comment:

DEFINITION | RESULTS: SUMMARY | RESULTS: DETAILS

		PUSH AND PULL	STD. SIMPLEX	DIFFERENCE
Status:		OPTIMAL	OPTIMAL	= =
Objective Function Value:		280	280	= =
Degeneracy:		NO	NO	= =
Total of Iterations:		4	4	= =
Total of Adds/Subs:		92	256	-164
Total of Mult/Divs:		119	318	-199
Total of Loops:		236	316	-80
Total of Decisions:		285	300	-15
Testing to Constraints:		OK	OK	= =
Basic Variable Set:	1	1	1	= =
	2	5	5	= =
	3	3	3	= =
Variable Values:	x(1)	20	20	= =
	x(2)	0	0	= =

Analysis: 32 problems included

Clear Remove Export Analysis Reset Results Execute Algorithm: ✓ Push And Pull ✓ Std. Simplex

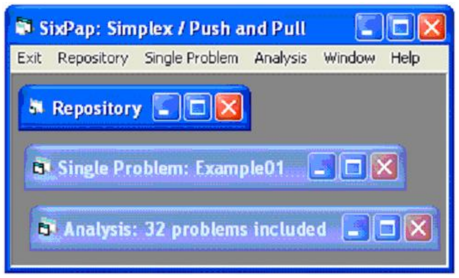
✓ Check All

	Definition					PUSH AND PULL									STD. SIMPLEX								
	Name	Objective	m	s	Constraints	Status	Deg.	Value	BVS	Iter.	+/-	*/+	Loop	Dec.	Status	Deg.	Value	BVS	Iter.	+/-	*/+	Loop	Dec.
✓	Bastic_19	MIN	4	4	< < < <	OPTIMAL	NO	-26	5 6 2 8	1	45	51	141	162	OPTIMAL	NO	-26	5 6 2 8	1	117	132	155	118
✓	Bastic_20	MAX	3	3	< < <	OPTIMAL	NO	56	1 5 2	2	50	58	139	157	OPTIMAL	NO	56	1 5 2	2	113	134	147	135
✓	Bastic_21	MAX	3	4	< < <	OPTIMAL	NO	31,3333	5 4 3	2	56	68	164	187	OPTIMAL	NO	31,3333	5 4 3	2	128	154	169	151
✓	Bastic_21a	MIN	3	4	< < <	OPTIMAL	NO	-3,3333	5 2 7	1	31	39	118	136	OPTIMAL	NO	-3,3333	5 2 7	1	79	94	111	96
✓	Bastic_22	MAX	3	3	< < <	OPTIMAL	NO	80	1 5 3	2	50	59	141	163	OPTIMAL	NO	80	1 5 3	2	113	135	147	137
✓	Bastic_22d	MAX	4	3	< < < =	OPTIMAL	NO	58	1 5 3 2	3	93	106	222	254	OPTIMAL	NO	58	1 5 3 2	3	236	272	284	243
✓	Bastic_22e	MAX	3	3	< < <	OPTIMAL	NO	76	1 5 3	2	50	59	139	163	OPTIMAL	NO	76	1 5 3	2	113	135	147	137
✓	Bastic_23	MAX	4	5	< < < <	OPTIMAL	NO	219,0714	6 4 2 3	4	172	206	377	443	OPTIMAL	NO	219,0714	6 4 2 3	4	372	442	450	383
✓	Bastic_23c	MAX	4	5	< < < <	OPTIMAL	NO	220,4286	6 4 2 1	5	213	253	451	524	OPTIMAL	NO	220,4286	6 4 2 1	5	453	538	542	463
✓	Bastic_24	MIN	3	3	< > =	OPTIMAL	YES	-96	4 2 3	2	43	53	134	153	OPTIMAL	YES	-96	2 1 3	3	181	221	224	217
✓	Bastic_25	MIN	3	3	= < <	OPTIMAL	NO	10	2 4 5	1	23	29	92	103	OPTIMAL	NO	10	2 4 5	4	200	245	249	245
✓	Bastic_26	MAX	2	4	< >	UNBOUNDED	NO	0	0 0	2	34	51	121	150	UNBOUNDED	NO	0	0 0	3	88	118	126	139
✓	Bastic_27	MAX	4	2	< < < >	OPTIMAL	NO	5,5	3 6 1 2	3	95	106	214	257	OPTIMAL	NO	5,5	3 6 1 2	3	237	273	278	244
✓	Example01	MAX	3	4	> < =	OPTIMAL	NO	280	1 5 3	4	92	119	236	285	OPTIMAL	NO	280	1 5 3	4	256	318	316	300
✓	Example02	MIN	4	3	< < = =	OPTIMAL	NO	-2,4583	2 5 3 1	3	79	94	200	233	OPTIMAL	NO	-2,4583	2 5 3 1	3	237	274	284	245
✓	Example03	MAX	3	5	< < <	OPTIMAL	NO	1,33e+03	3 1 8	2	62	82	180	210	OPTIMAL	NO	1,33e+03	3 1 8	2	143	178	191	175
✓	Example04	MIN	3	4	> > >	OPTIMAL	NO	61	1 3 2	4	106	134	286	344	OPTIMAL	YES	61	1 3 2	5	381	478	459	436
✓	Example05	MAX	3	3	< < <	OPTIMAL	NO	100.000	4 5 3	7	160	193	351	416	OPTIMAL	NO	100.000	4 5 3	7	328	404	402	400
✓	Example06	MIN	2	2	> >	OPTIMAL	NO	1,29e+04	2 1	2	26	34	92	115	OPTIMAL	NO	1,29e+04	2 1	2	80	105	102	119

3.4.1. Main Window

Figure 2 shows the Main Window with its elements.

Figure 2: The Main Window


SixPap: Simplex / Push and Pull
Exit Repository Single Problem Analysis Window Help
Repository
Single Problem: Example01
Analysis: 32 problems included

3.4.2. Repository Window

The Repository Window serves as an interface between the user and the Repository sub-system, which, as mentioned above, handles all data storage operations. It has a problem-level approach, meaning, that the main tasks concern a problem as a whole. Those tasks are selecting, opening, creating new, renaming, duplicating, adding to analysis, and deleting problems.

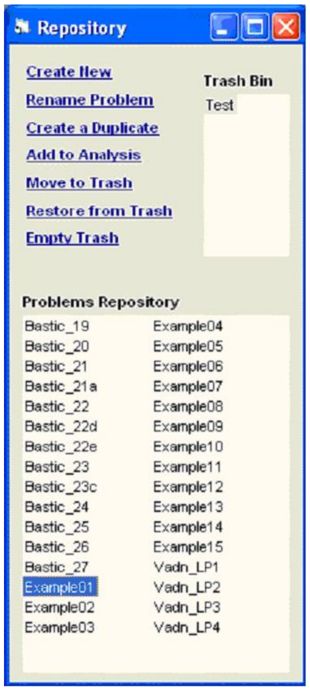
As is shown in Figure 3, the Repository Window, beside a standard window bar, consists of three main elements: the Repository Menu, the Trash Bin and the Problems Repository List.

The Repository Menu is a MyActiveLabels user-control with menu options that execute the Repository tasks.

The Trash Bin is a MyFileListView user control. It acts as a container for deleted problems. With some Repository Menu options they can become permanently deleted or can be restored to the Problem Repository.

The Problem Repository is a MyFileListView user control, too. It is the main Repository display element, showing the collection of existing problems. With other Repository Window elements it takes care of the execution of the Repository sub-system tasks.

Figure 3: The Repository Window



Bastic_19 Example04
Bastic_20 Example05
Bastic_21 Example06
Bastic_21a Example07
Bastic_22 Example08
Bastic_22d Example09
Bastic_22e Example10
Bastic_23 Example11
Bastic_23c Example12
Bastic_24 Example13
Bastic_25 Example14
Bastic_26 Example15
Bastic_27 Vadn_LP1
Example01 Vadn_LP2
Example02 Vadn_LP3
Example03 Vadn_LP4

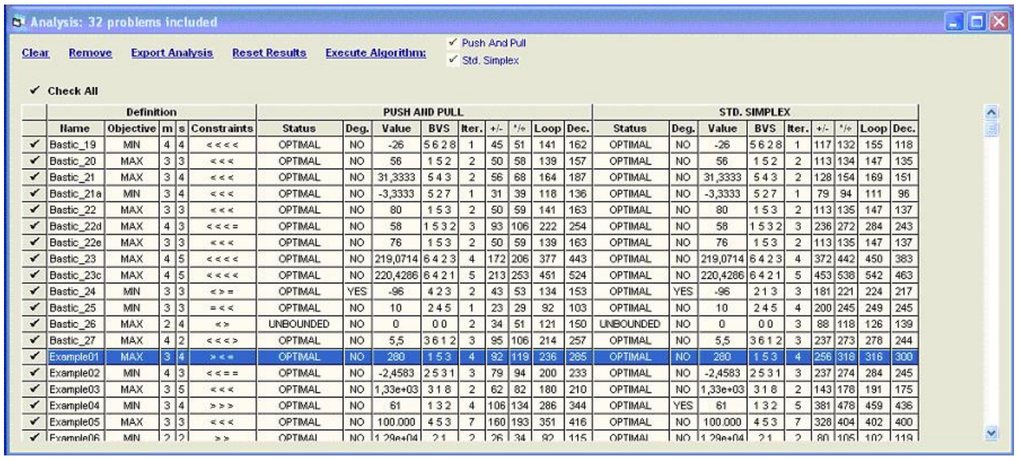
3.4.3. Analysis Window

The Analysis Window is created to work with multiple problems. As can be seen in Figure 4, it consists of two main elements (beside a standard window bar, stating the number of problems included in analysis): the Analysis Window Menu and the Analysis Window Problem List.

The Analysis Menu consists of two User-Controls: a MyActiveLabels and a MyCheckLabels controls. The MyCheckLabels part enables user to select one or both algorithms to be executed (or results cleared). The MyActiveLabels part takes care of carrying out the Analysis Window tasks, such as clearing the Problem List, removing a single problem from the Problem List, exporting current Problem List as a Tab-delimited text file (it is suitable for further analyzing with e.g. MS Excel), resetting results and executing (selected) methods.

The Problem List is based on MyGrid user-control. Its purpose is to display multiple problems with their definitions and result summaries for each linear programming algorithm, in a single table. That helps the user to find and understand some interesting behaviors of algorithms execution. The Problem List fields are explained in detail in the SixPap on-line help.

Figure 4: The Analysis Window



Analysis: 32 problems included

Clear Remove Export Analysis Reset Results Execute Algorithm: ✓ Push And Pull ✓ Std. Simplex

✓ Check All

	Definition					PUSH AND PULL									STD. SIMPLEX								
	Name	Objective	m	s	Constraints	Status	Deg.	Value	BVS	Iter.	+/-	*/+	Loop	Dec.	Status	Deg.	Value	BVS	Iter.	+/-	*/+	Loop	Dec.
✓	Bastic_19	MIN	4	4	< < < <	OPTIMAL	NO	-26	5 6 2 8	1	45	51	141	162	OPTIMAL	NO	-26	5 6 2 8	1	117	132	155	118
✓	Bastic_20	MAX	3	3	< < <	OPTIMAL	NO	56	1 5 2	2	50	58	139	157	OPTIMAL	NO	56	1 5 2	2	113	134	147	135
✓	Bastic_21	MAX	3	4	< < <	OPTIMAL	NO	31,3333	5 4 3	2	56	68	164	187	OPTIMAL	NO	31,3333	5 4 3	2	128	154	169	151
✓	Bastic_21a	MIN	3	4	< < <	OPTIMAL	NO	-3,3333	5 2 7	1	31	39	118	136	OPTIMAL	NO	-3,3333	5 2 7	1	79	94	111	96
✓	Bastic_22	MAX	3	3	< < <	OPTIMAL	NO	80	1 5 3	2	50	59	141	163	OPTIMAL	NO	80	1 5 3	2	113	135	147	137
✓	Bastic_22d	MAX	4	3	< < < =	OPTIMAL	NO	58	1 5 3 2	3	93	106	222	254	OPTIMAL	NO	58	1 5 3 2	3	236	272	284	243
✓	Bastic_22e	MAX	3	3	< < <	OPTIMAL	NO	76	1 5 3	2	50	59	139	163	OPTIMAL	NO	76	1 5 3	2	113	135	147	137
✓	Bastic_23	MAX	4	5	< < < <	OPTIMAL	NO	219,0714	6 4 2 3	4	172	206	377	443	OPTIMAL	NO	219,0714	6 4 2 3	4	372	442	450	383
✓	Bastic_23c	MAX	4	5	< < < <	OPTIMAL	NO	220,4286	6 4 2 1	5	213	253	451	524	OPTIMAL	NO	220,4286	6 4 2 1	5	453	538	542	463
✓	Bastic_24	MIN	3	3	< > =	OPTIMAL	YES	-96	4 2 3	2	43	53	134	153	OPTIMAL	YES	-96	2 1 3	3	181	221	224	217
✓	Bastic_25	MIN	3	3	= < <	OPTIMAL	NO	10	2 4 5	1	23	29	92	103	OPTIMAL	NO	10	2 4 5	4	200	245	249	245
✓	Bastic_26	MAX	2	4	< >	UNBOUNDED	NO	0	0 0	2	34	51	121	150	UNBOUNDED	NO	0	0 0	3	88	118	126	139
✓	Bastic_27	MAX	4	2	< < < >	OPTIMAL	NO	5,5	3 6 1 2	3	95	106	214	257	OPTIMAL	NO	5,5	3 6 1 2	3	237	273	278	244
✓	Example01	MAX	3	4	> < =	OPTIMAL	NO	280	1 5 3	4	92	119	236	285	OPTIMAL	NO	280	1 5 3	4	256	318	316	300
✓	Example02	MIN	4	3	< < = =	OPTIMAL	NO	-2,4583	2 5 3 1	3	79	94	200	233	OPTIMAL	NO	-2,4583	2 5 3 1	3	237	274	284	245
✓	Example03	MAX	3	5	< < <	OPTIMAL	NO	1,33e+03	3 1 8	2	62	82	180	210	OPTIMAL	NO	1,33e+03	3 1 8	2	143	178	191	175
✓	Example04	MIN	3	4	> > >	OPTIMAL	NO	61	1 3 2	4	106	134	286	344	OPTIMAL	YES	61	1 3 2	5	381	478	459	436
✓	Example05	MAX	3	3	< < <	OPTIMAL	NO	100.000	4 5 3	7	160	193	351	416	OPTIMAL	NO	100.000	4 5 3	7	328	404	402	400
✓	Example06	MIN	2	2	> >	OPTIMAL	NO	1,29e+04	2 1	2	26	34	92	115	OPTIMAL	NO	1,29e+04	2 1	2	80	105	102	119

3.4.4. Single Problem Window

The purpose of the Single Problem Window is to help the user to work with single (selected) problem and its details more easily.

The Single Problem Window consists, as is shown in Figure 5, of four main elements (beside a standard window bar with the name of currently opened problem): the Single Problem Menu, the Definition, Result Summary and Result Details area.

The Single Problem Menu helps executing tasks, such as saving and printing selected problem, and executing and resetting results for selected algorithms. It consists of a MyActiveLabels and MyCheckLabels user-controls.

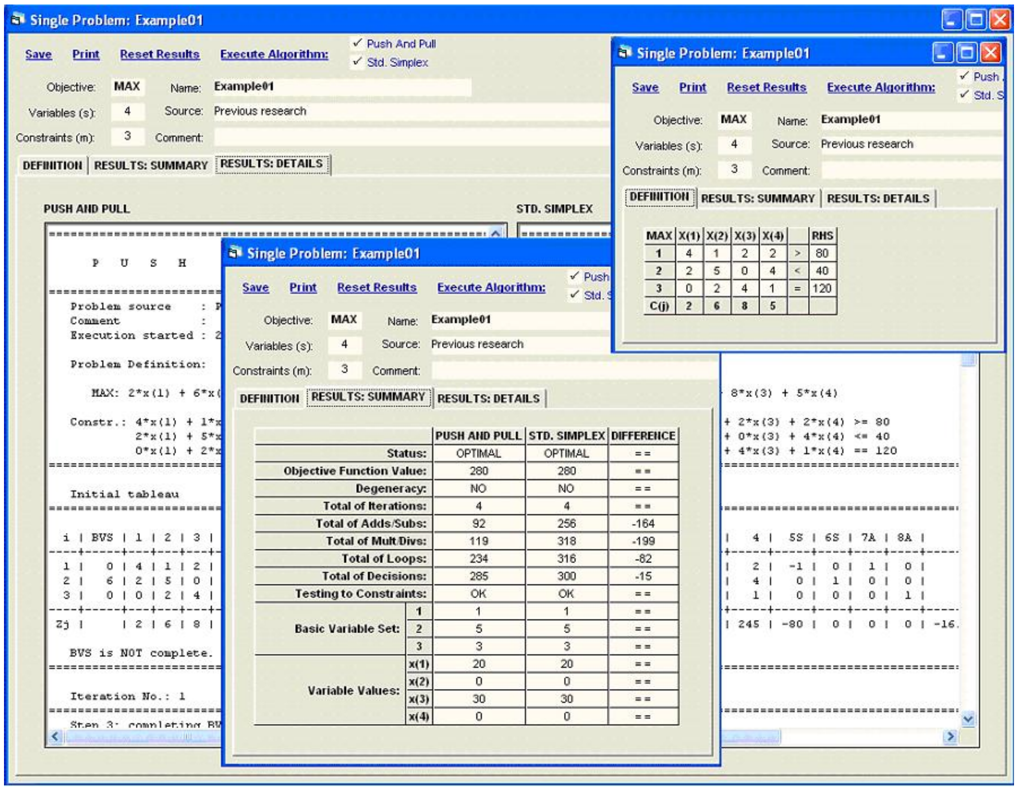
The problem general variables area is located immediately under the menu. It consists of six general variables, defining the linear programming problem: the objective function, the number of variables, the number of constraints, the problem name and two comment variables (the source of the problem and a general comment the user can give to the problem). All those variables can be edited, with exception of the problem name.

The Definition tab contains the Problem Definition Tableau. It is based on a MyGrid user-control, and displays the problem definition in detail. The Tableau can be easily edited.

The Result Summary tab contains the Result Summary Table (it is also based on a MyGrid user-control) where the characteristic result values for one or both algorithm are displayed. There is also a column added, that displays a calculated difference between algorithm results.

The Result Details tab contains a MyRichTextBox user-control. This control shows the contents of both algorithm detail files (empty, if it does not exists) for currently opened single problem.

Figure 5: The Single Problem Window



Objective:	MAX
Name:	Example01
Variables (s):	4
Source:	Previous research
Constraints (m):	3
Comment:	

```
==========================================================

     P   U   S   H

==========================
   Problem source     : P
   Comment            :
   Execution started  : 2

   Problem Definition:

      MAX: 2*x(1) + 6*x(

   Constr.: 4*x(1) + 1*x
            2*x(1) + 5*x
            0*x(1) + 2*x
==========================

   Initial tableau
==========================

  i | BVS | 1 | 2 | 3 |
----+-----+---+---+---+-
  1 |   0 | 4 | 1 | 2 |
  2 |   6 | 2 | 5 | 0 |
  3 |   0 | 0 | 2 | 4 |
----+-----+---+---+---+-
 Zj |     | 2 | 6 | 8 |

   BVS is NOT complete.
==========================

   Iteration No.: 1
==========================
   Step 3: completing BV
```

```
 8*x(3) + 5*x(4)

+ 2*x(3) + 2*x(4) >= 80
+ 0*x(3) + 4*x(4) <= 40
+ 4*x(3) + 1*x(4) == 120
========================================


========================================

|   4 |  5S | 6S | 7A | 8A |
+-----+-----+----+----+----+----
|   2 |  -1 |  0 |  1 |  0 |
|   4 |   0 |  1 |  0 |  0 |
|   1 |   0 |  0 |  0 |  1 |
+-----+-----+----+----+----+----
| 245 | -80 |  0 |  0 |  0 | -16.

========================================

========================================
```

MAX	X(1)	X(2)	X(3)	X(4)		RHS
1	4	1	2	2	>	80
2	2	5	0	4	<	40
3	0	2	4	1	=	120
C(j)	2	6	8	5		

		PUSH AND PULL	STD. SIMPLEX	DIFFERENCE
Status:		OPTIMAL	OPTIMAL	==
Objective Function Value:		280	280	==
Degeneracy:		NO	NO	==
Total of Iterations:		4	4	==
Total of Adds/Subs:		92	256	-164
Total of Mult/Divs:		119	318	-199
Total of Loops:		234	316	-82
Total of Decisions:		285	300	-15
Testing to Constraints:		OK	OK	==
Basic Variable Set:	1	1	1	==
	2	5	5	==
	3	3	3	==
Variable Values:	x(1)	20	20	==
	x(2)	0	0	==
	x(3)	30	30	==
	x(4)	0	0	==

3.4.5. User controls

Beside standard Windows controls, the user interface includes a set of custom made controls - user controls - such as MyActiveLabels, MyCheckLabels, MyFileListView, MyGrid and MyRichTextBox. They contribute to system functionality and make the system construction more transparent

MyActiveLabels user control acts as a command menu. It is based on an array of labels, which are sensitive to the mouse movement (text colour changes, when mouse cursor moves over) and reacts to clicking. When a label is clicked it activates an event which can be treated as a command. Array of labels can be displayed in vertical or horizontal direction.

MyCheckLabels user control acts as a check boxed menu with selectable options. It is based on an array of labels, which are sensitive to the mouse movement (text colour changes, when mouse cursor moves over) and react to clicking. When a label (or a box in front of it) is clicked the box in front of it toggles between clear and ticked. A label has a property which holds a selection value (True or False).

MyFileListView user control is based on standard MS Windows list view control. It shows names (without extensions) of files that exist in a selected folder. This control allows standard Multi Selection.

MyFileListView also allows label editing by two consecutive clicks on item if CanEdit property is set to True. If not canceled, after editing the control raises an event notifying that item name has been edited.

MyGrid user control is a table-like control, based on MSFlexGrid extended with the following functionality: Selecting And Checking, Hiding / Showing columns and Editing grid cells, which are accessible to the user as well as lots of other useful functions that help the developer.

MyRichTextBox user control is basically a pair of MSRichTextBox controls with some additional functionality. That includes reading data from files and displaying the content, moving the vertical delimiter bar to enlarge one (and shrink the other) display box. Each box also has a caption.

4. Numerical illustrative example

In following we show a step-by-step example of a computer implementation of Push-and-Pull algorithm, applied to Numerical Example 1. The fully coded version of the algorithm is available from the authors upon request.

Numerical Example 1:

Problem Definition:

$Min \quad 1X1 + 3X2 + 4X3 + 10X4$

Constr.:

$$1X1 + 0X2 + 1X3 + 1X4 \geq 10 \ ,$$
$$0X1 + 1X2 + 2X3 + 2X4 \geq 25 \ ,$$
$$1X1 + 2X2 + 0X3 + 1X4 \geq 20 \ .$$

Initial tabeleau:

i	BVS	1	2	3	4	5S	6S	7S	RHS
1	0	1	0	1	1	-1	0	0	10
2	0	0	1	2	2	0	-1	0	25
3	0	1	2	0	1	0	0	-1	20
Z_j		-1	-3	-4	-10	0	0	0	0

BVS is NOT complete.

Iteration No.: 1

Step 3: completing BVS ($k = 1$, $r = 1$)

i	BVS	1	2	3	4	5S	6S	7S	RHS
1	1	1	0	1	1	-1	0	0	10
2	0	0	1	2	2	0	-1	0	25
3	0	0	2	-1	0	1	0	-1	10
Z_j		0	-3	-3	-9	-1	0	0	10

BVS in NOT complete.

Iteration No.: 2

Step 3: completing BVS ($k = 5$, $r = 3$)

i	BVS	1	2	3	4	5S	6S	7S	RHS
1	1	1	2	0	1	0	0	-1	20
2	0	0	1	2	2	0	-1	0	25
3	5	0	2	-1	0	1	0	-1	10
Z_j		0	-1	-4	-9	0	0	-1	20

BVS in NOT complete.

Iteration No.: 3

Step 3: completing BVS ($k = 3$, $r = 2$)

i	BVS	1	2	3	4	5S	6S	7S	RHS
1	1	1	2	0	1	0	0	-1	20
2	3	0	0.5	1	1	0	-0.5	0	12.5
3	5	0	2.5	0	1	1	-0.5	-1	22.5
Z_j		0	1	0	-5	0	-2	-1	70

BVS is complete.

Continue with Step 4.

Iteration No.: 4

Step 4: Push to optimal solution ($k = 2$, $r = 3$)

i	BVS	1	2	3	4	5S	6S	7S	RHS
1	1	1	0	0	0.2	-0.8	0.4	-0.2	2
2	3	0	0	1	0.8	-0.2	-0.4	0.2	8
3	2	0	1	0	0.4	0.4	-0.2	-0.4	9
Z_j		0	0	0	-5.4	-0.4	-1.8	-0.6	61

Z_j positive does NOT exist.

Continue with Step 5.

Step 5: Test the iteration results:

All RHS ≥ 0 and all $Z_j \leq 0$: solution is optimal

RESULT TESTING

constraint no. 1 TRUE: $10 \geq 10$
constraint no. 2 TRUE: $25 \geq 25$
constraint no. 3 TRUE: $20 \geq 20$

Test result: OK

SUMMARY

Status: OPTIMAL
Degeneracy: NO
Test to constraints: OK

MIN: 61
BVS: [1,3,2]
$X1 = 2$
$X2 = 9$
$X3 = 8$
$X4 = 0$

Number of iterations: 4
Number of additions/subtractions: 106
Number of multiplications/divisions: 134
Number of loops: 286
Number of decisions: 344

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

$max \quad 2X_1 + 6X_2 + 8X_3 + 5X_4$

Subject to

$$\begin{array}{rcrcrcrcl} 4X_1 & + & X_2 & + & 2X_3 & + & 2X_4 & \geq & 80 \\ 2X_1 & + & 5X_2 & & & + & 4X_4 & \leq & 40 \\ & & 2X_2 & + & 4X_3 & + & X_4 & = & 120 \end{array}$$

Example 02

$min \quad 2X_1 + 3X_2 - 6X_3$

Subject to

$$\begin{array}{lllll} X_1 & + 3X_2 & + 2X_3 & \leq & 3 \\ & -2X_2 & + X_3 & \leq & 1 \\ X_1 & - X_2 & + X_3 & = & 2 \\ 2X_1 & + 2X_2 & - 3X_3 & = & 0 \end{array}$$

Example 03

$max \quad 20X_1 + 10X_2 + 40X_3 + 20X_4 + 15X_5$

Subject to

$$\begin{array}{llllll} X_1 & & + 4X_3 & + 2X_4 & + 4X_5 & \leq 120 \\ 2X_1 & + 5X_2 & + 2X_3 & + X_4 & & \leq 80 \\ 4X_1 & + X_2 & + 5X_3 & & + 4X_5 & \leq 240 \end{array}$$

Example 04

$min \quad X_1 + 3X_2 + 4X_3 + 10X_4$

Subject to

$$\begin{array}{lllll} X_1 + & & + X_3 & + X_4 & \geq 10 \\ & X_2 & + 2X_3 & + 2X_4 & \geq 25 \\ X_1 & + 2X_2 & & + X_4 & \geq 20 \end{array}$$

Example 05

$max \quad 100X_1 + 10X_2 + X_3$

Subject to

$$\begin{array}{llll} X_1 & & & \leq 1 \\ 20X_1 & + X_2 & & \leq 100 \\ 200X_1 & + 20X_2 & + X_3 & \leq 100\ 000 \end{array}$$

Example 06

$min \quad 2X_1 + 2.5X_2$

Subject to

$$\begin{array}{lll} 3X_1 & + 4X_2 & \geq 20000 \\ 0.75X_1 & + 0.65X_2 & \geq 4000 \end{array}$$

Example 07

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

Example 08

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

Example 09

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

Example 10

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

Example 11

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

Example 12

$$
\begin{array}{ll}
min & 4X_1 - 2X_2 + X_3 \\
\text{Subject to} & \\
& 2X_1 - X_2 + X_3 \leq 30 \\
& 3X_1 + X_2 - X_3 \leq 26 \\
& X_2 + X_3 \leq 13
\end{array}
$$

Example 13

$$
\begin{array}{ll}
max & 12X_1 + 6X_2 + 4X_3 \\
\text{Subject to} & \\
& X_1 + 2X_2 \leq 6 \\
& -X_1 - X_2 + 2X_3 \leq 4 \\
& X_2 + X_3 \leq 2
\end{array}
$$

Example 14

$$
\begin{array}{ll}
max & -X_1 + 21X_2 + 24X_3 + X_4 + 12X_5 \\
\text{Subject to} & \\
& 0.5X_1 + 6X_2 + 7X_3 - 2X_4 + X_5 \leq 64 \\
& -X_1 + 2X_2 + 4X_3 + 2X_4 + 5X_5 \leq 27 \\
& 2X_1 + 3X_2 - X_3 - 4X_4 + 3X_5 \leq 17 \\
& 3X_1 + 2X_2 + 4X_3 - X_4 - X_5 \leq 21
\end{array}
$$

Example 15

$$
\begin{array}{ll}
min & 3X_1 + X_2 - X_3 \\
\text{Subject to} & \\
& 2X_1 + X_2 - X_3 = 10 \\
& -X_1 + X_3 \leq 6 \\
& 3X_1 - 4X_3 \leq 8
\end{array}
$$

In our comparison analysis we used five counters: total number of iterations, additions / subtractions, multiplications / divisions, loops and decisions.

The results are displayed in Table 1, which has four sections: the characteristics of problem definitions for each example, the Push-and-Pull, ordinary simplex result summary, and the calculated Difference section. As can be seen in Table 1, Push-and-Pull needed less iterations to achieve optimal solutions in examples 4 and 15 and the same number as simplex in all others. Concerning other four parameters it appears that Push-and-Pull characteristic is that it uses less arithmetical operations but more decision steps (which are less expensive from calculation point of view).

Table 1. Results of comparison analysis

Definition					PUSH AND PULL								
Name	Obj.	m	s	Constr.	Status	Deg.	Value	BVS	Iter	+/-	*/+	Loop	Dec.
Example01	MAX	3	4	>\<=	OPTIMAL	NO	280	153	4	92	119	234	285
Example02	MIN	4	3	\<\<==	OPTIMAL	NO	-2.4583	2531	3	79	94	199	233
Example03	MAX	3	5	\<\<\<	OPTIMAL	NO	1.33E+03	318	2	62	82	180	210
Example04	MIN	3	4	>>>	OPTIMAL	NO	61	132	4	106	134	286	344
Example05	MAX	3	3	\<\<\<	OPTIMAL	NO	100 000	453	7	160	193	351	416
Example06	MIN	2	2	>>	OPTIMAL	NO	1.29E+04	21	2	26	34	92	115
Example07	MAX	3	3	\<\<\<	OPTIMAL	NO	34 200	416	2	50	61	144	173
Example08	MAX	2	2	\<\<	OPTIMAL	NO	1.29E+04	21	2	26	34	87	110
Example09	MAX	3	2	\<\<\<	OPTIMAL	NO	198	142	2	44	52	116	142
Example10	MIN	3	3	\<>=	OPTIMAL	YES	-96	531	3	62	76	169	201
Example11	MAX	3	2	\<\<\<	OPTIMAL	NO	20	315	1	25	28	83	102
Example12	MIN	3	3	\<\<\<	OPTIMAL	NO	-26	452	1	28	33	96	113
Example13	MAX	3	3	\<\<\<	OPTIMAL	NO	80	153	2	50	59	140	163
Example14	MAX	4	5	\<\<\<\<	OPTIMAL	NO	219.0714	6423	4	172	206	377	443
Example15	MIN	3	3	=\<\<	OPTIMAL	NO	10	245	1	23	29	95	103

STD.SIMPLEX									DIFFERENCE								
Status	Deg.	Value	BVS	Iter	+/-	*/-	Loop	Dec.	Status	Deg.	Value	BVS	Iter	+/-	*/+	Loop	Dec.
OPTIMAL	NO	280	153	4	256	318	316	300	ok	ok	ok	ok	0	-164	-199	-82	-15
OPTIMAL	NO	-2.4583	2531	3	237	274	284	245	ok	ok	ok	ok	0	-158	-180	-85	-12
OPTIMAL	NO	1.33E+03	318	2	143	178	191	175	ok	ok	ok	ok	0	-81	-96	-11	35
OPTIMAL	YES	61	132	5	381	478	459	436	ok	DIFF	ok	ok	-1	-275	-344	-173	-92
OPTIMAL	NO	100 000	453	7	328	404	402	400	ok	ok	ok	ok	0	-168	-211	-51	16
OPTIMAL	NO	1.29E+04	21	2	80	105	102	119	ok	ok	ok	ok	0	-54	-71	-10	-4
OPTIMAL	NO	43 200	416	2	113	137	147	141	ok	ok	ok	ok	0	-63	-76	-3	32
OPTIMAL	NO	1.29E+04	21	2	56	73	78	93	ok	ok	ok	ok	0	-30	-39	9	17
OPTIMAL	NO	198	142	2	98	118	125	127	ok	ok	ok	ok	0	-54	-66	-9	15
OPTIMAL	YES	-96	531	3	181	221	224	217	ok	ok	ok	ok	0	-119	-145	-155	-16
OPTIMAL	NO	20	315	1	61	71	81	78	ok	ok	ok	ok	0	-36	-43	2	24
OPTIMAL	NO	-26	452	1	70	82	96	86	ok	ok	ok	ok	0	-42	-49	0	27
OPTIMAL	NO	80	153	2	113	135	147	137	ok	ok	ok	ok	0	-63	-76	-7	26
OPTIMAL	NO	219.0714	6423	4	372	442	450	383	ok	ok	ok	ok	0	-200	-236	-73	60
OPTIMAL	NO	10	245	4	200	245	249	245	ok	ok	ok	ok	-3	-177	-216	-154	-142

For the further enhancement of this algorithm, some refinements might be needed in the computerized implementation phase. For example, in deciding on exchanging the variables, one might consider to total contribution, rather than marginal contribution to the current objective function value. In other words, recall that the choice of smallest non-negative C/R provide the marginal change in the objective function, while maintaining feasibility. However, since we are constructing the C/R for all candidate variables, we may obtain the largest contribution by considering the product C_j (C/R) in our selection rule. The following example illustrates this enhancement in the algorithm:

Example: The following problem is attributed to Klee and Minty.
$max \qquad 100X_1 + 10X_2 + X_3$

Subject to

$$\begin{array}{lcl} X_1 & & \leq 1 \\ 20X_1 & + X_2 & \leq 100 \\ 200X_1 & + 20X_2 + X_3 & \leq 100000, \end{array}$$

and $X_1, X_2, X_3 \geq 0$.

This example is constructed to show that the Simplex method is exponential in its computational complexity. For this example with $n = 3$, it takes $2^3 = 8$ iterations.
Converting the constraints to equalities, we have:
$max \qquad 100X_1 + 10X_2 + X_3$
Subject to

$$\begin{array}{llllll} X_1 & & + S_1 & & & = 1 \\ 20X_1 & + X_2 & & + S_2 & & = 100 \\ 200X_1 & + 20X_2 + X_3 & & & + S_3 & = 100000, \end{array}$$

and $X_1, X_2, X_3 \geq 0$.

Initialization: The initial tableau with the two slack variables as its BV is:

BVS	X_1	X_2	X_3	S_1	S_2	S_3	RHS	C/R1	C/R2	C/R3
S_1	1	0	0	1	0	0	1	1		
S_2	20	1	0	0	1	0	100	5	100	
S_3	200	20	1	0	0	1	100000	500	5000	100000
C_j	100	10	1	0	0	0				

Total Contribution Criterion:

- The smallest non-negative C/R1 is 1, which belongs to X_1, with marginal contribution of C1 = 100. Therefore, its total contribution to the current objective function value is C1(C/R1)= 100(1) = 100.
- The smallest non-negative C/R2 is 100, which belongs to X_2, with marginal contribution of C2 = 10. Therefore, its total contribution to the current objective function value is C2(C/R2)= 10(100) = 1000.
- The smallest non-negative C/R3 is 100000, which belongs to X_3 with marginal contribution of C3 = 1. Therefore, its total contribution to the current objective function value is C3(C/R3)= 1(100000) = 100000.

Therefore, based on the "total contribution criterion", in our variable selection, variable X_3 comes in. Beak any ties arbitrarily.

The Push Further Phase: The candidate variables to enter are X_1, X_2, and X_3 with C/R shown in the above table. Based on the total contribution criterion, the largest contribution to the objective function achieved by bringing X_3 in. Therefore, replacing S_3 with X_3, after pivoting we get:

BVS	X_1	X_2	X_3	S_1	S_2	S_3	RHS
S_1	1	0	0	1	0	0	1
S_2	20	1	0	0	1	0	100
X_3	200	20	1	0	0	1	100000
C_j	-100	-10	0	0	0	-1	

This tableau is the optimal one. The optimal solution is: $X_1 = 100000$, $X_2 = 0$, $X_3 = 0$, with the optimal value of 100000.

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 [9, 12] and increases the computational complexity [13]. 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 [15]. 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- A computer implementation system is developed to enable the reader to run one or both algorithms. The system also creates a detailed and comparison analysis between Push-and-Pull and ordinary simplex. The fully coded version of the algorithm is available from the authors upon request.

A comparative study has been done, which compares Push-and-Pull to ordinary simplex. The result of this study shows that the new proposed algorithm 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 and 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(3) (1997) 289-303.

[3] H. Arsham, *Initialization of the simplex algorithm: An artifical-free approach*, ***SIAM Rewiew***, 39(4) (1997) 736-744.

[4] H. Arsham, *Distribution-routes stability analysis of the transportation problem*, ***Optimization***, 43(1) (1998) 47-72.

[5] H. Arsham, B. Baloh, T. Damij and J. Grad, *An algorithm for simplex tableau reduction with numerical comparison*, ***International Journal of Pure Applied Mathematics***, Vol. 4, No. 1, (2003) 57-85

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

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

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

[9] T. Gao, T. Li, J. Verschelde, and M. Wu, *Balancing the lifting values to improve the numerical stability of polyhedral homotopy continuation methods*, ***Applied Mathematics and Computation***, 114 (2-3) (2000) 233-247.

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

[11] V. Klee and G. Minty, *How good is the simplex algorithm*, ***Inequalities-III***, Edited by 0. Shisha, Academic Press, (1972) 159-175.

[12] V. Lakshmikantham, S. Sen, M. Jain, and A. Ramful, $O(n^3)$ *Noniterative heuristic algorithm for linear programs with error-free implementation*, ***Applied Mathematics and Computation***, 110 (1) (2000) 53-81.

[13] G. Mohr, *Finite Element modelling of distribution problems*, ***Applied Mathematics and Computation***, 105 (1) (1999) 69-76.

[14] K. Papparrizos, *The two-phase simplex without artificial variables*, ***Methods of operations Research***, 61 (1) (1990) 73-83.

[15] A. Ravindran, *A comparison of the primal-simplex and complementary pivot methods for linear programming*, ***Naval Research Logistic Quarterly***, 20 (1) (1973) 95-100.

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