© Journal of Applied Mathematics & Decision Sciences, 2(1), 65–104 (1998)
Reprints Available directly from the Editor. Printed in New Zealand.

MANAGING COST UNCERTAINTIES IN TRANSPORTATION AND ASSIGNMENT PROBLEMS

V. ADLAKHA AND H. ARSHAM
Business Center, University of Baltimore,
1420 N. Charles Street, Baltimore, MD 21201-5779, USA

Abstract. In a fast changing global market, a manager is concerned with cost uncertainties of the cost matrix in transportation problems (TP) and assignment problems (AP). A time lag between the development and application of the model could cause cost parameters to assume different values when an optimal assignment is implemented. The manager might wish to determine the responsiveness of the current optimal solution to such uncertainties. A desirable tool is to construct a perturbation set (PS) of cost coefficients which ensures the stability of an optimal solution under such uncertainties.
The widely-used methods of solving the TP and AP are the stepping-stone (SS) method and the Hungarian method, respectively. Both methods fail to provide direct information to construct the needed PS. An added difficulty is that these problems might be highly pivotal degenerate. Therefore, the sensitivity results obtained via the available linear programming (LP) software might be misleading.
We propose a unified pivotal solution algorithm for both TP and AP. The algorithm is free of pivotal degeneracy, which may cause cycling, and does not require any extra variables such as slack, surplus, or artificial variables used in dual and primal simplex. The algorithm permits higher-order assignment problems and side-constraints. Computational results comparing the proposed algorithm to the closely-related pivotal solution algorithm, the simplex, via the widely-used package Lindo, are provided. The proposed algorithm has the advantage of being computationally practical, being easy to understand, and providing useful information for managers. The results empower the manager to assess and monitor various types of cost uncertainties encountered in real-life situations. Some illustrative numerical examples are also presented.

Keywords: Transportation, assignment, linear models, cost sensitivity analysis, push-and-pull algorithm.

1. Introduction

The widely-used methods of solving transportation problems (TP) and assignment problems (AP) are the stepping-stone (SS) method and the Hungarian method, respectively. Managerial applications are many and go beyond these prototype problems to include job scheduling, production inventory, production distribution, allocation problems, and investment analysis, among others. As a dual-simplex algorithm, the SS proved successful for solving a TP and became the standard technique for over 50 years. In practice, however, the SS algorithm encounters major obstacles as a solution procedure. It has difficulties identifying an initial basic feasible solution, resolving SS degeneracy, and enumerating SS paths. A recent flurry of activity has improved the SS algorithm to an extent, enabling it to over-

come some of these deficiencies.

(a) Finding an initial basic feasible solution: A prerequisite for SS is a basic feasible solution with a certain number of positive entries. The best known method to achieve this is Vogel's, which is not applicable to an unbalanced problem. Goyal [39] modified Vogel's method to cover unbalanced cases, and Ramakrishnan [68] improved Goyal's approach. Sultan [76] suggested a heuristic to find an initial feasible solution that may produce an SS degenerate basis. Kirka and Satir [50] present another heuristic method with minimal improvement over Vogel's method for the unbalanced case.

(b) Resolving SS degeneracy: Shafaat and Goyal's [72] is the first paper with a systematic approach for handling SS degeneracy (i.e., failure of the optimality test.) Their algorithm, however, is limited to a special case of degeneracy.

(c) Enumerating SS paths: Finding SS paths can be very tedious, particularly for larger problems. Intrator and Szwarc [43] extended an existing decomposition method. Their approach makes it easier to find SS paths but requires solving several sub-problems. Later, Wilsdon [82] provided a systematic method for finding the SS path that must be applied in each iteration.

The AP is a special case of TP and is traditionally solved using the Hungarian method. Even though the method is more efficient than the SS method to solve an AP, to "draw the minimum number of lines to cover all the zeros in the reduced cost matrix may not be an easy task"[36]. Moreover, the Hungarian method is limited [55] to 2-dimensional problems (e.g., people to projects, jobs to machines).

Although we have gained new insights, these above-mentioned improvements do not lend themselves to a unified approach for an efficient solution algorithm. The network-based approaches have similar difficulties [25], [65].

There are strong motivations for performing perturbation analysis (PA) to deal with a collection of managerial questions related to so-called "what-if" problems. Managers are concerned with unforeseen changes in the input cost parameters. They are likely to be unsure of their current values and even more uncertain about their future values at the time when the solution is to be implemented. Uncertainty is the prime reason why PA is helpful in making decisions. We can use PA to give us information like: the robustness of an optimal solution; critical values, thresholds, and breaking-even values where the optimal strategy changes; sensitivity of important variables; sub-optimal solutions; flexible recommendations; the values of simple and complex decision strategies; and the "riskiness" of a strategy or scenario. This information provides systematic guidelines for allocating scarce organizational resources to data collection, data refinement, and prediction activities of the cost parameters, and could be considered as a model validation activity. Examining the

effects of an increase or decrease in unit costs of transportation would help, for example, to negotiate rates with trucking companies and in reacting to changed conditions caused by rate changes.

Although it has long been known that TP and AP can be modeled as linear programs, this is generally not done, due to the relative inefficiency and complexity of the simplex methods (primal, dual, and other variations). These problems are usually treated by one of over 20 specialized algorithms for each (see, e.g., [4], [9], [8], [10], [13], [14], [12], [19], [20], [29], [30], [31], [32], [37], [42], [45], [48], [49], [57], [58], [59], [61], [65], [66], [74], [78], [83]). This leads to several difficulties. The solution algorithms are not unified as each algorithm uses a different strategy to exploit the special structure of a specific problem. Furthermore, a small variation in the problem, such as the introduction of side-constraints, destroys the special structure and requires a new solution algorithm. These algorithms obtain solution efficiency at the expense of managerial insight, as the final solutions from these algorithms do not easily provide sufficient information to perform post optimality analysis for TP and AP.

Another approach is to adapt the simplex network optimization through network simplex. This provides unification of the various problems but maintains all inefficiencies of the simplex, such as pivotal degeneracy, as well as inflexibility to handle side-constraints. Even ordinary (one-change-at-a-time) sensitivity analysis (OSA) limits, long available in the simplex, are not easily available in network simplex.

Most linear programming books, including management science and operations research books, have extensive discussion of linear programming (LP) sensitivity analysis (SA) but remain silent about the SA and side-constraints for TP and AP. Both methods, the SS and the Hungarian, lack the ability to test the validity of the current optimal solution with changes in cost parameters without resolving the problem. Some textbooks encourage the reader to formulate the problem as an LP and solve with a software package. The available computer software packages also have not proven to be effective in dealing with the SA of TP and AP as an LP.

Netsolve, Netflo, and Genos are three network-based software packages that have a subroutine for TP and AP. Netsolve provides OSA for problems with non-degenerate final tableau. Netflo and Genos have no SA capability [47]. Cplex [22], the current state-of-the-art code for solving network problems, incorporates nothing special for TP or AP's SA. It uses LP-type SA which can be misleading because of a degenerate optimal solution, especially for AP. Another popular software package, QSB, has a module for TP that automatically deletes the last constraint from the input to an LP formulation. (See [63] for details of these packages). Lindo, a general LP package, yields one change at a time for cost coefficient. However, the results are misleading for AP, and could be so for TP with degenerate optimal tableau. To the best of our knowledge, the literature still lacks managing cost uncertainties for

a degenerate final tableau case, which is common in these problems.

We propose a single unified algorithm that solves both TP and AP, and also provides useful information to perform cost-sensitivity analysis to a decision maker. Similar to the simplex algorithm and its many variants, the proposed solution algorithm is pivotal. The algorithm initiates the solution with a warm-start and does not require any slack/surplus or artificial variables. Unlike the Hungarian method, the algorithm can solve higher than 2-dimensional AP. The proposed solution algorithm also facilitates incorporation of side-constraints, which are frequently encountered in real-life applications. This algorithm makes available the full power of LP's SA extended to handle optimal degenerate solution. The essential calculations of proposed PA involve the same data and the same manipulation as the solution algorithm. With little extra computational effort we can obtain the perturbed solution, which provides the necessary information to construct the "critical" region, that is, the largest set of cost values for which the current solution remains optimal. In contrast to OSA, the proposed PA provides ranges for which the current optimal basis remains optimal, for simultaneous dependent or independent changes of the cost coefficients from their nominal values. The preliminary computational results demonstrate that the algorithm is more efficient than the simplex method in terms of number of iterations and size of tableaux. The proposed approach is easy to understand, easy to implement, and thus can serve as effective tools for solving TP, AP, and related problems in all phases, namely design, analysis, and operation.

The next section presents the solution algorithm which includes an illustrative TP numerical example. This is followed by a discussion on geometric representation and theoretical properties of the proposed solution algorithm in Section 3. Section 4 is devoted to the solution of a 3-dimensional AP. Computational behavior and computer implementation issues are presented in Section 5. General perturbation analysis of cost coefficients is covered in Section 6, followed by special cases of sensitivity analysis in Section 7. Handling the side-constraints and PA of the optimal degenerate case problems are given in Sections 8 and 9 respectively. The last section presents some concluding remarks. The proofs are given in the appendix.

2. The Solution Algorithm

A shipper having m warehouses with supply S_i of goods at his i^{th} warehouse must ship goods to n geographically dispersed retail centers, each with a given customer demand D_j which must be met. The objective is to determine the minimum possible transportation cost given that the unit cost of transportation between the i^{th} warehouse and the j^{th} retail center is C_{ij}.

This problem is known as the TP and has the following standard LP formulation:

$$\text{Minimize } \sum\sum C_{ij}X_{ij},$$

subject to

$$\sum X_{ij} = S_i \qquad i = 1, 2, .., m \tag{1}$$

$$\sum X_{ij} = D_j \qquad j = 1, 2, .., n \tag{2}$$

with the balanced condition

$$\sum S_i = \sum D_j \tag{3}$$

and $X_{ij}, D_j, S_i \geq 0$. Note that any unbalanced problem can be converted to a balanced one by adding a row or column. Clearly any of the above $m + n$ constraints which are redundant can be deleted. When S_i and $D_j = 1$, this problem refers to the assignment problem that involves determining the most efficient assignment of people to projects, jobs to machines, salespeople to territories, contracts to bidders, and so on. The objective is to minimize total costs or total time for performing the tasks at hand. In a 2-dimensional problem, the optimal solution of AP would be an integer, i.e., only one job or worker is assigned to one machine or project. Also note that in an AP, $n = m$ and in any optimal solution exactly n out of the $2n - 1$ basic variables have value 1 and $(n-1)^2$ non-basic variables along with $(n-1)$ basic variables have value 0. Thus, AP as an LP is extremely (primal) degenerate. The PA using any software packages provide unrealistic results since all these packages assume a problem to be non-degenerate.

Although the classical TP also requires that supply and demand (S_i and D_j) be integers (traditionally the number of units), we can relax this condition without loss of generality. Also, we relax the condition of an integer solution. We maintain, however, the condition in an AP that all workers and jobs etc., should be fully allocated, i.e., S_i and $D_j = 1$. The integrality condition of variables in these problems is superfluous, imposed by researchers only. The existing solution algorithms associated with TP and AP, such as the SS, Hungarian and network algorithms, can provide a solution only under such limiting condition. In real life situations, however, as in a higher dimensional AP, it is feasible and may be optimal to rotate workers among jobs, or salespeople among territories, and a TP could involve non-discrete demand and supply, such as weights and volume. When S_i and D_j are integers, our algorithm does provide an integer solution. Our approach, however, does not restrict these variables to being discrete. This is a significant difference between existing methods and the approach presented below. One can, however, introduce Gomory's cutting planes [6], [16] of LP to ensure an integer solution, if so desired.

2.1. A Pivotal Algorithm

We present steps of a simplex-type pivotal algorithm for the solution of general TP and AP, hereafter referred to as the *Push-and-Pull* algorithm, to reflect the two main strategies of the approach. The following additional notations of the simplex method are used:

GJP: Gauss-Jordan pivoting,
BV: basic variable,
BVS: BV set,
NB: non-basic variable,
PR: pivot row,
PC: pivot column,
PE: pivot element,
RHS: right hand side,
C/R: column ratio, RHS/PC,
OR: open row, a row not yet assigned to a BV, and
(?): label for an open row.

2.1.1. The Push-and-Pull Algorithm

The algorithm consists of cost adjustments and initialization followed by two phases.

Step 1 : *Reduced cost matrix*

1.1 : *Row-column reduction*
From each row subtract the smallest cost.
Then subtract the smallest cost in each column.
Accumulate the effect of row and column reductions into the base cost.

1.2 : *Eliminate redundant constraint*
Identify the row/s and/or column/s with the most zeros in the reduced cost matrix.
Eliminate the corresponding constraint/s (in case of an AP).

In this step we reduce the cost in the matrix cells by a base amount. This helps identify lowest cost cells with zeros for variables to enter into basis. Dropping the row with zeros facilitates a good start in the initialization phase, i.e., the next step.

Step 2 : *Initialization phase*

2.1 : *Set up the simplex tableau*
Use a row for each constraint and a column for each variable. Enter reduced C_{ij}'s in the cost row with a negative entry for the base cost under the RHS.

2.2 : *Identify BVs*
For each unit-vector column, label the row containing the "one" with the name of the variable for the column. Label the remaining rows as open rows.

2.3 : *Delete BV columns*

This phase tries to set up the initial tableau with as many BVs as possible in it without any iterations. These are the variables with only single +1 in its column. Since we drop the constraint with the most zeros, the phase gives many basic variables immediately. At the end of this phase, the tableau has entries of 1 and 0 only.

Step 3 : *Push phase*

3.1 : *BVS iteration termination test*
Check if a (?) label exists (open row). If none, BVS is complete. Go to step 4.

3.2 : *BV selection of PE*
PC: Select the smallest C_{ij} and any ties as candidate column(s). PR: Select open rows as candidate rows. PE: Select the candidate row and the column with the smallest non-negative C/R. If no non-negative C/R, choose the C/R with the smallest absolute value. If the pivot element is zero, then select the next best C_{ij}.

3.3 : *BVS augmentation*
Perform GJP. Replace the (?) row label with the variable name. Remove PC from the tableau. Go back to step 3.1.

The purpose of this phase is to complete the BVS of the preliminary phase while maintaining the optimality condition. The variables are brought into open rows marked by (?) only, and no replacement of variables in BVS is done in this phase. Thus, in this phase, we push toward optimality. We try to achieve this by maintaining feasibility and optimality (step 3.2) as much as possible. If by the end of this phase the RHS is positive, this is an optimum solution. Step 3.3 reduces the tableau by one column as you proceed to obtain short tableau.

Step 4 : Pull phase(feasibility phase)

4.1 : *Iteration termination test*
If $RHS \geq 0$, the tableau is optimal, STOP. Otherwise continue.

4.2 : *Selection of PE*
PR: row with the most negative RHS. PC: column with a negative element in the PR. Tie breaker: column with the smallest C_{ij}.

4.3 : *Transformation*
Save PC. Perform usual GJP. Exchange PC and PR labels. Replace the new PC with old saved PC. Go to 4.1.

The purpose of this phase is to make the current solution feasible while maintaining the optimality condition. This phase is similar to the dual simplex in that it starts from a vertex which is infeasible but close to the optimal solution (a warm-start). This is in contrast to the usual dual simplex which often starts far from the optimal vertex.

2.2. Numerical Example 1

We select a 3x3 TP used by Shih [73] to illustrate the Push-and-Pull algorithm.

Table 1. Cost matrix of Shih's problem.

	D_1	D_2	D_3	Supply
S_1	5	30	12	55
S_2	20	18	30	80
S_3	15	25	23	75
Demand	70	100	40	

Step 1 : Cost-matrix adjustment:

0	25	0
2	0	5
0	10	1

The base cost is $5(55) + 18(80) + 15(75) + 7(40) = 3120$ dollars. Delete the first demand constraint with two zeros from the initial simplex tableau.

Step 2 : Initialization Phase:
The following tableau is obtained after steps 2.1 through 2.3.

Table 2. Initial tableau for Shih's problem

BVS	X_{12}	X_{13}	X_{21}	X_{22}	X_{23}	X_{32}	X_{33}	RHS
X_{11}	1	1						55
?			1	1	1			80
X_{31}						1	1	75
?	1			1		1		100
?		1			1		1	40
Cost	25	0	2	0	5	10	1	-3120

Step 3 : Push phase: Variables X_{13}, X_{22}, and X_{21}, in that order, replace the (?) row labels, respectively. The following tableau is obtained after repeating steps 3.1 through 3.3.

BVS	X_{12}	X_{23}	X_{32}	X_{33}	RHS
X_{11}	1	-1		-1	15
X_{22}	1		1		100
X_{31}			1	1	75
X_{21}	-1	1	-1		-20
X_{13}		1		1	40
Cost	27	3	12	1	-3080

Step 4 : Pull phase: The $RHS < 0$, hence Pull phase is needed. Variable X_{32} enters to replace variable X_{21}. After GJP, the following optimal tableau is obtained with non-negative RHS.

Table 3. Optimal tableau for Shih's problem

BVS	X_{12}	X_{23}	X_{21}	X_{33}	RHS
X_{11}	1	-1		-1	15
X_{22}		1	1		80
X_{31}	-1	1	1	1	55
X_{32}	1	-1	-1		20
X_{13}		1		1	40
Cost	15	15	12	1	-3320

The optimal solution is: $X_{11} = 15$, $X_{22} = 80$, $X_{31} = 55$, $X_{32} = 20$, and $X_{13} = 40$, with a total cost of $ 3320.

2.3. Comments on the Push-and-Pull Algorithm

i) The entries in the body of the tableau are 1, -1, and 0 only.

ii) Cost row always has non-negative entries.

iii) RHS entries are all non-negative at optimality.

iv) There are only $m + n - 1$ basic variables in any solution. This is consistent with the theory that in a TP, no more than $m + n - 1$ cells should be occupied.

v) At the end of Step 3 we have an initial solution. The entries under non-basic variables would have a count of one more 1 than of -1. In fact, these entries are analogous to corner cells of a path of the SS method. When a NB variable comes into basis, we add to the BV cell with -1 and subtract from the cell with 1. For example in Table 3, to add a unit to NB variable X_{12} (an unoccupied cell), we have to take away a unit from X_{11} and X_{32} each with 1 entry and add a unit to variable X_{31} with -1 entry. This indeed is a closed SS path from X_{12} in this solution.

vi) The entries in the cost row for non-basic variables represent the amount by which the current total cost would increase if a unit of that variable is to be brought into the current basis.

vii) The algorithm avoids the use of the extra variables (slack and surplus) that are needed for the equality constraints, each of which must be converted to two inequalities, in the dual simplex [11] . The Push-and-Pull algorithm is also artificial-free, as compared with the simplex method. This reduces computational complexity considerably.

2.4. Alternate Solutions

The proposed algorithm provides a clear indication of the presence of alternate, optimal solutions upon termination. Clearly, different alternate solutions give the same cost. The decision maker, however, has the option of deciding which optimal solution to implement on the basis of all the other factors involved.

Note that generating all alternative solutions can be computationally burdensome and may not be necessary. Moreover, the occurrence of alternative solutions is rare and can easily be avoided by a slight perturbation of the cost parameters. A decision maker, however, can easily generate alternate optimal solutions, if necessary. Given a final solution, check the entries in the cost row to find if there are any alternate solutions. An entry of 0 in the cost row represents a non-basic variable which can come into the basis to provide an alternate solution. Discard any that result in the same optimal strategy, that is, the basis changes but optimal shipment routes do not change. This can happen if the final solution is primal degenerate. From the distinct optimal strategies, pick one to implement. Failure to identify alternate routes deprives a manager of the flexibility regarding decisions to reallocate capacity from one route to another while maintaining the same cost. The optimal solution for example 1 given in Table 3 indicates that there are no alternate solutions in this case.

3. Properties of The Push-and-Pull Algorithm

The Push-and-Pull algorithm operates in the space of the original variables and has a geometric interpretation of its strategic process. We provide this geometric interpretation of the algorithm by comparing it to the geometry behind the conventional simplex method.

The simplex method is a vertex-searching method. It starts at the origin, which is far away from the optimal solution for the TP and the AP. It then moves along the intersection of the boundary hyper-planes of the constraints, hopping from one vertex to 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, that is, 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 tries to reach feasibility, and the second phase strives for optimality. The simplex works in the space of $m*n+(m+n-1)$ dimensions, because there are $m*n$ X_{ij}s and $m+n-1$ artificial variables, where m is the number of supply points and n is the number of demand points for the TP or AP.

In contrast, the Push-and-Pull algorithm strives to create a full BVS, that is, the intersection of $m+n-1$ constraint hyper-planes of the TP or AP that 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, initial BVS (never empty for TP or AP), which may contain an optimal vertex or a vertex that is close to it. In the Push phase the algorithm pushes toward 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. Each successive iteration in the Push phase augments the BVS by including 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 we hit another hyper-plane. Recall that no replacement of variables is done in this phase. At each iteration we reduce the dimensionality of the working region until we fill up the BVS, indicating a vertex. This phase is free from pivotal degeneracy. The selection of an incoming variable as the one having the smallest C_{ij} helps push toward an optimal vertex. As a result, the next phase starts with a vertex in the neighborhood of an optimal vertex.

At the end of the Push phase the BVS is complete, indicating a vertex. If feasible, this is an optimal solution. If this basic solution is not feasible, it indicates that we have pushed too far. Note that, in contrast to the first phase of the simplex,

this infeasible vertex is to 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, and it is free from pivotal degeneracy since it removes any negative (not zero) RHS elements. The space of the Push-and-Pull algorithm is $m+n-1$ dimensions in the Push phase and $m*n$ dimensions in the Pull phase. Note that $m+n-1$ is the number of constraints and $m*n$ is the number of variables.

The importance of the Push-and-Pull algorithm is recognized by the fact that whereas 95 percent of the pivots in the simplex method are degenerate, which may cause cycling [62], in solving AP [28], the proposed algorithm is completely free of pivotal degeneracy.

Theorem 1: The Push-and-Pull algorithm is free from pivotal degeneracy which may cause cycling.

Theorem 2: The Push-and-Pull algorithm terminates successfully in a finite number of iterations.

The proofs of Theorem 1 and Theorem 2 are provided in the appendix.

4. An Assignment Problem

The Push-and-Pull algorithm provides a discrete optimal solution for a 2-dimensional AP, and if a discrete optimal solution exists, for higher-dimensional problems [7], [23], [35] . Since a 2-dimensional AP formulation is straight-forward, we present the algorithm for a 3-dimensional AP.

4.1. Numerical Example 2

Table 4. Cost matrix for the 2x2x2 AP

	Machine			
	1		2	
	Job		Job	
Person	1	2	1	2
Adams	10	5	7	8
Brown	4	7	9	2

The LP formulation of this AP is as follows:
Minimize $10X_{111} + 5X_{112} + 7X_{121} + 8X_{122} + 4X_{211} + 7X_{212} + 9X_{221} + 2X_{222}$
subject to

$$X_{111} + X_{112} + X_{121} + X_{122} = 1$$
$$X_{211} + X_{212} + X_{221} + X_{222} = 1$$
$$X_{111} + X_{112} + X_{211} + X_{212} = 1$$
$$X_{121} + X_{122} + X_{221} + X_{222} = 1$$
$$X_{111} + X_{121} + X_{211} + X_{221} = 1$$
$$X_{112} + X_{122} + X_{212} + X_{222} = 1$$

Two constraints are redundant, so we arbitrarily delete the 3rd and the 5th constraints.

Table 5. Initial tableau using the Push-and-Pull algorithm

BVS	X_{111}	X_{112}	X_{121}	X_{122}	X_{211}	X_{212}	X_{221}	X_{222}	RHS
?	1	1	1	1					1
?					1	1	1	1	1
?			1	1			1	1	1
?		1		1		1		1	1
Cost	10	5	7	8	4	7	9	2	0

The Push Phase takes four iterations resulting in the following optimal tableau:

Table 6. Optimal tableau of the AP

BVS	X_{111}	X_{122}	X_{212}	X_{221}	RHS
X_{112}	0.5	0.5	0.5	-0.5	0.5
X_{211}	0.5	-0.5	0.5	0.5	0.5
X_{222}	-0.5	0.5	0.5	0.5	0.5
X_{121}	0.5	0.5	-0.5	0.5	0.5
Cost	3	3	5	5	-9

The optimal solution of AP is $X_{112} = X_{211} = X_{222} = X_{121} = 0.5$, and all other $X_{ijk} = 0$. This implies that Adams should do job 1 on machine 2 for 50 percent of the time and job 2 on machine 1 for the other 50 percent of the time. Brown, likewise, should divide his time equally between job 1 on machine 1 and job 2 on machine 2. So each person spends 50 percent of his time on each job using both machines at the total cost of 9. Note that this problem does not have a totally unimodular matrix to ensure an integer solution. Solving this problem for an integer solution using Gomory's cutting planes [6] of LP yields solution $X_{111} = X_{222} = 1$, with inferior optimal value of 12.

The theoretical basis for the Push-and-Pull algorithm rests largely upon the total unimodularity of the coefficient matrix in the simplex tableau for TP, that is, the values of the coefficients are 0, -1, or 1 (see the appendix for a detailed proof). The *nominal problem* always has a unimodular constraint matrix. However, as observed in numerical example 2, this may not be the case when side-constraints are present. The violation of the total unimodularity does not affect the solution

procedure. This is a significant difference between existing methods dealing with side-constraints and our approach. If the solution is not integral, one may introduce cutting planes to ensure an integral solution, if so desired.

5. Computational Behaviour and Implementation

As mentioned earlier, there are well over 40 solution algorithms for the TP and AP. Many different algorithms are presented and coded in various sources (see, e.g., [19], [24], [40], [51]). The network simplex algorithm is consistently superior to the ordinary simplex and out-of-kilter algorithms. Most of this computational testing has been done on random problems generated by the well-known computer program, Netgen. To perform a meaningful comprehensive computational comparison, one needs to code all existing algorithms using similar data structure and processing. The computerized version of the algorithm must utilize the same compiler and the same platform. Then one can perform a thorough computational comparison either in terms of CPU, storage, or both. This is a major undertaking. Because of a lack of manpower resources at the present time, we are not able to present this type of comprehensive computational comparison.

The Push-and-Pull algorithm is pivotal like simplex-based algorithms and is more efficient than any other pivotal algorithm, such as the dual simplex and many other variants of simplex. A preliminary experiment to study computational behavior of the algorithm supports this assertion. Table 7 presents results comparing the proposed algorithm to the closely-related pivotal solution algorithm, namely the simplex algorithm, via the widely-used package Lindo on the VAX VMS 5.5-2. This computational comparison includes several small- to medium-sized published problems, which include some of the largest specific (non-random) problems of which we are aware. Note that Lindo is equipped with many helpful features such as anti-degeneracy devices, steepest-edge selection [38], [41], [84], crashing start, and so on. In spite of all these features in Lindo, the Push-and-Pull algorithm is more efficient in terms of number of iterations. As demonstrated by the results in Table 7, the proposed algorithm reduces the number by almost 50 percent as compared to Lindo.

5.1. Computer Implementation Issue

Up to now, the description we have given of the Push-and-Pull algorithm aimed at clarifying the underlying strategies and concepts. A second important aspect that we consider now is efficient computer implementation and data structures similar to those already developed by computer scientists for network-based solution algorithms [40]. Practical TP and AP may have thousands of constraints and even more variables. These problems have a large percentage of zero elements in the cost matrix (i.e., a sparse matrix). The proposed algorithm as described is not efficient for computer solution of large-scale problems since it updates all the elements of the

tableau at each iteration. Some useful modifications of the proposed algorithm as well as its efficient implementation will reduce the number of elementary operations and the amount of computer memory needed. This section describes how an efficient implementation of the Push-and-Pull algorithm capable of solving large-scale problems could be developed. Adaptation of sparse technology is motivated by the fact that, for a large sparse problem, an iteration of the revised simplex method takes less time than an iteration of the standard simplex GJP.

Table 7. Number of Iterations using the Push-and-Pull algorithm versus Lindo

Problem Type and Size	Simplex (Lindo)	Algorithm Push & Pull	Problem Source
TP 3 by 3	5	3	Shih [73]
TP 4 by 4	5	3	Taylor, example [77]
AP 4 by 4	5	3	Anderson et al., example [5]
TP 3 by 3	6	3	Davis et al., example [27]
AP 4 by 4	9	5	Taylor, example [77]
TP 3 by 3 with 3 side constrs	10	5	Shih, with 3 fixed capacitations [73]
TP 5 by 5	18	9	Shafaat and Goyal [72]
TP 5 by 6	19	9	Phillips and Garcia-Diaz, example [67]
Total	77	40	

Our implementation is a straightforward adaptation of the proposed algorithm through appropriate use of data structure, sparse technology, pivot strategy rules, and triangularity of the coefficient matrix. Computer implementation of all phases of the algorithm is discussed. We are concerned with computational efficiency, storage, accuracy, and ease of data entry.

5.2. Sparsity and Data Structure

The density of a matrix is the ratio of the number of non-zero entries in the matrix to the total number of entries in the matrix. A matrix that has a low density is said to be sparse. The sparse matrix technology is based on the following principles:

- only non-zero elements are stored,
- only those computations that lead to changes are performed,
- the number of fill-in (i.e., non-zero) elements is kept small,
- any unused locations (zero elements) can be used for storing of fill-ins (non-zeros produced later).

Sparse technology [56] enables us to reduce the total amount of work that is done in each iteration. Many schemes are available to avoid storing the zero entries of matrices. Since our calculations operate on tableaux by columns, we discuss a storage structure, called a linked list, that makes it easy to access data by columns. A piece of computer memory must contain three entries: one to hold an entry from the tableau, one to hold a row number, and one for the pointer to hold the next memory address. For example, the initial tableau of our numerical example 1 and its linked list for holding the data is as follows:

Table 8. Initial tableau of the numerical example 1

								RHS
	1	1	0	0	0	0	0	55
	0	0	1	1	1	0	0	80
	0	0	0	0	0	1	1	75
	1	0	0	1	0	1	0	100
	0	1	0	0	1	0	1	40
Cost	25	0	2	0	5	10	1	-3120

Table 9. The linked list for the coefficient matrix of example 1

Column	1	2	3	4	5	6	7	8
Entry	1	1						55
Row	1	1						1
Pointer								
Entry			1	1	1			80
Row			2	2	2			2
Pointer								
Entry						1	1	75
Row						3	3	3
Pointer								
Entry	1			1		1		100
Row	4			4		4		4
Pointer								
Entry		1			1		1	40
Row		5			5		5	5
Pointer								
Entry	25		2		5	10	1	-3120
Row	6		6		6	6	6	6
Pointer								

The empty spaces in Table 9 are unused locations. By having a matrix generator program in order to generate constraints and the objective function automatically at our disposal, we can store the non-zero values in a linked-list form. This scheme has minimal storage requirements and has proven to be very convenient for several important operations, such as permutation, addition, and multiplication of sparse matrices, in our solution algorithm and PA. For example, it can be shown that if b is a [1 by m] matrix and A is an [m by n] matrix that is stored as a linked list, then bA can be evaluated in d*m*n multiplications, where d is the density of A. An advanced variant of this scheme is Sherman's compression, which is useful in storing triangularized matrices. Such an approach can be applied in all phases of the Push-and-Pull solution algorithm and the needed PA, since the constrained coefficients matrix can be triangularized.

5.3. Matrix Triangularization

Working with the triangularization of matrix A rather than A itself has been shown to reduce the growth of new non-zero elements significantly. It is well known that matrix A is reducible to a unique triangular structure. The very structure of matrix A allows for rearranging it in a lower triangular form. The fill-in is zero when A is triangularized. The general idea is to permute the rows and columns of the basis to make the rearranged matrix lower triangular. There are two phases as follows:

Front phase: Among the unassigned rows, find the one with the minimum count of non-zeros. If this count is 1, assign the row and its (only) unassigned column to the front and repeat; otherwise, exit.

Rear phase: Among the unassigned columns, find the one with the minimum count of non-zeros. If this count is 1, assign the column and its (only) unassigned row to the rear and repeat; otherwise, exit.
Table 10 shows a given matrix at a given iteration:

Table 10. Row and column counts

	Col. 1	Col. 2	Col. 3	Col. 4	Count
Row 1	x	x	x		3
Row 2	x				1
Row 3			x		1
Row 4	x		x	x	3
Count	3	1	3	1	

Upon entering the Front phase, we find that row 2 is a singleton, so it is assigned to the front with column 1. Then, row 3 is assigned to the (new) front with column 3. Next the Rear phase is entered, and the singleton column 4 is assigned to the rear with row 4. Finally, the singleton column 2 is assigned to the rear with row 1.

The result is shown in Table 11:

Table 11. Result of triangularization

	Col.	Col. 3	Col. 4	Col. 2
Row 2	x			
Row 3		x		
Row 4	x	x	x	
Row 1	x	x		x

Push phase

In the Push phase one can construct a lower (block) triangular matrix by permuting matrix A with interchanging rows and columns. In order to preserve sparsity, an alternative approach for this reordering is block triangular diagonalization. This approach is attractive since under GJP, it maintains sparsity. In other words, this triangular structure is invariant under row or column interchanges to achieve sparsity within each block.

We were able to develop a Push phase employing the strategy of avoiding the use of artificial variables with large positive coefficients [15], [18], [26], [27]. In this phase, we start the initial tableau with a partially filled basic variable set. Therefore, the basic variable set is being developed. This particular crashing technique pushes toward a "vertex close to the optimal." Most commercial codes incorporate some other form of crashing techniques to provide the initial basis.

Pull phase

Using the matrix A, any iteration in this phase consists of solving $[A]x_B = b$ to determine the direction of movement (say, k) and updating the pivot column y_k by solving $[A]y_k = C_k$. This basic solution is used in the Pull phase to determine which current basic variable must leave the basic variable set (if needed) and to move to the new basic solution by performing the pivot operation (i.e., updating the A matrix). In the computer implementation, the inverse of the basis matrix is not needed [81]. If A is a triangular matrix, then the foregoing systems of equations can be solved easily by one forward and two backward substitutions, respectively [60].

We can also achieve efficiency in performing GJ operations in this phase from another point of view known as A = LU factorization of A (see, e.g., [69], [75]). Matrices L and U are lower and upper triangular matrices for which the inverses can be computed easily and then solved for the above systems of equations by inversion. In this approach L^{-1} is stored as a sequence of "elementary" matrices and U is stored as a permuted upper triangular matrix.

In [33], [79], [80] some ingenious variant of the Bartles-Golub triangular factored updating procedure to exploit sparsity is proposed. This approach could lead to a decrease in accuracy through the introduction of round-off error. Working with integers (not real numbers), however, and not using any large numbers such as big-M, the computations would be precise and reliable. Since the [A] elements are 0, 1 or -1, the column ratio (C/R) division operations can be replaced by a simple comparison operation.

The steepest-edge pivoting rule, also referred to as the largest-decrease rule, chooses the candidate whose entrance into the basis brings about the largest decrease in the objective function. This usually reduces the number of iterations, but may increase total computing time. A large fraction of the time needed to perform an iteration can be spent in retrieving the data. However, this approach is best suited to the Push-and-Pull algorithm, given the sparsity of TP and AP models after the row-column reduction.

Our discussion of computer implementation is by necessity and in no way is comprehensive. There are many interesting variants of what we mentioned that deserve further study to reduce the total number of iterations (the sum of all the iterations in all phases) and the amount of work that is done in each iteration.

6. Managing The Cost Uncertainties

The cost vector C, is composed of elements C_{ij} appearing row-wise, i.e., $C = \{C_{11}, C_{12}, ..., C_{1n}, C_{21}, C_{22}, ..., C_{2n}, C_{m1}, C_{m2}, ..., C_{mn}\}$. The cost parameters are more or less uncertain. We are likely to be unsure of their current values and even more uncertain about their future values at the time when the solution is to be implemented. A time lag between the development and application of the model could create such uncertainty. In a volatile global market, an ever-changing currency exchange rate could affect the cost of stationing executives in various countries in an AP. Therefore, managers are concerned with the stability of the optimal solution under uncertainty of the estimated cost parameters.

Uncertainty is the prime reason why PA is helpful in making decisions. We can use PA to give us information like: the robustness of an optimal solution; critical values, thresholds, and break-even values where the optimal strategy changes; sensitivity of important variables; sub-optimal solutions; flexible recommendations; the values of simple and complex decision strategies; and the "riskiness" of a strategy or scenario. For a more detailed discussion of PA in practice, see [64].

Current approaches to deal with cost uncertainties include:

Scenario Analysis - In this approach one assumes scenarios (a combination of possible values of uncertain costs) and solves the problem for each. By solving the

problem repeatedly for different scenarios and studying the solutions obtained, a manager observes sensitivity and heuristically decides on a subjective approximation.

Worst-Case Analysis - This technique attempts to account for putting safety margins into the problem in the planning stage.

Monte-Carlo Approach - In this approach, stochastic models assume that the uncertainty in cost is known by its distribution (see, e.g., [54]).

Reoptimization - The combinatorial and the network-simplex approaches in network-based SA can only handle integer changes in C_{ij}. The combinatorial approach requires the solution of a completely different maximum flow network problem for each unit change in any cost coefficient C_{ij}, until infeasibility is reached. The network-simplex approach can handle any number of unit changes [3]. Neither of these two approaches produce any managerially-useful prescriptive ranges for sensitivity analysis. A prerequisite for these approaches is to have an anticipated direction of change. From a manager's point of view, anticipation of possible scenarios may not be an easy task. Application of a transformation scheme to attain the integrality condition for the nominal problem in the network-based algorithm makes PA too complicated to interpret.

PA deals with a collection of questions related to so-called "what-if" problems in preservation of the current optimal strategy generated by the proposed solution algorithm. PA starts as soon as one obtains the optimal solution to a given nominal problem. There are strong motivations for a manager to perform PA for the parameters C_{ij} to:

- help determine the responsiveness of the solution to changes or errors in cost values,
- adapt a model to a new environment with an adjustment in these parameters,
- provide systematic guidelines for allocating scarce organizational resources to data collection and data refinement activities by using the sensitivity information,
- determine a cost perturbed region in which the current strategic decision is still valid.

Although all aspects of PA are readily available for LP models, even OSA is rarely performed on TP or AP. While the SS method, the Hungarian method, or other traditional network-based solution algorithms may be efficient in solving a TP and AP, they are not designed to perform PA. The final solutions generated by the SS method for TP or the Hungarian method for AP do not contain enough information to perform PA. If the needed sensitivity information were readily available in

these algorithms, the operations research and management science textbooks would have covered the SA of the TP and AP. Considerable additional work is involved in obtaining the needed information. Moreover, one must be careful in using the existing LP computer packages for performing SA for AP as the optimal solutions are degenerate.

The basis inverse matrix is an essential prerequisite to SA. The SS and Hungarian methods do not provide this matrix directly. One may suggest using the optimal solution obtained by these methods, or any other traditional algorithm, to construct the basis matrix. This basis matrix can then be inverted to obtain the basis inverse matrix and other needed information. However, in addition to these extra computational costs, there is the problem of recognizing the basic variables contained in the optimal solution since the optimal solution may be degenerate. This is always the case for the AP since the number of positive variables is much smaller than the size of the BVS, due to a degenerate optimal solution. Also, note that it is this degeneracy of the optimal solution which causes the regular simplex to provide very misleading SA for the AP (and probably the TP).

Two recent developments in network-based SA, the combinatorial approach and the network approach (including the SS and the Hungarian methods), can handle only integer changes [3]. The combinatorial approach requires the solution of a completely different problem, until infeasibility is reached. Specifically, it requires a maximum flow problem for each unit change in a cost coefficient. More complex changes such as simultaneous SA must be dealt with a sequence of this simple change. The network algorithm approach to SA can handle any number of unit changes. The change, however, is restricted to one change at a time as in the combinatorial approach. These limitations are caused by the inability of these solution algorithms (SS, Hungarian, network-based, and others) to handle a larger scope of SA. The network community has put much effort on developing efficient solution algorithms at the expense of SA.

Neither of these approaches produces any managerially useful prescriptive ranges on costs for the SA. Additionally, a prerequisite for some of these approaches is to have an anticipated direction of changes. From a managerial point of view, anticipation of possible scenarios may not be an easy task.

Define the perturbed TP to be

$$\text{Minimize } \sum\sum(C_{ij} + C'_{ij})X_{ij} \tag{4}$$

with the same constraints as before and the admissibility condition $C'_{ij} \geq -C_{ij}$. The PA starts as soon as we obtain the final tableau of the Push-and-Pull algorithm. At this stage we need to introduce some more notations as shown in Table 12. The initial tableau is partitioned into B, the BV coefficients, and N, the NB

coefficients, as they appear in the final tableau, and the RHS column. Note that, as it progresses, the solution algorithm removes the nonbasic columns.

Table 12. Necessary components for cost perturbation analysis from the final tableau

BVS	X_N	RHS
X_B	[A]	b
Cost	$C_N^* = C_N - C_B \cdot$ [A]	

As mentioned earlier, the occurrence of alternative solutions is rare and can easily be avoided by a slight perturbation of the cost parameters. Note, however, that if a decision maker wants alternate optimal solutions and related SA, he needs to generate many (a combinatorial problem) optimal solutions to get basis matrices, B, if using any other solution methods. Then each B has to be converted and multiplied by the N matrix. However, [A] $= B^{-1}N$ is provided by the Push-and-Pull solution algorithm as a by-product. We can easily generate other distinct [A]'s by using this one to generate all other solutions, if needed.

Having obtained the necessary information from the final tableau, cost PA can be started. To find the critical region, that is, the largest set of perturbed costs for which the current solution remains optimal, we must find the allowable changes in the cost coefficients. The necessary and sufficient condition to maintain the current optimal strategy is that $C_N^* \geq 0$, where 0 stands for a zero vector with the appropriate dimension. Let θ denote the set of perturbed costs $C_{ij}^{'}$ to maintain optimality:

$$\theta = \{C_{ij}^{'} | C_N^* \geq 0 \qquad and \qquad C_{ij}^{'} \geq -C_{ij}\}. \tag{5}$$

The set θ is non-empty since it contains the origin $C_{ij}^{'} = 0$, for all i, j. It is convex with linear boundary functions, by virtue of GJP operations. This set can be used to check whether the given perturbed (dependent or independent) cost coefficients have the same optimal solution as the nominal problem. To help clarify what we have done up to now, consider the numerical example 1. From the final tableau in table 3, we have:

$$[A] = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 0 & 1 & 1 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The perturbed cost vector is:

$$C + C^{'} = [5 + C^{'}_{11}, 30 + C^{'}_{12}, 12 + C^{'}_{13}, 20 + C^{'}_{21}, 18 + C^{'}_{22}, 30 + C^{'}_{23}, 15 + C^{'}_{31}, 25 + C^{'}_{32}, 23 + C^{'}_{33}],$$

with

$C_B = [5+C^{'}_{11},\ 18+C^{'}_{22},\ 15+C^{'}_{31},\ 25+C^{'}_{32},\ 12+C^{'}_{13}]$, and
$C_N = [30+C^{'}_{12},\ 20+C^{'}_{21},\ 30+C^{'}_{23},\ 23+C^{'}_{33}]$.

The perturbed set is

$$\begin{aligned}\theta = \{C^{'}_{ij} |\ & 15 + C^{'}_{12} - C^{'}_{11} + C^{'}_{31} - C^{'}_{32} \geq 0, \\ & 12 + C^{'}_{21} - C^{'}_{22} - C^{'}_{31} + C^{'}_{32} \geq 0, \\ & 15 + C^{'}_{23} + C^{'}_{11} - C^{'}_{22} - C^{'}_{31} + C^{'}_{32} - C^{'}_{13} \geq 0, \\ & 1 + C^{'}_{33} + C^{'}_{11} - C^{'}_{31} - C^{'}_{13} \geq 0; \\ & C^{'}_{11} \geq -5, C^{'}_{12} \geq -30, C^{'}_{13} \geq -12, \\ & C^{'}_{21} \geq -20, C^{'}_{22} \geq -18, C^{'}_{23} \geq 30, \\ & C^{'}_{31} \geq -15, C^{'}_{32} \geq -25, C^{'}_{33} \geq -23\}\end{aligned} \tag{6}$$

The set θ can be used to check whether a specific scenario has the same optimal basis as the nominal problem. For example, assume that the perturbed cost vector for a given scenario is: $C + C^{'} = \{9, 22, 15, 13, 16, 15, 17, 28, 26\}$ with $C^{'} = \{4, -8, 3, -7, -2, -15, 2, 3, 3\}$. By substituting the values of $C^{'}_{ij}$, one can easily check that the current solution is still optimal.

For our numerical example 2 of AP, the critical region is:

$$\begin{aligned}\theta = \{C^{'}_{ijk} |\ & 3 + C^{'}_{111} - 0.5C^{'}_{112} - 0.5C^{'}_{211} + 0.5C^{'}_{222} - 0.5C^{'}_{121} \geq 0, \\ & 3 + C^{'}_{122} - 0.5C^{'}_{112} + 0.5C^{'}_{211} - 0.5C^{'}_{222} - 0.5C^{'}_{121} \geq 0, \\ & 5 + C^{'}_{212} - 0.5C^{'}_{112} - 0.5C^{'}_{211} - 0.5C^{'}_{222} + 0.5C^{'}_{121} \geq 0, \\ & 5 + C^{'}_{221} + 0.5C^{'}_{112} - 0.5C^{'}_{211} - 0.5C^{'}_{222} - 0.5C^{'}_{121} \geq 0; \\ & C^{'}_{111} \geq -10, C^{'}_{112} \geq -5, C^{'}_{121} \geq -7, \\ & C^{'}_{122} \geq -8, C^{'}_{211} \geq -4, C^{'}_{212} \geq -7, \\ & C^{'}_{221} \geq -9, \text{ and } C^{'}_{222} \geq -2\}.\end{aligned} \tag{7}$$

7. Special Cases of Sensitivity Analysis

The PA construction given in the previous section is the most general one that can handle the simultaneous and independent changes in the cost parameters. In this section we derive some special cases of the sensitivity analysis in a direct method. The PA set would generate the same results.

7.1. Parametric Perturbation Analysis

Parametric sensitivity analysis is of particular interest whenever there is dependency among the cost parameters. This analysis can be considered as simultaneous changes in a given direction. Define a perturbation vector P specifying a perturbed direction of the cost coefficients. Introducing a scalar parameter $\delta \geq 0$ (thus, perturbation analysis,) we would like to find out how far we can move in the direction of P, δ being the step size, while still maintaining optimality of the current assignment.

Step 1 : Identify P_N and P_B, sub-vectors of P corresponding to the NB and BVS, respectively, as they appear in the final tableau of the nominal problem.

Step 2 : Define $S = \{i \to j | (P_N - P_B.[A]) < 0\}$.

Step 3 : If $S = \Phi$, then $\delta^{'} = \infty$. Go to Step 5.

Step 4 : Calculate $\delta^{'} = \min[-(Old\, C_N^*)_{ij}/(P_N - P_B \cdot [A])_{ij}]$ over all $\{i \to j\} \in S$.

Step 5 : Determine $\delta^{'} = \min[\delta^{'}, \min(\delta | C_{ij} - \delta P_{ij} \geq 0)]$.

Step 6 : New C_N^* = Old $C_N^* + \delta(P_N - P_B \cdot [A])$ for any $\delta \in [0, \delta^{'}]$.

For our numerical example 1, let us assume that the cost vector C is perturbed along vector P = {0, -1, 1, 1, 1, -2, -1, 1, 1}, i.e., the perturbed cost coefficient vector is {5, 30-δ, 12+δ, 20+δ, 18+δ, 30-2δ, 15-δ, 25+δ, 23+δ}.

The perturbed coefficients for non-basic and basic coefficients are P_N = (-1, 1, -2, 1) and P_B= (0, 1, -1, 1, 1) respectively. Thus, $P_N - P_B$.[A] = (-1, 1, -2, 1) - (2, -1, 0, 0) = (-3, 2, -2, 1). This gives $S = \{1 \to 2, 2 \to 3\}$, which results in $\delta^{'} = \min[-15/-3, -15/-2] = 5$. Since the admissibility condition of Step 5 is satisfied for all $\delta \in [0, 5]$, the current optimal basis remain optimal for any $\delta \in [0, 5]$. Clearly, for $\delta = 5$ there is an alternate optimal solution which can be obtained by bringing X_{12} into basis. By one additional GJ row operation, we find the alternate optimal solution as: $X_{12} = 15$, $X_{22} = 80$, $X_{31} = 70$, $X_{32} = 5$, and $X_{13} = 40$.

7.2. Ordinary Sensitivity Analysis

From a managerial point of view, anticipation of possible scenarios or directions of change may not be possible. In this sub-section, we find the range for any particular arc cost, holding all other costs unchanged. Ordinary sensitivity analysis, OSA, is very popular and readily available in LP. One cannot, however, use existing LP computer packages for performing SA for these problems when the optimal solution is degenerate. The other references present in current literature require significant additional computation beyond the solution algorithm.

OSA is a special case of parametric analysis where we would like to find out how far we can move in the direction of any one of the axes in the parametric space C'_{ij}. Here, P is a unit row vector or its negative, depending on whether we want to compute an upper or lower limit. The step size δ is the amount of increase or decrease in that direction. Alternately, note that we can also find the allowable changes for any particular cost C'_{ij} by setting all other costs to zero in the set θ , equation (6). The results are as follows:

	Lower Limit	Upper Limit
C'_{11}	-1	15
C'_{12}	-15	∞
C'_{13}	-12	1
C'_{21}	-12	∞
C'_{22}	-18	12
C'_{23}	-15	∞
C'_{31}	-15	1
C'_{32}	-12	15
C'_{33}	-1	∞

Similarly, for numerical example 2 for AP, these limits are obtained from the corresponding set θ , equation (7), by letting all $C'_{ijk} = 0$, except the one for which we are calculating the bounds.

7.3. The 100% Rule

The above analysis is for one-change-at-a-time. Suppose we want to find the simultaneous allowable increases in all cost coefficients. Bradley et al [17] discuss the use of the 100 percent rule for simultaneous increase or decrease of all costs. This rule is based on the ordinary sensitivity limits. The 100 percent rule says that optimal basis will be preserved if

$$\sum\sum C_{ij}^{'}/C_{ij} \leq 1 \tag{8}$$

where the sum is over all i and j and the denominators (C_{ij}) are the allowable increases from the ordinary sensitivity analysis. That is, as long as the sum of all of the percentages based on the allowable changes for each cost coefficient is less than 100 percent, the current optimal basis remains unchanged. For the above example, the current routings are optimal as long as cost increases are such that $C_{11}^{'}/15 + C_{13}^{'} + C_{22}^{'}/12 + C_{31}^{'} + C_{32}^{'}/15 \leq 1$. This condition is sufficient and not necessary. Similarly, the application of the 100 percent rule when decreasing all cost coefficients provides $\sum\sum C_{ij}^{'}/-C_{ij} \leq 1$, where the sum is over all i and j and the denominators are the allowable decreases from the ordinary sensitivity analysis with a similar interpretation. Note that the upper limit and lower limit must be rounded down and up respectively.

8. Models with Side-Constraints

In real-life situations, it is common for a few side-constraints to evolve during the time period of development and implementation of an optimal solution [2], [46], [71], [73]. For example, there could be a limit on the shipment along a particular route or combination of routes. Alternately, there could be a constraint on one route in relation to other routes. A general side-constraint is to determine the amount of shipment from source i to destination j under the conditions that $L_{ij} \leq \sum a_{ij}X_{ij} \leq U_{ij}$ where X_{ij} denotes the number of units shipped from i to j and L_{ij}, U_{ij} and a_{ij} are constants. Without loss of generality, we assume the L_{ij} to be zero. If an $L_{kt} > 0$, then a change in variable X_{kt} reduces the lower bound to zero. We provide a method to accommodate these constraints through the Push-and-Pull algorithm.

8.1. Method Capacitated

Step 1 : Ignore the upper bounds. Solve this nominal problem by Push-and-Pull algorithm.

Step 2 : Derive the final tableau of the nominal TP.

Step 3 : Check if all conditions $\sum a_{ij}X_{ij} \leq U_{ij}$ are satisfied. If yes, go to step 7.

Step 4 : Pick the basic variable X_{kt} for which $U_{ij} - \sum a_{ij}X_{ij}$ is the largest. Add an open row with constraint $\sum a_{ij}X_{ij} = U_{kt}$.

Step 5 : Pivot on X_{kt} column so X_{kt} remains basic. By construction, RHS is infeasible with a negative entry in the open row.

Step 6 : Perform the Pull phase of the Push-and-Pull algorithm. Go to step 3.

Step 7 : This is the optimal tableau. STOP. Obtain the solution.

Consider the example of Table 1 with constraint $X_{31} \leq 2X_{32}$. This constraint is not satisfied in the optimal solution of the nominal TP, Table 3. After performing steps 4 and 5 of the Method Capacitated, we obtain the following table:

Table 13. Tableau after adding an open row and adjusting X_{31} and X_{32}

BVS	X_{12}	X_{21}	X_{23}	X_{33}	RHS
X_{11}	1		-1	-1	15
X_{22}		1	1		80
X_{31}	-1	1	1	1	55
X_{32}	1	-1	-1		20
X_{13}			1	1	40
?	3	-3	-3	-1	-15
Cost	15	12	15	1	-3320

Note that due to the unrestricted nature of the given constraints, the non-zero entries of a tableau in Method Capacitated could deviate from the regular pattern of being 1 and -1 only. This deviation from unimodularity does not affect the solution algorithm. After performing the Pull phase to bring X_{33} into BVS, we obtain the following optimal tableau:

Table 14. Optimal tableau of TP with side-constraints

BVS	X_{12}	X_{21}	X_{23}	RHS
X_{11}	-2	3	2	30
X_{22}		1	1	80
X_{31}	2	-2	-2	40
X_{32}	1	-1	-1	20
X_{13}	3	-3	-2	25
X_{33}	-3	3	3	15
Cost	18	9	12	-3335

8.2. Numerical Example for Fixed Upper Bounds in TP

Shih [73] discusses the special case of $L_{ij} \leq X_{ij} \leq U_{ij}$ where X_{ij} are bound by constants only and offers a solution by modifying the SS method. Though conceptually the modification Shih offers is simple, creating new tables and finding new alternative paths repeatedly, turns out to be cumbersome. We illustrate the steps of Method Capacitated further by walking through the example of Table 1 with fixed upper bound constraints $X_{22} \leq 70$, $X_{31} \leq 50$, $X_{21} \leq 8$.

Step 1: Perform the row and column cost reductions. Solve the TP by using Push-and-Pull algorithm.

Step 2: Obtain the final tableau as presented in Table 3 in the previous section.

Step 3: Check feasibility. Constraints $X_{22} \leq 70$ and $X_{31} \leq 50$ are violated.

Step 4: Open a row with $X_{22} = 70$, corresponding to the most violated constraint $X_{22} \leq 70$.

Step 5: Adjust the column for the basic variable X_{22}.

Table 15. Modified tableau after adjusting column X_{22}

BVS	X_{12}	X_{21}	X_{23}	X_{33}	RHS
X_{11}	1		-1	-1	15
X_{22}		1	1		80
X_{31}	-1	1	1	1	55
X_{32}	1	-1	-1		20
X_{13}			1	1	40
?		-1	-1		-10
Cost	15	12	15	1	-3320

Step 6: Pull phase - Since X_{21} has lower cost, enter X_{21} in the open row.

Table 16. Revised tableau with X_{21} as a BV

BVS	X_{12}	X_{23}	X_{33}	RHS
X_{11}	1	-1	-1	15
X_{22}				70
X_{31}	-1		1	45
X_{32}	1			30
X_{13}		1	1	40
X_{21}		1		10
Cost	15	3	1	-3440

Check feasibility. Constraint $X_{21} \leq 8$ is violated. Set $X_{21} = 8$. Repeat steps 4 through 6.

Table 17. Optimal tableau with fixed upper bounds

BVS	X_{12}	X_{33}	RHS
X_{11}	1	-1	17
X_{22}			70
X_{31}	-1	1	45
X_{32}	1		30
X_{13}		1	38
X_{21}			8
X_{23}			2
Cost	15	1	-3446

Step 7: This tableau is optimal with solution $X_{11} = 17$, $X_{22} = 70$, $X_{31} = 45$, $X_{32} = 30$, $X_{13} = 38$, $X_{21} = 8$, and $X_{23} = 2$ at the cost of 3446.

9. Degenerate Optimal Solution

The effect of degenerate optimal solution on SA has been studied by many research workers (see, e.g., [34], [53], [52]) and is well known. However, to the best of our knowledge, no algorithm or LP package exists for cost SA in case of an optimal degenerate problem. The researchers have completely overlooked the degeneracy phenomenon, a common occurrence in a TP, in cost SA. Lindo and Netsolve give wrong cost sensitivity results if the optimal solution of a TP turns out to be degenerate, which is always the case with an AP. Note that the set θ given by equation (5) is valid for a non-degenerate problem only, that is, when RHS > 0. For a degenerate optimal solution, we modify the procedure for solving the perturbed problem (4) as follows:

Step 1 : Let Z = count of zeros in the RHS of the optimal tableau. Set I = Z+1, i = 0.

Step 2 : Let i = i+1. Determine θ_i for the current optimal tableau.

Step 3 : If i = I, go to Step 8.

Step 4 : PR = pick a degenerate row (i.e., a row with RHS=0) to exit.

Step 5 : PC = pick the column with the largest negative row ratio, i.e., cost row/PR.

Step 6 : Generate the next distinct final tableau.

Step 7 : Go to Step 2.

Step 8 : Determine $\theta = \cap\theta_i$ i, $i \in I$.

Note that as before, we can use θ for any scenario analysis. The set can also be used for parametric, or ordinary cost SA. Alternately, we find $\delta^{'}$ for parametric PA, and lower and upper limits for OSA, for each distinct final tableau and then use the corresponding intersection for analysis. We illustrate using the AP from Anderson et al. [5].

9.1. Numerical Example 3

Cost matrix of the AP:

10	15	9
9	18	5
6	14	3

We solve this AP using the Push-and-Pull algorithm. The following final tableau is obtained.

Table 18. Final degenerate tableau

BVS	X_{21}	X_{22}	X_{13}	X_{32}	RHS
X_{31}	1	1	-1	1	1
X_{23}	1	1			1
X_{33}	-1	-1	1		0
X_{11}		-1	1	-1	0
X_{12}		1		1	1
C_{ij}	1	5	2	3	

There are no alternative optimal strategies. As expected, however, the optimal tableau is degenerate. The other optimal tableaux are as follows:

Table 19. Second degenerate final tableau

BVS	X_{33}	X_{22}	X_{13}	X_{32}	RHS
X_{31}	1			1	1
X_{23}	1		1		1
X_{21}	-1	1	-1		0
X_{11}		-1	1	-1	0
X_{12}		1		1	1
C_{ij}	1	4	3	3	

Table 20. Third degenerate final tableau

BVS	X_{21}	X_{22}	X_{13}	X_{11}	RHS
X_{31}	1			1	1
X_{23}	1	1			1
X_{33}	-1	-1	1		0
X_{32}		1	-1	-1	0
X_{12}			1	1	1
C_{ij}	1	2	5	3	

The critical regions, θ_i's, based on these final tableaux are:

$$\begin{aligned}\theta_1 = \{C'_{ij} | 1 + C'_{21} - C'_{31} - C'_{23} + C'_{33} \geq 0,\\ 5 + C'_{22} - C'_{31} - C'_{23} + C'_{33}C'_{11} - C'_{12} \geq 0,\\ 2 + C'_{13} + C'_{31} - C'_{33} - C'_{11} \geq 0,\\ 3 + C'_{32} - C'_{31} + C'_{11} - C'_{12} \geq 0\},\end{aligned}$$

$$\begin{aligned}\theta_2 = \{C'_{ij} | 1 + C'_{33} - C'_{31} - C'_{23} + C'_{21} \geq 0,\\ 4 + C'_{22} - C'_{21} - C'_{11} - C'_{12} \geq 0,\\ 3 + C'_{13} - C'_{23} + C'_{21} - C'_{11} \geq 0,\\ 3 + C'_{32} - C'_{31} + C'_{11} - C'_{12} \geq 0\},\end{aligned}$$

$$\begin{aligned}\theta_3 = \{C'_{ij} | 1 + C'_{21} - C'_{31} - C'_{23} + C'_{33} \geq 0,\\ 2 + C'_{22} - C'_{23} + C'_{33} - C'_{32} \geq 0,\\ 5 + C'_{13} - C'_{33} + C'_{32} - C'_{12} \geq 0,\\ 3 + C'_{11} - C'_{31} + C'_{32} - C'_{12} \geq 0\},\end{aligned}$$

and finally, $\theta = \theta_1 \cap \theta_2 \cap \theta_3$. The ordinary sensitivity limits based on θ are:

	Lower Limit	Upper Limit
C'_{11}	-3	2
C'_{12}	-∞	3
C'_{13}	-2	∞
C'_{21}	-1	4
C'_{22}	-2	∞
C'_{23}	-∞	1
C'_{31}	-2	1
C'_{32}	-3	∞
C'_{33}	-1	2

The 100% rule limits for $C'_{ij} \geq 0$ and $C'_{ij} \leq 0$ can be obtained directly from these above limits. Similarly, to carry out parametric analysis given any perturbation vector P, we first find δ'_1, δ'_2, δ'_3. The current basis are optimal for any step size $\delta \in [0, \delta']$ where $\delta' = \min\{\delta'_1, \delta'_2, \delta'_3\}$.

10. Concluding Remarks

We have proposed a general-purpose unified algorithm to solve the classical TP and AP. The algorithm provides one treatment for both problems, in place of the SS and Hungarian methods. The proposed algorithm solves these problems with a warm-start and uses Gauss-Jordan pivoting only. It is computationally practical in the sense that it does not require any slack/surplus variables (as in simplex) or any artificial variables (as in dual simplex). In addition, the Push-and-Pull algorithm and the proposed PA have the advantage of being easy for managers to understand and implement. The results empower the manager to assess and monitor various types of cost uncertainties encountered in real-life situations. The final solution tableau provides the information required to handle any special cases of side-constraints. For such a case, it is not necessary to restart the algorithm. If the current solution does not satisfy some of the side-constraints, the method brings the most violated constraint into the final tableau and uses the "catch-up" operations of the Pull phase to generate the updated final tableau. This catch-up capability is also useful in coping with structural changes, such as the deletion of a route that becomes inaccessible due to a storm, construction, or bad road conditions. Finally, the algorithm is free from pivotal degeneracy (cycling) and can solve AP of higher dimensions than the traditional 2-dimensional AP.

The proposed solution algorithm deals effectively with change and chance and is compatible with other solution algorithms. The PA in this paper is efficient since it uses the same data and the same manipulation as the solution algorithm. All the information needed to carry out PA of cost coefficients is readily provided by the final

tableau. In contrast to OSA, our algorithm allows for simultaneous, independent, or dependent change of the cost coefficients from the estimated values. Unlike the existing network-based SA, the PA provides prescriptive uncertainty ranges. As long as the estimated parameters remain within ranges specified by the PA, the current optimal strategy remains optimal. In cases where the optimal solution is not unique, one must be careful in using any part of the PA results, because the results may not be correct for an alternate optimal solution. We demonstrate the applications of PA when an optimal solution is degenerate and provide modified steps [44].

From a managerial point of view, the PA results provide an assessment and analysis of the stability of the optimal strategy by monitoring the admissible range that preserves it. This permits evaluation of the impact of uncertainty in the data. PA gives a manager more leverage in allocating scarce resources. Monitoring the admissible ranges of PA that preserve the current optimal strategy aids in determining how resources should be applied. Information about departure from these limits allows a manager to anticipate the consequences and to pre-determine back-up strategies. The results empower a manager to assess, analyze, monitor, and manage various types of cost uncertainties in all phases of the model, namely design, solution, and implementation.

The proposed approach can also provide a rich modeling environment for strategic management of complex transportation and assignment problems using Monte Carlo simulation experiments. In this treatment, one may select *a priori* cost coefficients for each route or assignment with the support domain produced by the PA. The stability of optimal shipments with respect to changes in supply and demand in TP is given in [1].

Computational results comparing the proposed algorithm to the closely-related pivotal solution algorithm, the simplex, via the widely-used package Lindo, indicate that the Push-and-Pull algorithm reduces the number of iterations by almost 50 percent as compared to the simplex. Future work should develop efficient codes to implement this approach for computational studies, design and experiment to compare the Push-and-Pull algorithm to existing algorithms. An immediate extension would apply our approach to a study of the more-for-less paradox, that is, given an optimal solution, is it possible to find a less (or equivalent) cost solution by shipping more total goods under the restriction that the same amount of goods are shipped from each origin and to each destination and all shipment costs are non-negative [70]. Another possible direction for future research would be design of an intelligent computer-assisted system of various parts of PA for the decision maker.

Acknowledgments

We are most appreciative to the referees for their useful comments and suggestions. Dr. Arsham's work is supported by NSF Grant CCR-9505732.

References

1. V.G. Adlakha and H. Arsham. Distribution-routes stability analysis of the transportation problem. *Optimization*, 43, 1998. in press.
2. V. Aggarwal. A Lagrangean-relaxation method for the constrained assignment problem. *Computers and Operations Research*, 12:97–106, 1985.
3. R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, New Jersey, 1993.
4. M. Amini. Vectorization of an auction algorithm for linear cost assignment problem. *Computers and Industrial Engineering*, 24:141–149, 1994.
5. D. Anderson, D. Sweeny, and T. William. *An Introduction to Management Science*. West Publishing Co, St. Paul, MA, 1997.
6. E. Balas, S. Ceria, G. Gornuejols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.
7. E. Balas and M.J. Saltzman. An algorithm for the three-index assignment problem. *Operations Research*, 39:207–302, 1991.
8. M.L. Balinski. Signature methods for the assignment problem. *Operations Research*, 33:207–305, 1985.
9. M.L. Balinski. A competitive (dual) simplex method for the assignment problem. *Mathematical Programming*, 34:125–141, 1986.
10. R. Barr, F. Glover, and D. Klingman. The alternating basis algorithm for assignment problems. *Mathematical Programming*, 13:1–13, 1977.
11. M.S. Bazaraa, J.J. Jarvis, and H. Sherali. *Linear Programming and Networks Flows*. Wiley, New York, 1990.
12. D.P. Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21:152–171, 1981.
13. D.P. Bertsekas. The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces*, 20:133–149, 1990.
14. D.P. Bertsekas and P. Tseng. The relax codes for linear minimum cost network flow problems. *Annals of Operations Research*, 13:125–190, 1988.
15. R. Bixby. Implementing the simplex method: The initial basis. *ORSA Journal of Computing*, 4:267–282, 1992.
16. E. Boyd. Fenchel cutting planes for integer programs. *Operations Research*, 40:53–64, 1994.
17. S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematics Programming*. Addison-Wesley, Reading, MA, 1977.
18. J. Camm, A. Raturi, and S. Tsubakitani. Cutting big M down to sizes. *Interfaces*, 20:61–66, 1990.
19. G. Carpaneto, S. Martello, and P. Toth. Algorithms and codes for the assignment problem. *Annals of Operations Research*, 13:193–223, 1988.
20. G. Carpaneto and P. Toth. Algorithm for the solution of the assignment problem for sparse matrices. *Computing*, 31:83–94, 1983.
21. A. Charens and M. Kress. Two simple applications of the unimodularity property. *Operations Research Letters*, 14:257–260, 1993.
22. CPLEX Optimization, Inc. *Using the CPLEX Callable Library*. CPLEX Optimization, Inc., Incline Village, NV, 1996.
23. Y. Crama and F. Spiekma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European Journal of Operational Research*, 60:273–279, 1992.

24. W. Cunningham. A network simplex method. *Mathematical Programming*, 11:105–116, 1976.
25. W. Cunningham and J. Klincewicz. On cycling in the network simplex method. *Mathematical Programming*, 26:182–189, 1983.
26. G. Dantzig and M. Thapa. *Linear Programming.* Springer-Verlag, New York, 1997.
27. K. Davis, P. McKeown, and T. Rakes. *Management Science, An Introduction.* Kent, Boston, MA, 1986.
28. U. Derigs. *Programming in Networks and Graphs.* Springer-Verlag, New York, 1988.
29. U. Derigs and A. Metz. An efficient labeling technique for solving sparse assignment problems. *Computing*, 36:301–311, 1986.
30. U. Derigs and A. Metz. An in-core/out-of-core method for solving large scale assignment problems. *Optimization*, 30:181–195, 1986.
31. M. Engquist. A successive shortest path algorithm for the assignment problem. *INFOR*, 20:370–384, 1982.
32. M. Florian and M. Klein. An experimental evaluation of some methods of solving the assignment problem. *INFOR*, 8:101–106, 1970.
33. J. Forrest and J. Tomlin. Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1979.
34. T. Gal. Shadow prices and sensitivity analysis in linear programming under degeneracy. *OR Spektrum*, 8:59–71, 1986.
35. K. Gilbert and R. Hofstra. Multidimensional assignment problems. *Decision Sciences*, 19:306–321, 1988.
36. B. Gillett. *Introduction to Operations Research: A Computer-oriented Algorithmic Approach.* McGraw-Hill, New York, 1976.
37. D. Goldfarb. Efficient dual simplex algorithms for the assignment problem. *Mathematical Programming*, 33:187–203, 1985.
38. D. Goldfarb and J. Reid. A practicable steepest-edge simplex algorithm. *Mathematical Programming*, 12:361–371, 1977.
39. S.K. Goyal. Improving VAM for unbalanced transportation problems. *Journal of the Operational Research Society*, 35:1113–1114, 1984.
40. M. Grigoriadis. An efficient implementation of network simplex method. *Mathematical Programming*, 26:83–111, 1986.
41. B. Hattersley and J. Wilson. A dual approach to primal degeneracy. *Mathematical Programming*, 42:135–145, 1988.
42. M.S. Hung and W.O. Rom. Solving the assignment problem by relaxation. *Operations Research*, 28:969–982, 1980.
43. J. Intrator and W. Szwarc. An inductive property of transportation problem. *Asia-Pacific Journal of Operational Research Society*, 5:79–83, 1988.
44. B. Jansen. Sensitivity analysis in linear programming: just be careful! *European Journal of Operational Research*, 101:15–28, 1997.
45. R. Jonker and A. Volgenant. Improving the Hungarian assignment algorithm. *Operations Research Letters*, 5:171–175, 1986.
46. K. Jornsten and M. Nasberg. A new Lagrangian relaxation approach to the generalized assignment problem. *European Journal of Operational Research*, 27:313–323, 1986.
47. J. Kennington and R. Helgason. *Algorithms for Network Programming.* Wiley, New York, 1980.
48. J. Kennington and Z. Wang. An empirical analysis of the dense assignment problem: Sequential and parallel implementations. *ORSA Journal on Computing*, 3:299–306, 1986.
49. G. Kindervater, A. Volgenant, G. de Leve, and V. Gijlswijk. On dual solution of the linear assignment problem. *European Journal of Operational Research*, 19:76–81, 1985.
50. O. Kirka and A. Satir. A heuristic for obtaining an initial solution for the transportation problem. *Journal of the Operational Research Society*, 41:865–871, 1990.
51. D. Klingman, A. Napier, and J. Stutz. Netgen – a program for generating large scale uncapacitated assignment, transportation and minimum cost flow network problems. *Management Science*, 20:814–821, 1974.

52. G. Knolmayer. How many-sided are shadow prices at degenerate primal optimal? *Omega*, 4:493–494, 1976.
53. G. Knolmayer. The effect of degeneracy on cost-coefficient ranges and an algorithm to resolve interpretation problems. *Decision Sciences*, 15:14–21, 1984.
54. A.J. Lazrus. Certain expected values in the random assignment problems. *Operations Research Letters*, 14:207–214, 1993.
55. R. Lehmann. Contribution to Hungarian method. *Math. Operationforsch. u. Statist., ser. Optimization*, 15:91–97, 1984.
56. J.W.H. Liu. The role of elimination trees in spares factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.
57. V. Lotfi. A labeling algorithm to solve the assignment problems. *Computers and Operations Research*, 16:397–408, 1989.
58. C. Mack. The Bradford method for the assignment problem. *New Journal of Statistics and Operational Research*, 1:17–29, 1969.
59. S. Martello and P. Toth. Linear assignment problems. *Annuals of Discrete Mathematics*, 31:259–282, 1987.
60. D. McBride. A bump triangular dynamic factorization algorithm for the simplex method. *Mathematical Programming*, 18:49–61, 1980.
61. L.F. McGinnis. Implementation and testing of a primal-dual algorithm for the assignment problem. *Operations Research*, 31:32–48, 1983.
62. B. Moores. A note on degeneracy in integer linear programming problems. *Journal of the Operational Research Society*, 39:1175–1178, 1988.
63. J. More and S. Wright. *Optimization Software Guide.* SIAM, Philadelphia, PA, 1993.
64. D. Pannell. *Introduction to Practical Linear Programming.* Wiley, New York, 1997.
65. K. Paparrizos. An infeasible exterior point simplex algorithm for assignment problems. *Mathematical Programming*, 51:45–54, 1991.
66. K. Paparrizos. A non improving simplex algorithm for transportation problems. *Recherche Operationnelle*, 30:1–15, 1996.
67. D. Phillips and A Garcia-Diaz. *Fundamentals of Network Analysis.* Waveland Press, Inc., 1990.
68. C.S. Ramakrishnan. An improvement to Goyal's modified VAM for the unbalanced transportation problem. *Journal of Operational Research Society*, 39:609–610, 1988.
69. J. Reid. A sparsity-exploiting variant of the bartles-golub decomposition for linear program basis. *Mathematical Programming*, 24:55–69, 1982.
70. D. Robb. The more for less paradox in distribution models: an intuitive explanation. *IIE Transactions*, 22:377–378, 1990.
71. M. Rosenwein. An improved bounding procedure for the constrained assignment problem. *Computers and Operations Research*, 18:531–535, 1991.
72. A. Shafaat and S.K. Goyal. Resolution of degeneracy in transportation problems. *Journal of the Operational Research Society*, 39:411–413, 1988.
73. W. Shih. Modified stepping-stone method as a teaching aid for capacitated transportation problems. *Decision Sciences*, 18:662–676, 1987.
74. R. Silver. An algorithm for the assignment problem. *Communications of ACM*, 3:605, 1960.
75. L. Suhl and U. Suhl. *A fast LU update for linear programming.* In Applied Mathematical Programming and Modelling, Baltzer Press, 1992.
76. A. Sultan. Heuristic for finding an initial B. F. S. in a transportation problem. *Opsearch*, 25:197–199, 1988.
77. B. Taylor. *III. Introduction to Management Science.* Prentice Hall, New Jersey, 1996.
78. G. Thompson. *A recursive method for solving assignment problems.* In Studies on Graphs and Discrete Programming, North-Holland, Amsterdam, 1981.
79. P. Tolla. A stable and sparsity exploiting lu factorization of the basis matrix in linear programming. *European Journal of Operational Research*, 24:247–251, 1986.
80. J. Tomlin and J. Welch. Mathematical programming systems. *Handbooks in OR & MS*, 3:561–601, 1992.

81. H. Williams. *Model Solving in Mathematical Programming.* Wiley, Chichester, England, 1993.
82. C.E. Wilsdon. A simple, easily programmed method for locating Rook's tours in the transportation problem. *Journal of the Operational Research Society*, 41:879–880, 1990.
83. M. Wright. Speeding up the Hungarian algorithm. *Computers and Operations Research*, 17:95–96, 1990.
84. P. Zornig. *Degeneracy Graphs and Simplex Cycling.* Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, 1991.

Note. More relevant references can be found at
http://ubmail.ubalt.edu/~harsham/refop/Refop.htm

Appendix

All notations and abbreviations in the appendix are the same as used in the paper.

Proof of Theorem 1: As is well known, whenever a RHS element is zero in any simplex tableau (except the final tableau) the subsequent iterations would be degenerate, which may cause cycling if one follows ordinary simplex pivotal rules. 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 which variable goes out. This phase is also free from pivotal degeneracy (cycling) since its aim is to replace any negative (not zero) RHS entries. Hence, unlike primal simplex and network simplex-based algorithms, the Push-and-Pull algorithm is free from pivotal degeneracy.

Note that we do not mean to imply that an optimal solution would be non-degenerate, which could have some RHS = 0. This is always the case for the AP since the number of positive variables is much smaller than the size of the BVS. Rather, in the process of searching for an optimal solution, the algorithm is free from pivotal degeneracy, which may cause cycling, which in turn would prevent generating an optimal solution.

Proof of Theorem 2: The Push-and-Pull algorithm consists of the Initialization phase to generate an initial tableau that contains some basic variables, followed by the Push and Pull phases. The Push phase is a BVS augmentation process that develops a basic solution, which may or may not be feasible. If feasible, this basic solution is optimal. Otherwise, the Pull phase is activated to obtain a feasible optimal solution. Both phases use the GJP, but differ in the method used to select the pivot element (PE). The Push phase uses modified simplex column selection criteria to enter one variable at a time into an open row, rather than replacing a variable, while moving toward a vertex that is "close" to the optimal vertex. This strategy pushes toward an optimal solution, which may result in pushing too far into non-feasibility. The Pull phase, if needed, pulls back to a feasible solution that is optimal. The concise statement of the proposed scheme is as follows:

Row-column reduction : Reduce the cost matrix by a base amount to get a good start in the initialization phase.

Initialization: Start with the initial tableau with as many basic variables as possible without GJP iterations.

Push phase: Fill up the BVS completely while pushing toward an optimal vertex.

Pull phase: If the feasibility condition is not satisfied, pull back to an optimal vertex (a dual simplex approach).

The theoretical basis for the Push-and-Pull algorithm for the nominal (classical) TP and AP rests largely upon the total unimodularity of the coefficient matrix (see, e.g., [11], [21]). We use the property that a unimodal matrix remains unimodal under the GJP. Under LP formulation, optimality is attained when all C_{ij}s in the cost row are non-negative and the algorithm terminates successfully (without any simplex-type degeneracy that may cause cycling). The current algorithm starts with all non-negative C_{ij}s. We show that the C_{ij}s remain non-negative under the push and pull operations to obtain a basic and feasible solution, .

Clearly, in the Row-column reduction phase, by row-column (or column-row) reduction of the cost matrix, non-negativity of the C_{ij}s is preserved. The number of basic variables readily available is equal to the number of C_{ij}s equal to zero in the row (or column) to be eliminated. In the nominal TP and AP, each variable has at most three non-zero elements, corresponding to its origin row, destination column, and its cost. Removing the redundant constraint eliminates one non-zero element, and a zero cost removes a second, turning the column in to a unit vector identifying basic variables for the Initialization phase.

Since we are adding (not replacing) variables to the BVS in the Initialization and Push phases, deletion of basic columns is permissible. This reduces the complexity significantly and results in a smaller tableau.

The selection criteria for entering variables in the Push phase leaves several possibilities. Selection of a variable with the smallest C_{ij} ensures that the C_{ij}s remain non-negative after pivoting. Negative C/R relaxes the feasibility, but does not affect non-negativity of the C_{ij}s. If the PE is positive, we are subtracting a smaller C_{ij} from a larger C_{ij}. Finally, a negative PE adds to the C_{ij}. If pivoting is not possible (no finite C/R), we select the next smallest C_{ij}. However, the value of the smallest C_{ij} is unchanged since the pivot row element in its column is zero. By this pivoting selection rule in the Push phase, the proposed algorithm is free from pivotal degeneracy that may cause cycling.

In the Pull phase, if an RHS < 0, there exists at least one element of -1 in that row. If this were not the case, we would have an inconsistent constraint, which is impossible for the reduced TP and AP. In this phase, the reduced pivoting rule produces the same results as the usual pivoting with a smaller tableau. The proof follows from the well-known reduced pivoting rule in GJP. Notice that in the Pull phase we are considering negative, or zero, RHS elements only. Thus, the Pull phase is also free from pivotal degeneracy.

Knowing that the Push-and-Pull algorithm is free from pivotal degeneracy, we must show that the proposed algorithm converges successfully. Since the path

through the Push and Pull phases does not contain any loops, it suffices to show that each phase terminates successfully.

The Initialization phase uses the structure of the problem to fill-up the BVS as much as possible without requiring GJP iterations. The Push phase completes the BVS. The number of iterations is finite since the size of the BVS is finite. At the end of the Push phase we have a basic solution that may not be feasible. The Pull phase terminates successfully by the well-known theory of dual simplex for these problems.

Mathematical Problems in Engineering

Special Issue on
Modeling Experimental Nonlinear Dynamics and Chaotic Scenarios

Call for Papers

Thinking about nonlinearity in engineering areas, up to the 70s, was focused on intentionally built nonlinear parts in order to improve the operational characteristics of a device or system. Keying, saturation, hysteretic phenomena, and dead zones were added to existing devices increasing their behavior diversity and precision. In this context, an intrinsic nonlinearity was treated just as a linear approximation, around equilibrium points.

Inspired on the rediscovering of the richness of nonlinear and chaotic phenomena, engineers started using analytical tools from "Qualitative Theory of Differential Equations," allowing more precise analysis and synthesis, in order to produce new vital products and services. Bifurcation theory, dynamical systems and chaos started to be part of the mandatory set of tools for design engineers.

This proposed special edition of the *Mathematical Problems in Engineering* aims to provide a picture of the importance of the bifurcation theory, relating it with nonlinear and chaotic dynamics for natural and engineered systems. Ideas of how this dynamics can be captured through precisely tailored real and numerical experiments and understanding by the combination of specific tools that associate dynamical system theory and geometric tools in a very clever, sophisticated, and at the same time simple and unique analytical environment are the subject of this issue, allowing new methods to design high-precision devices and equipment.

Authors should follow the Mathematical Problems in Engineering manuscript format described at http://www.hindawi.com/journals/mpe/. Prospective authors should submit an electronic copy of their complete manuscript through the journal Manuscript Tracking System at http://mts.hindawi.com/ according to the following timetable:

Manuscript Due	December 1, 2008
First Round of Reviews	March 1, 2009
Publication Date	June 1, 2009

Guest Editors

José Roberto Castilho Piqueira, Telecommunication and Control Engineering Department, Polytechnic School, The University of São Paulo, 05508-970 São Paulo, Brazil; piqueira@lac.usp.br

Elbert E. Neher Macau, Laboratório Associado de Matemática Aplicada e Computação (LAC), Instituto Nacional de Pesquisas Espaciais (INPE), São Josè dos Campos, 12227-010 São Paulo, Brazil ; elbert@lac.inpe.br

Celso Grebogi, Center for Applied Dynamics Research, King's College, University of Aberdeen, Aberdeen AB24 3UE, UK; grebogi@abdn.ac.uk