

MATRIX BALANCING - A COMPARATIVE STUDY

by

Barbara A. Carothers

Submitted in Partial Fulfillment of the Requirements

for the degree of

Master of Science

In the

Mathematics

Program

YOUNGSTOWN STATE UNIVERSITY

July, 2000

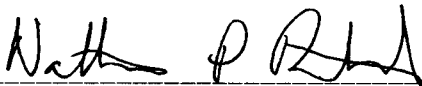
MATRIX BALANCING - A COMPARATIVE STUDY

Barbara A. Carothers

I hereby release this thesis to the public. I understand this thesis will be housed at the Circulation Desk of the University Library and will be available for public access. I also authorize the University and other individuals to make copies of this thesis as needed for scholarly research.

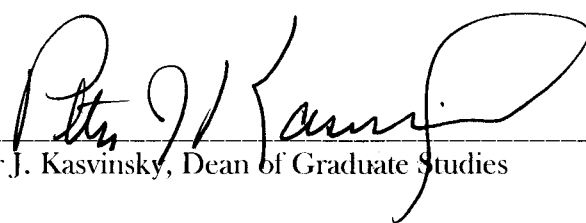
Signature:  _____ 7/25/00
Barbara Carothers, Student Date

Approvals:

 _____ 7/25/00
Dr. Nathan P. Ritchey, Thesis Advisor Date

 _____ 7/25/00
Dr. John J. Buoni, Committee Member Date

 _____ 7/25/00
Dr. Anita C. Burris, Committee Member Date

 _____ 7/31/00
Peter J. Kasvinsky, Dean of Graduate Studies Date

MATRIX BALANCING - A COMPARATIVE STUDY

Barbara A. Carothers

Abstract:

The problem of “balancing” a matrix is not new, but there have been many changes in the methods used to find a balanced matrix. This paper will introduce several different methods and present the various algorithms using a uniform notation. The algorithms will be derived, detailed, and finally applied to the same sample problem in order to compare the results. Although the first three algorithms are undoubtedly faster, the Modified Transportation Model offers more flexibility. It returns a final matrix that is often in a more useable form, in light of the various applications.

The software package “EXCEL” by Microsoft was chosen for testing these methods for a variety of reasons. Although code is readily available for similar algorithms in various programming languages, the applications of matrix balancing problems are seldom taken into consideration; these applications are in the fields of business and industry, where this particular software is already being used and is easy to access.

*Special thanks to Dr. Nathan Ritchey, whose patience and guidance
led me to believe in my own abilities;
to my children, Kyle and Tara, who didn't always understand,
but always tolerated;
and finally,
this work is lovingly dedicated to my husband, Jack, without
whose constant support none of my work is possible.
To you, who understands me better than anyone,
thank you for being around when I needed you.
Love always,
Barb
July 25, 2000*

Contents

Introduction	<i>Matrix Balancing</i>	1
	Applications	1
	Matrix Balancing - A Definition	2
	Methods of Attaining Solutions	3
Section I	<i>The Method of Least Squares</i>	3
	The Algorithm	3
	<i>Example 1: Application to the Sample Problem</i>	4
	<i>Example 2: Application w/o Weight Matrix</i>	6
	Derivation of the Least Squares Approximation	6
	<i>Theorem 1</i>	7
	<i>proof</i>	7
	<i>Example 3: Solving the Problem Directly</i>	9
Section II	<i>The RAS Algorithm</i>	10
	The Algorithm	10
	<i>Example 4: Application to the Sample Problem</i>	11
	Derivation of the RAS	12
	<i>Theorem 2 and proof</i>	12
	<i>Theorem 3</i>	12
	<i>proof</i>	13
	<i>Example 5: Comparing Column-First & Row-First Adjustments</i>	14
Section III	<i>The Modified RAS Algorithm</i>	15
	The Algorithm	15
	<i>Example 6: Application to the Sample Problem</i>	16
Section IV	<i>The Modified Transportation Model</i>	19
	The Algorithm	21
	<i>Example 7: Application to the Sample Problem</i>	22
	<i>Example 8: Sample Problem w/o Weight Matrix</i>	26
	<i>Example 9: Sample Problem with Varying Weights</i>	30
Conclusion		32
	A Note About EXCEL	32
	Observations	32
	References	34
Appendix A	<i>Visual Basic Code</i>	35
	The Least Squares Approximation	36
	The RAS Algorithm	38
	The Modified RAS Algorithm	40
	The Modified Transportation Model	42
Appendix B	<i>Test Matrices and Results</i>	52
	Matrix A - Sample Matrix	53
	Matrix B - Sample Matrix Without Weights	55
	Matrix C - Sample Matrix With Varying Weights	57
	Table I - Run Times for Random 50x50 Matrices	58

Matrix Balancing: a Comparative Study

Applications:

Applications of matrix balancing are found in various fields of study, from accounting to sociology; the most common application is known as input-output analysis.

Input-output analysis is used in a quantitative analysis of interrelations between various sectors of an economy, or, on a smaller scale, interrelations between industries, or within a single industry itself. The matrix may represent all transactions that involve sales of products or services within an economy. It is a way of indicating simultaneously the sectors responsible for production and the sectors receiving the goods or services. The matrix itself is used to analyze the effects of actual or predicted changes an action may have upon the remaining sectors.

Because the entries in an input-output matrix must be found by statistical methods that are less than precise, adjustments must necessarily be made. In other words, though one can accurately account for totals in a given area, specifics must be estimated for practical purposes. Thus the row and column sums are usually known, while the actual matrix entries are less accurate. Therefore the need to balance the matrix to known row and column sums becomes an obvious concern

A Social Accounting Matrix, or SAM, is a square matrix whose entries represent the flow-of-funds between the national income accounts of a country's economy at a fixed point in time. The indices of the matrix represent accounts, or agents, in the economy. A cell entry (a_{ij}) is positive if agent j receives funds from agent i . The agents of an economy include institutions, factors of production, households, and the rest-of-the-world (to account for transactions with the economies of other countries). SAMs are used as the core data base for complex economy-wide general equilibrium models. Inconsistent data are an inherent problem when statistical methods are used to estimate underlying economic models. Therefore, balancing the matrix becomes a necessity.

Networks are used in urban transportation analysis to model traffic flow over physical transportation systems involving, for example, urban highways, subways, etc. The matrix representing the number of travelers between each pair of vertices is called the origin-destination matrix. Matrix balancing problems arise when origin-destination matrices are estimated from observed traffic flows on selected sets of arcs.

Applications in demography include the estimation of interregional migration flows. The estimations are based on partial and often outdated information. Such problems arise when figures are available for net in- and out-migration from every region and an estimate is needed of the interregional migration patterns. In the United States, for example, flow matrices with detailed migration characteristics become available once every ten years from the general census. In the interim, net migration estimates for every region are available as by-products from annual population estimates.

Contingency tables are used to classify items by several criteria, each of which is partitioned into a finite number of categories. For many applications, it is important to estimate the underlying cell probabilities. Determining the cell probabilities directly by sampling the population is generally not possible because such a procedure is often quite expensive. Therefore the probabilities must be found with partial observations. This was the impetus for the first matrix balancing algorithm, the Least Squares Algorithm, since Deming and Stephan were working with data from the 1940 census in the United States.

What began with the data from the 1940's remained a necessity through the rest of the twentieth century. Unless a method of gathering data can be made perfect, it will remain necessary to balance matrices from this data. What follows is a collection of algorithms designed to do so.

Matrix Balancing – A Definition:

Balancing a matrix, simply stated, means adjusting the individual cell entries in order to achieve specific row and column sums. A weight matrix is included in two of the algorithms, which allows for individual interpretation of the accuracy of the cell entries. Without a weight matrix, each cell has the same level of “importance” as another; weight matrices will prevent large changes from occurring in an entry that is considered to be accurate. When a statistically calculated entry in a matrix is known to be precise, the weight function may enable the balancing procedure to take this fact into consideration. An exact entry may be given a large weight, where a less precise entry may be given a smaller weight, depending on the individual’s perception of the entry’s accuracy. Once again, this is subjective, lending the process to deviations due to observers’ opinions.

Realistically speaking, changing entries arbitrarily will result in a total loss of the original intent of the matrix, and therefore some sort of optimization algorithm is desirable. Again, there are many methods of achieving this desired optimization, but normally it is found by minimizing some sort of penalty function that tests the difference between the balanced matrix and the original matrix.

The Problem:

Given an $m \times n$ matrix A , and desired row and column totals, u and v (given in vector form), respectively, calculate an adjusted solution matrix that is “close to” A and has the desired row and column sums. A weight matrix, W , may also be assigned, if desired. This weight matrix represents the relative importance of each element of the original matrix.

Symbolically speaking:

$$\sum_{i=1}^m a_{ij} = v_j, \quad \text{for } j = 1 \dots n,$$

and

$$\sum_{j=1}^n a_{ij} = u_i, \quad \text{for } i = 1 \dots m;$$

where

u represents the desired row sums, and

v represents the desired column sums.

Sample problem:

Let the original A matrix be represented by the following:

$A =$

783	7426	4709	2145
517	928	622	703
207	373	337	425

and $v = \{1501, 8849, 5687, 3138\}$ $u = \{15028, 2844, 1303\}$

The problem is now to determine a matrix with the row sums equal to the u vector, and the column sums equal to the v vector. Balancing a matrix is accomplished by minimizing a penalty function. The penalty function must consider the difference between the original and final matrices. A weight matrix may be assigned, if desired, to stress the relative accuracy of an individual cell entry.

Methods of Attaining Solutions:

The algorithms that will be introduced in this paper are presented with the proofs of convergence to each of their minimizing functions. The Least Squares, RAS, Modified RAS Algorithms and the Modified Transportation Model are coded in VBasic – Microsoft Excel 2000 in appendix A; data and results are included in appendix B.

Over the years, many different algorithms have been developed and proven to converge to a satisfactory solution matrix. Among those is a method developed in the 1940's by an economist whose more recent notable accomplishments included the introduction of TQM, or "total quality management" to the Japanese economy. Since then, many American corporations have chosen to introduce this concept, and though they are lagging behind the Japanese in this respect, it should be noted that the late W. E. Deming is still much respected for his contributions to the field of economics. His algorithm will be the first we explore and it is known as the Least Squares Approximation.

The next type of algorithm that was introduced used diagonal matrices and was applied to an original matrix that needed to be square. The method was published in the 1960's by A. W. Marshall and I. Olkein. Because of the restrictions on the original matrix, we will not explore this method. It is known as the "DAD" algorithm, because a diagonal matrix 'D' is both pre- and post-multiplied with the original A matrix.

Following this algorithm was a similar method that did not require a square original matrix. The RAS is actually an old method that has been independently discovered and analyzed by researchers working in different areas and different countries. Kruithof (1937) proposed the algorithm, which he called the *method of twin factors*, as a procedure for predicting traffic flows between exchanges in a telephone service network. Deming and Stephan used RAS, which they called the *method of iterative proportions*, to find an approximate solution to the least-squares problem for estimating the cell frequencies of a contingency table. Other results indicate its development in the 1970's by J. Grad and improvement upon later in that same decade by A. Bachem and B. Korte. We will explore this method, as well as the proof of its convergence. (The "improved" method is called the "MRAS", or modified RAS algorithm.)

Finally, a new algorithm will be introduced which is a modification of a standard transportation model from linear programming. It was developed in 2000 by N. Ritchey and will be coded and compared to the preceding methods for efficiency and accuracy. We will refer to this method as the MTM, or modified transportation model. This model creates a change to the original matrix in only $m+n-1$ cells, and returns values for those cells that are integer solutions, two important considerations. It may be more cost-effective to minimize the number of alterations to the original matrix. One must also consider the feasibility of non-integer solutions; it would seem as though most applications would lend themselves to whole number answers.

Section I: The Method of Least Squares

The following is the general form of the least squares algorithm. If a weight matrix W is not given for the problem at hand, Deming and Stephan suggest that the matrix entries for a weight matrix be estimated to be proportional to the reciprocal of the a_{ij} 's.

The Algorithm:

Given $A^{m \times n}$, $W^{m \times n}$, $u^{m \times 1}$, $v^{1 \times n}$; initialize the iterative variables as follows:

$$\text{Let } A_{(0)} = A$$

$$\hat{u}_{(0)} = u$$

$$\hat{v}_{(0)} = v$$

$$y_{i(0)} = 0, i = 1 \dots m$$

$$z_{j(0)} = 0, j = 1 \dots n$$

Begin by calculating \hat{u} and \hat{v} , the differences between desired and current row/column sums:

For i = 1 to m

$$\hat{u}(i) = u(i) - \sum_{j=1}^n a(i,j)$$

Next i

For j = 1 to n

$$\hat{v}(j) = v(j) - \sum_{i=1}^m a(i,j)$$

Next j

Then calculate the values for the iterative variables, y and z:

For i = 1 to m

$$y(i) = y(i) + \hat{u}(i) / \sum_{j=1}^n w(i,j)$$

Next i

For j = 1 to n

$$z(j) = z(j) + \hat{v}(j) / \sum_{i=1}^m a(i,j)$$

Next j

Finally, update the A matrix according to the following algorithm:

For i = 1 to m

For j = 1 to n

$$a(i,j) = a(i,j) + w(i,j) \{ y(i) + z(j) \}$$

Next j

Next i

When the difference of two successive iterations is within a pre-specified tolerance level, the matrix will be considered balanced. For our purposes, we will specify that tolerance level as 0.1. It should be noted that the solution matrix does not contain integer values.

Example 1: Application to the Sample Problem:

A=

783	7426	4709	2145
517	928	622	703
207	373	337	425

with:

$$v = \{1501, 8849, 5687, 3138\}$$

and

$$u = \{15028, 2844, 1303\}$$

After the initialization of the iterative variables, a weight matrix must be determined. This is not considered part of the mathematical process itself, since these weights are highly subjective. If a weight matrix is not given, none should be used. In this example, the following weight matrix accompanied the original problem:

W=

75	455	358	176
52	95	56	70
19	38	31	39

$$y_i^{(0)} = 0, i = 1 \dots m$$

$$z_j^{(0)} = 0, j = 1 \dots n$$

After one iteration, the iterative variables are:

$$\hat{u}(1) = \{-35, 74, -39\}$$

$$\hat{v}(1) = \{-6, 122, 19, -135\}$$

$$y(1) = \{-0.0410958904, .2074829932, .0426966292, -.4736842105\}$$

$$z(1) = \{-0.0328947368, .2710622711, -.3070866142\}$$

The matrix resulting from the first iteration is:

A(1):

777.4467556	7505.413709	4712.490236	2055.832842	(15051.18354)
528.9582518	973.4618001	639.5704984	688.8164642	(2830.807015)
200.3845324	369.2150624	328.8039105	394.5499378	(1292.953443)
(1506.7894)	(8848.090572)	(5680.864645)	(3139.199244)	

(The corresponding row and column totals are noted in parenthesis.)

The next several updates of the A matrix are:

A(2):

772.8424461	7496.227367	4709.644459	2051.266648	(15029.989092)
529.4091734	978.199697	643.0488414	691.9047298	(2842.562442)
201.1341273	372.2798914	331.6836264	397.4709939	(1302.568639)
(1503.385747)	(8846.706955)	(5684.376927)	(3140.642372)	

A(3):

771.4772586	7497.154643	4711.088194	2049.307197	(15029.02729)
528.8332758	979.0704211	643.6738196	691.6243307	(2843.201847)
200.8881877	372.55715	331.9716502	397.2418714	(1302.658859)
(1501.198722)	(8848.782214)	(5686.733664)	(3138.173399)	

A(4):

771.302763	7496.883866	4710.956812	2049.030187	(15028.17363)
528.9041499	979.3643944	643.8598841	691.77274265	(2843.900855)
200.9133635	372.6732983	332.0734746	397.3229029	(1302.983039)
(1501.120276)	(8848.921559)	(5686.890171)	(3138.125516)	

A(5):

771.2287385	7496.870315	4711.112657	2048.923955	(15028.13567)
528.8801966	979.4115687	643.913738	691.7670198	(2843.972523)
200.9002486	372.6834426	332.0961684	397.3109356	(1302.990795)
(1501.009184)	(8848.965326)	(5687.122563)	(3138.00191)	

Example 2: Sample Problem WITHOUT the Weight Matrix:

A=

783	7426	4709	2145
517	928	622	703
207	373	337	425

with:

$$v = \{1501, 8849, 5687, 3138\}$$

and

$$u = \{15028, 2844, 1303\}$$

Let $w_{ij}=1$ for $i=1..m, j=1..n$

$$\text{Then } y(1) = \{-2, 40.667, 6.333, -45\} \quad \text{and} \quad z(1) = \{-8.75, 18.5, -9.75\}$$

The first update of the matrix becomes balanced:

A(1):

772.25	7457.92	4706.58	2091.25	(15028.5)
533.5	987.17	646.83	676.5	(2844.0)
195.25	403.92	333.58	370.25	(1303.0)
(1501.5)	(8849.01)	(5686.99)	(3138.0)	

and $A(1) \approx \text{Solution}$

Derivation of the Least Squares Approximation:

The least squares approximation is based on minimizing the following sum:

$$S = \left(\sum (x_{ij} - a_{(0)ij})^2 / a_{(0)ij} \right)$$

For the solution we seek those values of x_{ij} 's that minimize the sum as found in the equation above, wherein the x_{ij} 's are subject to the following conditions:

$$\sum_{i=1}^m x_{ij} = u_i \quad ,$$

and

$$\sum_{j=1}^n x_{ij} = v_j,$$

Note that the u_i 's and v_j 's are NOT independent, because, by their very design, it is necessary that:

$$\sum_{i=1}^m u_i = \sum_{j=1}^n v_j \quad .$$

Theorem 1: (Deming & Stephan, 1940)

The set of all x_{ij} 's that minimize $S = (\sum (x_{ij} - a^{(0)}_{ij})^2 / a^{(0)}_{ij})$ is determined by

$$\sum_i \sum_j \{2(x_{ij} - a_{ij})/a_{ij} - \lambda_i - \mu_j\} = 0,$$

$$\sum_{i=1}^m x_{ij} = u_i, \text{ and}$$

$$\sum_{j=1}^n x_{ij} = v_j.$$

Proof:

Given: $S = (\sum (x_{ij} - a^{(0)}_{ij})^2 / a^{(0)}_{ij})$

Apply the method of LaGrange multipliers:

Minimize:

$$S = \begin{aligned} &(x_{11} - a_{11})^2/a_{11} + (x_{12} - a_{12})^2/a_{12} + \dots + (x_{1n} - a_{1n})^2/a_{1n} + \\ &(x_{21} - a_{21})^2/a_{21} + (x_{22} - a_{22})^2/a_{22} + \dots + (x_{2n} - a_{2n})^2/a_{2n} + \\ &\vdots \\ &\vdots \\ &(x_{m1} - a_{m1})^2/a_{m1} + (x_{m2} - a_{m2})^2/a_{m2} + \dots + (x_{mn} - a_{mn})^2/a_{mn} \end{aligned}$$

Subject to the following conditions:

$$x_{11} + x_{12} + \dots + x_{1n} = u_1$$

$$x_{21} + x_{22} + \dots + x_{2n} = u_2$$

⋮

$$x_{m1} + x_{m2} + \dots + x_{mn} = u_m$$

and

$$x_{11} + x_{21} + \dots + x_{m1} = v_1$$

$$x_{12} + x_{22} + \dots + x_{m2} = v_2$$

⋮

$$x_{1n} + x_{2n} + \dots + x_{mn} = v_n$$

Applying LaGrange multipliers to the above yields:

$$\begin{aligned} 2(x_{11} - a_{11})/a_{11} &= \lambda_1 \\ 2(x_{21} - a_{21})/a_{21} &= \lambda_2 \\ &\vdots \\ &\vdots \\ 2(x_{m1} - a_{m1})/a_{m1} &= \lambda_m \end{aligned}$$

and

$$\begin{aligned} 2(x_{11} - a_{11})/a_{11} &= \mu_1 \\ 2(x_{12} - a_{12})/a_{12} &= \mu_2 \\ &\vdots \\ &\vdots \\ 2(x_{1n} - a_{1n})/a_{1n} &= \mu_n \end{aligned}$$

It should be noted that minimizing S is equivalent to minimizing $\frac{1}{2} S$, so the application of LaGrange multipliers can be simplified to be the following system of equations:

$$\begin{aligned} (x_{11} - a_{11})/a_{11} &= \lambda_1 \\ (x_{21} - a_{21})/a_{21} &= \lambda_2 \\ &\vdots \\ &\vdots \\ (x_{m1} - a_{m1})/a_{m1} &= \lambda_m \\ (x_{11} - a_{11})/a_{11} &= \mu_1 \\ (x_{12} - a_{12})/a_{12} &= \mu_2 \\ &\vdots \\ &\vdots \\ (x_{1n} - a_{1n})/a_{1n} &= \mu_n \end{aligned}$$

which become:

$$\begin{aligned} \sum_i \sum_j \{2(x_{ij} - a_{ij})/a_{ij} - \lambda_i - \mu_j\} &= 0. \\ \sum_{i=1}^m x_{ij} &= u_i, \text{ and} \\ \sum_{j=1}^n x_{ij} &= v_j. \end{aligned}$$

Q.E.D.

From this equation we get a system of $m+n-1$ equations, which, if the last μ is allowed to be zero, can easily be solved using any preferred method for the individual LaGrange multipliers AND the final matrix entries. Since solving systems is much more efficient using modern technology, it is not much more difficult to solve directly for the x_{ij} 's than it is to solve using the iterative method.

Example 3: Solving the Sample Problem Directly:

The direct method applied to the example problem gives the following problem

Given:

A=

783	7426	4709	2145
517	928	622	703
207	373	337	425

with:

$$v = \{1501, 8849, 5687, 3138\}$$

and

$$u = \{15028, 2844, 1303\}$$

Using the direct method, we arrive at the following system of equations:

$$\begin{aligned} 2x_{11} - 783\lambda_1 - 783\mu_1 &= 2(783) \\ 2x_{12} - 7426\lambda_1 - 7426\mu_2 &= 2(7426) \\ 2x_{13} - 4709\lambda_1 - 4709\mu_3 &= 2(4709) \\ 2x_{14} - 2145\lambda_1 - 2145\mu_4 &= 2(2145) \\ 2x_{21} - 517\lambda_2 - 517\mu_1 &= 2(517) \\ 2x_{22} - 928\lambda_2 - 928\mu_2 &= 2(928) \\ 2x_{23} - 622\lambda_2 - 622\mu_3 &= 2(622) \\ 2x_{24} - 703\lambda_2 - 703\mu_4 &= 2(703) \\ 2x_{31} - 207\lambda_3 - 207\mu_1 &= 2(207) \\ 2x_{32} - 373\lambda_3 - 373\mu_2 &= 2(373) \\ 2x_{33} - 337\lambda_3 - 337\mu_3 &= 2(337) \\ 2x_{34} - 425\lambda_3 - 425\mu_4 &= 2(425) \\ x_{11} + x_{12} + x_{13} + x_{14} &= 15028 \\ x_{21} + x_{22} + x_{23} + x_{24} &= 2844 \\ x_{31} + x_{32} + x_{33} + x_{34} &= 1303 \\ x_{11} + x_{21} + x_{31} &= 1501 \\ x_{12} + x_{22} + x_{32} &= 8849 \\ x_{13} + x_{23} + x_{33} &= 5687 \\ x_{14} + x_{24} + x_{34} &= 3138; \end{aligned}$$

but since $\mu_4 = 0$, the resulting system is:

2	0	0	0	0	0	0	0	0	0	0	0	0	-783	0	0	-783	0	0	1566
0	2	0	0	0	0	0	0	0	0	0	0	0	-7426	0	0	0	-7426	0	14832
0	0	2	0	0	0	0	0	0	0	0	0	0	-4709	0	0	0	0	-4709	9418
0	0	0	2	0	0	0	0	0	0	0	0	0	-2145	0	0	0	0	0	4290
0	0	0	0	2	0	0	0	0	0	0	0	0	0	-517	0	-517	0	0	1034
0	0	0	0	0	2	0	0	0	0	0	0	0	0	-928	0	0	-928	0	1856
0	0	0	0	0	0	2	0	0	0	0	0	0	0	-622	0	0	0	-622	1244
0	0	0	0	0	0	0	2	0	0	0	0	0	0	-703	0	0	0	0	1406
0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	-207	-207	0	0	414
0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	-373	0	-373	0	746
0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	-337	0	0	-337	674
0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	-425	0	0	0	850
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15028
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	2844
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1303
1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1501
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	8849
0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	5687

with the right hand side:

rhs = {1566, 14832, 9418, 4290, 1034, 1856, 1244, 1406, 414, 746, 674, 850, 15028, 2844, 1303, 1501, 8849, 5687}

The following solution was achieved with the use of a hand-held calculator:

$$\lambda_1 = -.0946861647$$

$$\lambda_2 = -.0196645228$$

$$\lambda_3 = -.1248800403$$

$$\mu_1 = .0651333733$$

$$\mu_2 = .1182500736$$

$$\mu_3 = .0949529034$$

$$\mu_4 = 0$$

$$x_{11} = 771.4300822$$

$$x_{12} = 7503.492793$$

$$x_{13} = 4709.628036$$

$$x_{14} = 2043.449088$$

$$x_{21} = 528.7536979$$

$$x_{22} = 973.7436955$$

$$x_{23} = 645.4146864$$

$$x_{24} = 696.0879202$$

$$x_{31} = 200.81622$$

$$x_{32} = 371.7635112$$

$$x_{33} = 331.9572774$$

$$x_{34} = 398.4629914$$

Note that this solution is the same as that found using the iterative method earlier.

Section II: The RAS Algorithm

The RAS algorithm is a modification of a method developed by a Russian mathematician. The RAS technique is a method of adjusting the original matrix by finding a set of multipliers r_i and s_j such that the resulting matrix satisfies the predetermined row and column sums.

This is accomplished by using an iterative technique that alternates between:

- i.) adjusting each entry by multiplying by a ratio of the desired column sums with the previous matrix's column sums
- ii.) adjusting each entry by multiplying by a ratio of the desired row sums with the previous matrix's row sums

The Algorithm:

Adjust by rows first:

For i = 1 to m

For j = 1 to n

$$a(i,j) = a(i,j) \left\{ \frac{u(i)}{\sum_{j=1}^n a(i,j)} \right\}$$

Next j

Next i

Then adjust by columns:

For i = 1 to m

For j = 1 to n

$$a(i,j) = a(i,j) \left\{ v(j) / \sum_{i=1}^m a(i,j) \right\}$$

Next j

Next i

The iterative method will be completed when both the row and column sums converge to the desired totals.

Example 4: Application to the sample problem:

A:

783	7426	4709	2145	(15063)
517	928	622	703	(2270)
207	373	337	425	(1342)
(1507)	(8727)	(5668)	(3273)	

$$v = \{1501, 8849, 5687, 3138\} \quad \text{and} \quad u = \{15028, 2844, 1303\}$$

A(1):

779.8825481	7529.812536	4724.785286	2056.526123	(15091.00649)
514.9416058	940.9730721	624.0850388	674.0036664	(2754.003383)
206.1758461	378.2143921	338.1296754	407.4702108	(1329.990124)
(1501)	(8849)	(5687)	(3138)	

A(2):

776.626459	7498.374801	4705.058826	2047.939917	(15028)
531.7691096	971.7226324	644.479183	696.029075	(2844)
201.9918213	370.539107	331.2678486	399.2012234	(1303)
(1510.38739)	(8840.63654)	(5680.805858)	(3143.170215)	

A(3):

771.7995547	7505.468448	4710.189049	2044.571251	(15032.0283)
528.4640476	972.641906	645.1818994	694.8841736	(2841.172027)
200.7363977	370.8896461	331.6290509	398.5445755	(1301.79967)
(1501)	(8849)	(5687)	(3138)	

A(4):

771.592727	7503.457124	4708.926801	2044.023344	(15028)
528.9900566	973.6100294	645.824084	695.5758296	(2844)
200.9214876	371.2316265	331.9348309	398.9120552	(1303)
(1501.504271)	(8848.29878)	(5686.685716)	(3138.693695)	

A(5):

771.3335923	7504.051766	4709.187048	2043.571586	(15028.14399)
528.8123985	973.6871871	645.8597765	695.4220977	(2843.78146)
200.8540094	371.2610463	331.9531758	398.8238901	(1302.888206)
(1501)	(8849)	(5687)	(3138)	

A(6):

771.3262019	7503.979867	4709.141928	2043.552006	(15028)
528.8530369	973.7620134	645.9094098	695.4755398	(2844)
200.8712436	371.2929023	331.981659	398.8581111	(1303)
(1501.050482)	(8849.034783)	(5687.032997)	(3137.885657)	

Derivation of the RAS:

Theorem 2: (Bachem & Korte, 1979)

Given the matrix $A^{m \times n}$, and the vectors $u^{m \times 1}$ and $v^{1 \times n}$, there exist vectors $r^{m \times 1}$ and $s^{1 \times n}$ such that:

$$\sum_j r_i a_{ij} s_j = u_i \quad \text{and} \quad \sum_i r_i a_{ij} s_j = v_j$$

These vectors are unique.*

Proof:

There are $m+n-1$ conditions (each independent) since necessarily the total sum is

$$\sum_i u_i = \sum_j v_j$$

and there are $m + n$ elements of the two vectors, r and s , to be determined. The conditions are one too few to determine them uniquely, but since each element of r appears with each element of s and never alone, the conditions determine the r and s uniquely, except for a multiplier $\lambda r_i, 1/\lambda s_j$. Thus in any case the conditions suffice to determine the solution matrix uniquely.

Q.E.D.

*Actually, the vectors themselves are not unique, since there is an infinite number of solutions. They are, however, unique in pairs, since each r has a unique corresponding s .

Theorem 3: (Bachem & Korte, 1979)

The following:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} \left(\sum_{j=1}^n x_{ij} / \sum_{j=1}^n a_{ij}^{(k)} \right)$$

and

$$a_{ij}^{(k+2)} = a_{ij}^{(k+1)} \left(\sum_{i=1}^m x_{ij} / \sum_{i=1}^m a_{ij}^{(k+1)} \right)$$

converge to a solution such that:

$$\sum_j x_{ij} = u_i$$

and

$$\sum_i x_{ij} = v_j.$$

Proof:

Consider the first full iteration:

$$\begin{aligned} a_{ij}^{(1)} &= a_{ij} \left(\sum_j x_{ij} / \sum_j a_{ij} \right) \\ &= a_{ij} \left(\sum_j x_{ij} / \sum_j x_{ij} \right) (1 / r_i s_j) \\ &= x_{ij} / (s_j s'_i) \end{aligned}$$

$$\text{where } s'_i = \sum_j x_{ij} (1 / s_i) / \sum_j x_{ij}$$

and

$$\begin{aligned} a_{ij}^{(2)} &= x_{ij} / (s'_i s'_j) \left(\sum_i x_{ij} / \sum_i x_{ij} \right) (1 / s_j s'_i) \\ &= x_{ij} / (s'_i s''_j) \end{aligned}$$

$$\text{where } s''_j = \sum_i x_{ij} (1 / s'_i) / \sum_i x_{ij}.$$

It is easily found that in general

$$a_{ij}^{(2k)} = x_{ij} / (s'^{(2k-1)}_i s''^{(2k)}_j) \quad \text{for } k = 1, 2, \dots$$

The expression in parenthesis is a weighted mean multiplied by the weighted mean of the reciprocal set of such means, the weights being elements of the columns of the final matrix. If the elements of that matrix are strictly positive, each operation of taking weighted means will strictly reduce the range of the sets, with the result that the expression in parenthesis approaches one, and hence:

$$a_{ij}^{(2k)} \rightarrow x_{ij} \quad \text{as } k \rightarrow \infty.$$

It is however, possible, if some $x_{ij} < 0$, for the sequence to be divergent. In the version above, the calculation begins by adjusting rows. If the calculation begins by adjusting columns, the expressions are the same, with r_i 's and s_j 's being reversed. The sequence may converge much more quickly one way than the other, and it is possible for the process to converge one way, and not the other. If both processes are convergent, then they will converge to the same solution matrix.

Q.E.D.

Example 5: Comparing Column-First and Row-First Adjustments:

A:

12	13	14
16	17	18

u: {30, 30, 30} v: {40,50}

Adjusting by columns first:

A(1):

12.85714286	13	13.125
17.14285714	17	16.875

A(2):

13.19285387	13.33944114	13.46770499
16.79341835	16.6534732	16.53102119

A(3):

13.19889359	13.34259251	13.46827686
16.80110641	16.65740749	16.53172314

A(4):

13.19567287	13.33933672	13.46499041
16.80438762	16.66066064	16.53495174

A(5):

13.19564626	13.33933789	13.46501638
16.80435374	16.66066211	16.53498362

A(6):

13.19564609	13.33933771	13.4650162
16.80435392	16.66066229	16.5349838

A(7):

13.19564608	13.33933772	13.4650162
16.80435392	16.66066228	16.5349838

Adjusting by rows first:

A(1):

12.30769231	13.33333333	14.35897436
15.68627451	16.66666667	17.64705882

A(2):	13.18965517	13.33333333	13.45900094
	16.81034483	16.66666667	16.54099906

A(3):	13.19559667	13.33933955	13.46506377
	16.80429174	16.66066531	16.53504295

A(4):	13.19564575	13.33933739	13.46501587
	16.80435425	16.66066261	16.53498413

A(5):	13.19564608	13.33933772	13.4650162
	16.80435392	16.66066228	16.5349838

In the case where rows/columns are already “close” to balanced, the choice of which adjustment should be made first is obvious. Most other times, however, it makes little difference. This is especially true when the adjustments are being done by computer.

Section III: The Modified RAS (MRAS)

The modified RAS algorithm is an improvement over the original RAS algorithm. It overcomes the main disadvantage of the RAS algorithm by reducing the number of algebraic operations performed per iteration. It also returns the multipliers without extra computational cost. Computational results have shown that the above improvements reduce the number of iterations to reach a given precision and reduce considerably the overall time for solving a sample problem.

The Algorithm:

Initialize: $A_{(0)} = A, k=0$
 $r(1) = 1$

Calculate the remainder of the diagonal elements of the “R” matrix:

For $i = 2$ to m

$$r(i) = u(i) / \sum_{j=1}^n a(i,j)$$

Next i

Calculate the diagonal elements of the “S” matrix:

For $j = 1$ to n

$$s(i) = v(i) / \sum_{i=1}^m a(i,j)$$

Next j

Finally alternate between adjusting the matrix using the "R" and "S" matrices:

```

For i = 1 to m
  For j = 1 to n
    a(i,j) = r(i) { a(i,j) }
  Next j
Next i

```

```

For i = 1 to m
  For j = 1 to n
    a(i,j) = { a(i,j) } s(j)
  Next j
Next i

```

Example 6: Application to the Sample Problem:

A:

783	7426	4709	2145
517	928	622	703
207	373	337	425

v: {1501, 8849, 5687, 3138}

u^T: {15028, 2844, 1303}

r(1):

1	0	0
0	2844/2770	0
0	0	1303/1342

A(1):

783	7426	4709	2145
530.8115523	952.7913357	638.6166065	721.7805054
200.9843517	362.1602086	327.2064083	412.6490313

s(2):

1501/1514.795904	0	0	0
0	8849/8740.951544	0	0
0	0	5687/5674.823015	0
0	0	0	3138/3279.429537

A(2):

775.868879	7517.794106	4719.104531	2052.494168
525.9772211	964.5689588	639.9869444	690.6528103
199.1539	366.6369354	327.908525	394.8530211

r(3):

1	0	0
0	2844/2821.185935	0
0	0	1303/1288.552382

A(3):

775.868879	7517.794106	4719.104531	2052.494168
530.2306375	972.369132	645.1623224	696.2379077
201.3868705	370.7477736	331.5851292	399.2802262

s(4):

1501/1507.486387	0	0	0
0	8849/8860.911012	0	0
0	0	5687/5695.851983	0
0	0	0	3138/3148.012302

A(4):

772.5304835	7507.688539	4711.770521	2045.966179
527.94917	971.0620542	644.1596689	694.0235122
200.5203464	370.2494071	331.0698093	398.0103092

r(5):

1	0	0
0	2844/2837.194405	0
0	0	1303/1299.849872

A(5):

772.5304835	7507.688539	4711.770521	2045.966179
529.2155648	973.3913465	645.7048185	695.6882705
201.0062985	371.14669	331.8721421	398.9748694

s(6):

1504/1502.752347	0	0	0
0	8849/8852.226576	0	0
0	0	5687/5689.347482	0
0	0	0	3138/3140.629319

A(6):

771.6296421	7504.952037	4709.826397	2044.253306
528.598451	973.036552	645.4383942	695.1058438
200.7719068	371.0114095	331.7352083	398.6408496

r(7):

1	0	0
0	2844/2842.179241	0
0	0	1303/1302.159374

A(7):

771.6296421	7504.952037	4709.826397	2044.253306
528.9370821	973.6598994	645.8518754	695.551143
200.9015177	371.2509208	331.9493643	398.8981974

s(8):

1501/1501.468242	0	0	0
0	8849/8849.862857	0	0
0	0	5687/5687.627637	0
0	0	0	3138/3138.702646

A(8):

771.3890047	7504.220309	4709.306662	2043.795669
528.7721298	973.5649681	645.7806049	695.3954334
200.8388653	371.214724	331.9127333	398.8088981

r(9):

1	0	0
0	2844/2843.513136	0
0	0	1303/1302.775221

A(9):

771.3890047	7504.220309	4709.306662	2043.795669
528.8626658	973.7316611	645.8911749	695.5144984
200.8735178	371.2787729	331.9700011	398.877708

s(10):

1501/1501.125188	0	0	0
0	8849/8849.230743	0	0
0	0	5687/5687.167838	0
0	0	0	3138/3138.187875

A(10):

771.3246739	7504.024637	4709.167682	2043.673313
528.8185607	973.7062711	645.8721135	695.4728598
200.8567657	371.2690918	331.9602041	398.8538282

r(11):

1	0	0
0	2844/2843.869805	0
0	0	1303/1302.93989

A(11):

771.3246739	7504.024637	4709.167682	2043.673313
528.8427705	973.7508483	645.9016822	695.5046992
200.8660321	371.28622	331.9755188	398.872229

s(12):

1501/1501.033477	0	0	0
0	8849/8849.061705	0	0
0	0	5687/5687.044883	0
0	0	0	3138/3138.050241

A(12):

771.3074713	7503.972311	4709.130517	2043.640593
528.8309759	973.7440583	645.8965846	695.493564
200.8615522	371.283631	331.9728988	398.865843

r(13):

1	0	0
0	2844/2843.965183	0
0	0	1303/1302.983925

A(13):

771.3074713	7503.972311	4709.130517	2043.640593
528.8374501	973.7559792	645.9044919	695.5020785
200.8640303	371.2882116	331.9769943	398.8707638

and A(13) \approx Solution.

Section IV: The Modified Transportation Model:

The problem, as applied in this more recent model, is to minimize a penalty function resulting from a weighted absolute value. This new algorithm is similar to the algorithm that solves instances of the transportation model. The transportation model solves the following linear program:

$$\min z = \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij}$$

subject to the following constraints:

$$c_{ij} \geq 0$$

and

$$x_{ij} \geq 0.$$

where c_{ij} represents the cost. The transportation model itself is a linear program that can be solved using the simplex method. The transportation model is used because it is significantly faster than the simplex method.

We will use a similar method by making some adjustments, both to the matrix balancing problem and the transportation model. We begin by adjusting the original A matrix to become a matrix representing the difference between the given matrix (the a_{ij} 's) and the solution matrix (the x^{ij} 's):

$$y_{ij} = a_{ij} - x_{ij}$$

The adjustment to the transportation model becomes an obvious necessity, since the original transportation model required all non-negative values. We will denote the entries for the y-matrix in the following manner:

If y is positive: y_{ij}^+

and

if y is negative: y_{ij}^-

We then will adjust the weight matrix, which represents the cost in the transportation model, in a similar manner. The positive weight will be used when the y-entry is positive, and the negative weight when the y-entry is negative.

If w is positive: w_{ij}^+
 and
 if w is negative: w_{ij}^-

This presents an opportunity unique to this method of matrix balancing: the ability to assign different positive and negative weights. If we allow d_{ij} to represent the difference between the original and the final matrix entries, our Modified Transportation Model becomes the following:

$$\min \sum_{i=1}^m \sum_{j=1}^n w_{ij} |d_{ij}|$$

subject to:

$$\sum_{i=1}^m d_{ij} = \sum_{i=1}^n a_{ij} - v_j$$

$$\sum_{j=1}^n d_{ij} = \sum_{j=1}^m a_{ij} - u_i$$

or, using the notation for possible variations in positive and negative weights:

$$\min \sum_{i=1}^m \sum_{j=1}^n [w_{ij}^+ y_{ij}^+ - w_{ij}^- y_{ij}^-]$$

s.t.:

$$\sum_{j=1}^n [y_{ij}^+ - y_{ij}^-] = \hat{v}_j$$

$$\sum_{i=1}^m [y_{ij}^+ - y_{ij}^-] = \hat{u}_i$$

$$y_{ij}^+ \geq 0$$

$$y_{ij}^- < 0$$

Conditions of the Modified Transportation Model:

The dual to the above is:

$$\max \left\{ \sum_{i=1}^m a_i \hat{v}_i + \sum_{j=1}^n b_j \hat{u}_j \right\}$$

s.t.:

$$w_{ij}^- \leq a_i + b_j \leq w_{ij}^+$$

With the following complementary slackness conditions:

$$(y_{ij}^+)(a_i + b_j - w_{ij}^+) = 0$$

$$(y_{ij}^-)(-a_i - b_j + w_{ij}^-) = 0$$

$$(a_i) \left[\sum_{j=1}^n (y_{ij}^+ - y_{ij}^-) - \hat{u}_i \right] = 0$$

$$(b_j) \left[\sum_{i=1}^m (y_{ij}^+ - y_{ij}^-) - \hat{v}_j \right] = 0 \quad \{\text{and } w_{ij}^+ \geq 0, w_{ij}^- < 0\}$$

The complimentary slackness conditions ensure that if a feasible solution and the primal dual variable solutions satisfy these conditions, then the solution is optimal.

The Algorithm:

i.) Find the row and column constraints:

For i = 1 to m

$$\hat{u}(i) = u(i) - \sum_{j=1}^n a(i,j)$$

Next i

For j = 1 to n

$$\hat{v}(j) = v(j) - \sum_{i=1}^m a(i,j)$$

Next j

ii.) Determine the initial tableau in the following format:

	b_1	b_2	b_n	
a_1	w_{ij}^-	w_{ij}^-		w_{ij}^-	\hat{u}_1
	w_{ij}^+	w_{ij}^+	w_{ij}^+	
	y_{11}	y_{12}		y_{1n}	
a_2	w_{ij}^-	w_{ij}^-		w_{ij}^-	\hat{u}_2
	w_{ij}^+	w_{ij}^+	w_{ij}^+	
	y_{21}	y_{22}		y_{2n}	
\vdots					\vdots
\vdots	\vdots
\vdots					\vdots
a_m	w_{ij}^-	w_{ij}^-		w_{ij}^-	\hat{u}_m
	w_{ij}^+	w_{ij}^+	w_{ij}^+	
	y_{m1}	y_{m2}		y_{mn}	
	\hat{v}_1	\hat{v}_2		\hat{v}_n	

We will use, for simplicity, a rule called the “Northwest Corner Rule”. The largest possible absolute value is allocated to the y_{ij} entry in row 1, column 1. That means either \hat{u} or \hat{v}_1 will become the entry in the first cell.

The satisfied row (or column) is then “crossed out,” or eliminated from the remainder of the procedure. The remaining column (or row) is then satisfied, followed by a row (or column,) and so on. The procedure continues until there are $m+n-1$ “basic variables” and all row and column totals are satisfied.

- iii.) Find the values for the dual variables (a_i 's and b_j 's).
 - a.) $a_1 = 0$
 - b.) $a_i + b_j = w_{ij}^{\pm}$;
- iv.) Check the tableau for optimality. The tableau is consider optimal when: $w_{ij}^- \leq a_i + b_j \leq w_{ij}^+$
- v.) If the tableau is not optimal, find a variable to enter the basis. For simplicity, we will choose the first encountered that does not meet the criteria for iv.). Determine whether the entering variable violates the criteria by being too large (bring in a positive value) or too small (bring in a negative value.) This is crucial to the solution.
- vi.) Construct a loop, consisting of horizontal and vertical arcs, containing only basic variables AND the entering variable. This loop is unique, and direction is immaterial.
- vii.) If the entering variable is to be positive, determine the largest possible positive value that can be successively added to and subtracted from the elements of the loop without changing the sign of an individual entry. If negative, determine the smallest possible negative value. The entry with this value becomes the leaving variable.
- viii.) Update the loop by adding/subtracting the appropriate value. This becomes the new tableau.
- ix.) Repeat the process until the tableau is optimal.

Example 7: Application to the Sample Problem:

A=

783	7426	4709	2145
517	928	622	703
207	373	337	425

With the weight matrix:

W=

75	455	358	176
52	95	56	70
19	38	31	39

	1	2	3	4	
1	-75	-455	-358	-176	-35
	75	455	358	176	
	-35	---	---	---	
2	-52	-95	-56	-70	74
	52	95	56	70	
	29	45	---	---	
3	-19	-38	-31	-39	-39
	19	38	31	39	
	---	77	19	-135	
	-6	122	19	-135	

It should be noted that the resultant optimal tableau will represent the difference between the original matrix and the final matrix; i.e.,

$$a_{ij} + y_{ij} = \text{solution matrix entries}$$

Hereafter, all non-basic variables will not be noted in the sample tableaux, but rather the entries shall be left blank. It can be assumed that blank entries represent zeroes.

The marginal values a_i and b_j , which represent the values for the dual variables, are then calculated in the following manner:

- ✓ $a_1 = 0$
- ✓ $a_i + b_j = w_{ij}^{\pm}$;

	$b_1 = -75$	$b_2 = -32$	$b_3 = -39$	$b_4 = -109$	
$a_1 = 0$	-75	-455	-358	-176	-35
	75	455	358	176	
	-35				
$a_2 = 127$	-52	-95	-56	-70	74
	52	95	56	70	
	29	45			
$a_3 = 70$	-19	-38	-31	-39	-39
	19	38	31	39	
		77	19	-135	
	-6	122	19	-135	

Next, the tableau is checked for optimality. The tableau represents an optimal solution when, for each non-basic variable (represented by blank entries in the tableau) the sum of the corresponding a's and b's is within the interval of the costs/weights.

It is sometimes recommended that the “best” choice for an entering variable needs to be determined by any one of a number of methods, but most are highly subjective. For our purposes, the new entering variable will be the first encountered when working left to right, top to bottom:

	$b_1 = -75$	$b_2 = -32$	$b_3 = -39$	$b_4 = -109$	
$a_1 = 0$	$\begin{array}{ c } \hline -75 \\ \hline 75 \\ \hline \end{array}$ -35	$\begin{array}{ c } \hline -455 \\ \hline 455 \\ \hline \end{array}$ $-455 \leq (0 + -32) \leq 455$ ✓	$\begin{array}{ c } \hline -358 \\ \hline 358 \\ \hline \end{array}$ $-358 \leq (0 + -39) \leq 358$ ✓	$\begin{array}{ c } \hline -176 \\ \hline 176 \\ \hline \end{array}$ $-176 \leq (0 + -109) \leq 176$ ✓	-35
$a_2 = 127$	$\begin{array}{ c } \hline -52 \\ \hline 52 \\ \hline \end{array}$ 29	$\begin{array}{ c } \hline -95 \\ \hline 95 \\ \hline \end{array}$ 45	$\begin{array}{ c } \hline -56 \\ \hline 56 \\ \hline \end{array}$ $(127 + -51) < 56$ entering variable	$\begin{array}{ c } \hline -70 \\ \hline 70 \\ \hline \end{array}$	74
$a_3 = 70$	$\begin{array}{ c } \hline -19 \\ \hline 19 \\ \hline \end{array}$	$\begin{array}{ c } \hline -38 \\ \hline 38 \\ \hline \end{array}$ 77	$\begin{array}{ c } \hline -31 \\ \hline 31 \\ \hline \end{array}$ 19	$\begin{array}{ c } \hline -39 \\ \hline 39 \\ \hline \end{array}$ -135	-39
	-6	122	19	-135	

This tableau does not represent an optimal solution, so another must be formed. Since the test for optimality denoted the non-basic variable’s test value was outside of the interval on the positive side, it must be brought in at a positive value:

$$\begin{aligned} \text{if } a_i + b_j > w_{ij}^+, \text{ then } y_{ij} > 0 \\ \text{if } a_i + b_j < w_{ij}^-, \text{ then } y_{ij} < 0 \end{aligned}$$

The leaving variable is found using a network of horizontal and vertical lines, creating a loop with either a basic variable or the entering variable at the corner points:

	b_1	b_2	b_3	b_4	
a_1	$\begin{array}{ c } \hline -75 \\ \hline 75 \\ \hline \end{array}$ -35	$\begin{array}{ c } \hline -455 \\ \hline 455 \\ \hline \end{array}$	$\begin{array}{ c } \hline -358 \\ \hline 358 \\ \hline \end{array}$	$\begin{array}{ c } \hline -176 \\ \hline 176 \\ \hline \end{array}$	-35
a_2	$\begin{array}{ c } \hline -52 \\ \hline 52 \\ \hline \end{array}$ 29	$\begin{array}{ c } \hline -95 \\ \hline 95 \\ \hline \end{array}$ 45	$\begin{array}{ c } \hline -56 \\ \hline 56 \\ \hline \end{array}$ y_{23}	$\begin{array}{ c } \hline -70 \\ \hline 70 \\ \hline \end{array}$	74
a_3	$\begin{array}{ c } \hline -19 \\ \hline 19 \\ \hline \end{array}$	$\begin{array}{ c } \hline -38 \\ \hline 38 \\ \hline \end{array}$ 77	$\begin{array}{ c } \hline -31 \\ \hline 31 \\ \hline \end{array}$ 19	$\begin{array}{ c } \hline -39 \\ \hline 39 \\ \hline \end{array}$ -135	-39
	-6	122	19	-135	

To determine the leaving variable, the entering variable must be made as large in absolute value as possible, without causing any basic variable to change in sign. In our example, y_{23} must be positive, so its new value is 19. This leaves y_{33} with a value of zero, thus making it the leaving variable. The value 45 could not be used, since it would cause a sign change in the entry for row 3, column 3. The new tableau is:

	$b_1 = -75$	$b_2 = -32$	$b_3 = -71$	$b_4 = -109$	
$a_1 = 0$	$\begin{array}{c} -75 \\ 75 \\ -35 \end{array}$	$\begin{array}{c} -455 \\ 455 \\ -455 \leq (0 + -32) \leq 455 \\ \checkmark \end{array}$	$\begin{array}{c} -358 \\ 358 \\ -358 \leq (0 + -71) \leq 358 \\ \checkmark \end{array}$	$\begin{array}{c} -176 \\ 176 \\ -176 \leq (0 + -109) \leq 176 \\ \checkmark \end{array}$	-35
$a_2 = 127$	$\begin{array}{c} -52 \\ 52 \\ 29 \end{array}$	$\begin{array}{c} -95 \\ 95 \\ 26 \end{array}$	$\begin{array}{c} -56 \\ 56 \\ 19 \end{array}$	$\begin{array}{c} -70 \\ 70 \\ -70 < (-127 + -109) < 70 \\ \checkmark \end{array}$	74
$a_3 = 70$	$\begin{array}{c} -19 \\ 19 \\ -19 \leq (70 + -75) \leq 19 \\ \checkmark \end{array}$	$\begin{array}{c} -38 \\ 38 \\ 96 \end{array}$	$\begin{array}{c} -31 \\ 31 \\ -31 \leq (70 + -71) \leq 31 \\ \checkmark \end{array}$	$\begin{array}{c} -39 \\ 39 \\ -135 \end{array}$	-39
	-6	122	19	-135	

Since all sums of a_i 's and b_j 's fall between the costs/weights, this tableau represents an optimal solution. Therefore, the final solution matrix is represented by:

$783 + -35$	$7426 + 0$	$4709 + 0$	$2145 + 0$
$517 + 29$	$928 + 26$	$622 + 19$	$703 + 0$
$207 + 0$	$373 + 96$	$337 + 0$	$425 + -95$

or, finally:

748	7426	4709	2145
546	954	641	703
207	469	337	290

which gives desired row and column sums.

Example 8: Sample Problem WITHOUT the Weight Matrix:

A=

783	7426	4709	2145
517	928	622	703
207	373	337	425

with:

$$v = \{1501, 8849, 5687, 3138\}$$

and

$$u = \{15028, 2844, 1303\}$$

become the following tableau:

The marginal values a_i and b_j are then calculated in the following manner:







- ✓ $a_1 = 0$
- ✓ $a_i + b_j = 1^{\pm}$;
 - it should also be noted that since there is no cost involved, the cost/weight value of positive one will be used for y_{ij} 's with positive entries, while negative one will be used for negative entries.

	$b_1 = -1$	$b_2 = -1$	$b_3 = -1$	$b_4 = -3$	
$a_1 = 0$	$\begin{matrix} -1 \\ 1 \end{matrix}$ -35	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$	-35
$a_2 = 2$	$\begin{matrix} -1 \\ 1 \end{matrix}$ 29	$\begin{matrix} -1 \\ 1 \end{matrix}$ 45	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$	74
$a_3 = 2$	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$ 77	$\begin{matrix} -1 \\ 1 \end{matrix}$ 19	$\begin{matrix} -1 \\ 1 \end{matrix}$ -135	-39
	-6	122	19	-135	

Next, check the tableau for optimality. Recall that the tableau represents an optimal solution when, for each non-basic variable (represented by the blank entries in the tableau) the sum of the corresponding a 's and b 's is within the interval of the costs/weights. The first non-basic variable encountered that does not meet this requirement will then become the entering variable.

	$b_1 = -1$	$b_2 = -1$	$b_3 = -1$	$b_4 = -3$	
$a_1 = 0$	$\begin{matrix} -1 \\ 1 \end{matrix}$ -35	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0 + -1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0 + -1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ $(0 + -3) > 1$ entering variable	-35
$a_2 = 2$	$\begin{matrix} -1 \\ 1 \end{matrix}$ 29	$\begin{matrix} -1 \\ 1 \end{matrix}$ 45	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$	74
$a_3 = 2$	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$ 77	$\begin{matrix} -1 \\ 1 \end{matrix}$ 19	$\begin{matrix} -1 \\ 1 \end{matrix}$ -135	-39
	-6	122	19	-135	

This tableau does not represent an optimal solution, so another must be formed. The leaving variable is found using a network of horizontal and/or vertical lines, creating a loop with either a basic variable or the entering variable at the corner points:

	b ₁	b ₂	b ₃	b ₄	
a ₁	$\begin{matrix} -1 \\ 1 \end{matrix}$ -35 	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$ y ₁₄ 	-35
a ₂	$\begin{matrix} -1 \\ 1 \end{matrix}$ 29 	$\begin{matrix} -1 \\ 1 \end{matrix}$ 45 	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$	74
a ₃	$\begin{matrix} -1 \\ 1 \end{matrix}$	$\begin{matrix} -1 \\ 1 \end{matrix}$ 77 	$\begin{matrix} -1 \\ 1 \end{matrix}$ 19	$\begin{matrix} -1 \\ 1 \end{matrix}$ -135 	-39
	-6	122	19	-135	

To determine the leaving variable, the entering variable must be made as large in absolute value as possible. In our example, y_{14} must be negative, so its new value is -35. This leaves y_{11} with a value of zero, thus making it the leaving variable. The new tableau is:

	b ₁ = -1	b ₂ = 1	b ₃ = 1	b ₄ = -1	
a ₁ = 0	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0+1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0+1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0+1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ -35	-35
a ₂ = 0	$\begin{matrix} -1 \\ 1 \end{matrix}$ -6	$\begin{matrix} -1 \\ 1 \end{matrix}$ 80	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0+1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0+1) \leq 1$ ✓	74
a ₃ = 0	$\begin{matrix} -1 \\ 1 \end{matrix}$ $-1 \leq (0+1) \leq 1$ ✓	$\begin{matrix} -1 \\ 1 \end{matrix}$ 42	$\begin{matrix} -1 \\ 1 \end{matrix}$ 19	$\begin{matrix} -1 \\ 1 \end{matrix}$ -100	-39
	-6	122	19	-135	

Since all sums of a_i 's and b_j 's fall between the costs/weights, this tableau represents an optimal solution. Therefore, the solution matrix is represented by:

783 + 0	7426 + 0	4709 + 0	2145 + -35
517 + -6	928 + 80	622 + 0	703 + 0
207 + 0	373 + 42	337 + 19	425 + -100

or, finally:

783	7426	4709	2110
511	1008	622	703
207	415	356	325

which gives desired row and column sums.

Example 9: Sample Problem with Varying Weights

One of the main advantages of using the modified transportation model is the ability to assign different values to the positive and negative costs/weights. This is not true with the preceding algorithms. We will solve the sample problem, applying various changes in costs/weights:

A=

783	7426	4709	2145
517	928	622	703
207	373	337	425

with:

$$v = \{1501, 8849, 5687, 3138\}$$

and

$$u = \{15028, 2844, 1303\}$$

W=

-1 or 75	-455 or 1	-1 or 358	-176 or 1
-52 or 1	-1 or 95	-56 or 1	-1 or 70
-1 or 19	-38 or 1	-1 or 31	-39 or 1

Solution:

	1	2	3	4	
1	$\frac{-1}{75}$ -35	$\frac{-455}{1}$ -----	$\frac{-1}{358}$ -----	$\frac{-176}{1}$ -----	-35
2	$\frac{-52}{1}$ 29	$\frac{-1}{95}$ 45	$\frac{-56}{1}$ -----	$\frac{-1}{70}$ -----	74
3	$\frac{-1}{19}$ -----	$\frac{-38}{1}$ 77	$\frac{-1}{31}$ 19	$\frac{-39}{1}$ -135	-39
	-6	122	19	-135	

	$b_1 = -1$	$b_2 = 93$	$b_3 = 123$	$b_4 = 53$	
$a_1 = 0$	$\begin{array}{ c } \hline -1 \\ \hline 75 \\ \hline \end{array}$ -35	$\begin{array}{ c } \hline -455 \\ \hline 1 \\ \hline \end{array}$ $(0+93) > 1$ entering variable	$\begin{array}{ c } \hline -1 \\ \hline 358 \\ \hline \end{array}$	$\begin{array}{ c } \hline -176 \\ \hline 1 \\ \hline \end{array}$	-35
$a_2 = 2$	$\begin{array}{ c } \hline -52 \\ \hline 1 \\ \hline \end{array}$ 29	$\begin{array}{ c } \hline -1 \\ \hline 95 \\ \hline \end{array}$ 45	$\begin{array}{ c } \hline -56 \\ \hline 1 \\ \hline \end{array}$	$\begin{array}{ c } \hline -1 \\ \hline 70 \\ \hline \end{array}$	74
$a_3 = -92$	$\begin{array}{ c } \hline -1 \\ \hline 19 \\ \hline \end{array}$	$\begin{array}{ c } \hline -38 \\ \hline 1 \\ \hline \end{array}$ 77	$\begin{array}{ c } \hline -1 \\ \hline 31 \\ \hline \end{array}$ 19	$\begin{array}{ c } \hline -39 \\ \hline 1 \\ \hline \end{array}$ -135	-39
	-6	122	19	-135	

	$b_1 = -1$	$b_2 = 1$	$b_3 = -1$	$b_4 = -39$	
$a_1 = 0$	$\begin{array}{ c } \hline -1 \\ \hline 75 \\ \hline \end{array}$ -61	$\begin{array}{ c } \hline -455 \\ \hline 1 \\ \hline \end{array}$ 26	$\begin{array}{ c } \hline -1 \\ \hline 358 \\ \hline \end{array}$ $-1 \leq (0+1) \leq 358$ ✓	$\begin{array}{ c } \hline -176 \\ \hline 1 \\ \hline \end{array}$ $-176 \leq (0+39) \leq 1$ ✓	-35
$a_2 = 2$	$\begin{array}{ c } \hline -52 \\ \hline 1 \\ \hline \end{array}$ 55	$\begin{array}{ c } \hline -1 \\ \hline 95 \\ \hline \end{array}$ $-1 \leq (2+1) \leq 95$ ✓	$\begin{array}{ c } \hline -56 \\ \hline 1 \\ \hline \end{array}$ 19	$\begin{array}{ c } \hline -1 \\ \hline 70 \\ \hline \end{array}$ $(2+39) < -1$ entering variable	74
$a_3 = 0$	$\begin{array}{ c } \hline -1 \\ \hline 19 \\ \hline \end{array}$	$\begin{array}{ c } \hline -38 \\ \hline 1 \\ \hline \end{array}$ 96	$\begin{array}{ c } \hline -1 \\ \hline 31 \\ \hline \end{array}$	$\begin{array}{ c } \hline -39 \\ \hline 1 \\ \hline \end{array}$ -135	-39
	-6	122	19	-135	

	$b_1 = -1$	$b_2 = 1$	$b_3 = -1$	$b_4 = -3$	
$a_1 = 0$	$\begin{array}{ c } \hline -1 \\ \hline 75 \\ \hline \end{array}$ -157	$\begin{array}{ c } \hline -455 \\ \hline 1 \\ \hline \end{array}$ 122	$\begin{array}{ c } \hline -1 \\ \hline 358 \\ \hline \end{array}$ $-1 \leq (0+1) \leq 358$ ✓	$\begin{array}{ c } \hline -176 \\ \hline 1 \\ \hline \end{array}$ $-176 \leq (0+3) \leq 1$ ✓	-35
$a_2 = 2$	$\begin{array}{ c } \hline -52 \\ \hline 1 \\ \hline \end{array}$ 151	$\begin{array}{ c } \hline -1 \\ \hline 95 \\ \hline \end{array}$ $-1 \leq (2+1) \leq 95$ ✓	$\begin{array}{ c } \hline -56 \\ \hline 1 \\ \hline \end{array}$ 19	$\begin{array}{ c } \hline -1 \\ \hline 70 \\ \hline \end{array}$ -96	74
$a_3 = -36$	$\begin{array}{ c } \hline -1 \\ \hline 19 \\ \hline \end{array}$ $(-36+1) < -1$ entering variable	$\begin{array}{ c } \hline -38 \\ \hline 1 \\ \hline \end{array}$	$\begin{array}{ c } \hline -1 \\ \hline 31 \\ \hline \end{array}$	$\begin{array}{ c } \hline -39 \\ \hline 1 \\ \hline \end{array}$ -39	-39
	-6	122	19	-135	

	$b_1 = -1$	$b_2 = 1$	$b_3 = -1$	$b_4 = -3$	
$a_1 = 0$	-1 75 -157	-455 1 122	-1 358 $-1 \leq (0 + -1) \leq 358$ ✓	-176 1 $-176 \leq (0 + -3) \leq 1$ ✓	-35
$a_2 = 2$	-52 1 190	-1 95 $-1 \leq (2 + 1) \leq 95$ ✓	-56 1 19	-1 70 -135	74
$a_3 = 0$	-1 19 -39	-38 1 $-38 \leq (0 + 1) \leq 1$ ✓	-1 31 $-1 \leq (0 + -1) \leq 31$ ✓	-39 1 $-39 \leq (0 + -3) \leq 1$ ✓	-39
	-6	122	19	-135	

Since all sums of a_i 's and b_j 's fall between the costs/weights, this tableau represents an optimal solution. Therefore, the final solution matrix is represented by:

$783 + -157$	$7426 + 122$	$4709 + 0$	$2145 + 0$
$517 + 190$	$928 + 0$	$622 + 19$	$703 + -135$
$207 + -39$	$373 + 0$	$337 + 0$	$425 + 0$

or, finally:

626	7548	4709	2145
707	928	641	568
168	373	337	425

which gives desired row and column sums.

A Note About EXCEL Software:

Microsoft's EXCEL software was chosen for testing these methods for a variety of reasons. Although code is readily available for similar algorithms in various programming languages, the applications of matrix balancing problems are seldom taken into consideration. Most applications are in business and industry, fields where this particular software is already being used and is easy to access. The algorithms can be run on any computer with simple business/pc software.

The use of VisualBasic as a programming language for writing macros in EXCEL made the testing of the algorithms a relatively simple procedure. The code in Appendix A is taken directly from each of the macros written for EXCEL; the only changes that might be made are to the actual worksheets and cells that are used for input/output of the matrices. Appendix B includes the actual result matrices from the worksheets in EXCEL.

Observations:

In order to determine the accuracy and/or efficiency of any of these algorithms, they need to be tested on matrices with various weights and of various sizes. Appendix A includes the code for each algorithm, written as a macro in EXCEL software. Appendix B includes the results from the various test matrices. Some obvious differences exist between these algorithms.

First, the initial three algorithms were developed when speed and efficiency of computing systems was a viable consideration in the use of any algorithm. The Least Squares Algorithm was developed during a period when the drudgery of calculations needed mostly to be done by hand. The RAS and Modified RAS Algorithms were later developments, but speed (and accuracy) of computers was still an issue. All three original algorithms minimize the same basic penalty function, which may or may not be worth considering. Also, each of these algorithms not only returns similar results, but also triggers a change in each variable of the original matrix. The solutions are almost always decimal solutions, another factor to be considered in terms of practicality.

The final method, the Modified Transportation Model, minimizes a completely different penalty function. It creates a change to the original matrix in only $m+n-1$ cells, and returns values for those cells that are integer solutions, two important considerations. It may be more cost-effective to minimize the number of alterations to the original matrix. One must also consider the feasibility of non-integer solutions.

Most applications lend themselves to whole number answers. In each of the examples of the applications of matrix balancing, it appears that the majority of the time, integer solutions should be returned. Since the original matrix entries themselves were integers, it would make sense that integer solutions would be desired. For this reason, the Modified Transportation Model, although not always the fastest, will give the best solution.

Dual variables can have significant economic interpretations. What appears to be a minimal change/cost to a mathematician can have an extreme impact on the financial operations of a large business or industry.

In addition to returning integer solutions, it should be noted that a minimum number of cells are altered from their original matrix values. In the case of the SAM, many original entries changing can produce a larger overall disruption to the original intent of the matrix.

The final advantage that the MTM has over the other algorithms is its ability to differentiate between positive and negative costs/weights. This is an important observation. In reality, a positive change and a negative change are rarely considered of equal importance. Situations generally do not uniformly consider an "overage" to be of the same cost as a "shortage."

The tests that were run on the algorithms showed very little difference between the results of the first three algorithms; data has shown that any one is as effective as another. The MTM algorithm returned results quite different from the others. With the ready availability of efficient computer software, the choice is no longer determined by speed, but by usability of results.


References:

- SCHNEIDER, MICHAEL H., AND ZENIOS, STRAVROS A. 1989. A Comparative Study of Algorithms for Matrix Balancing. *Operations Research*. 38, 439-455
- BACHEM, A. AND KORTE, B. 1979. On the RAS-Algorithm. Report No. D-5300, Institut für Okonometrie und Operations Research. Rheinische Friedrich-Wilhelms-Universität Bonn
- GRAD, J. 1970. Matrix Balancing. *The Computer Journal*. 14, 280-284
- MARSHALL, ALBERT W., AND OLKIN, INGRAM. 1968. Scaling of Matrices to Achieve Specified Row and Column Sums. *Numerische Mathematik*. 12, 83-90
- STEPHAN, FREDERICK F. 1942. An Iterative Method of Adjusting Sample Frequency Tables When Expected Marginal Totals Are Known. *Annals of Mathematical Statistics* 13, 166-178
- DEMING, W. EDWARDS, AND STEPHAN, FREDERICK F. 1940. On a Least Squares Adjustment of a Sampled Frequency Table When the Expected Marginal Totals Are Known. *Annals of Mathematical Statistics*. 11, 427-444



Appendix A

VisualBasic
Code for Matrix
Balancing
Algorithms
(for use with
EXCEL
software)




```

Sub LSA0
' LSA Macro
' The Least Squares Algorithm for balancing a matrix
' Macro recorded 6/28/2000 by Barbara Carothers
' Maximum matrix size - 200x200
' Keyboard Shortcut: Ctrl+l

Dim m As Integer, n As Integer, i As Integer, j As Integer, k As Integer
Dim a(0 To 200, 0 To 200) As Variant, result(0 To 200, 0 To 200) As Variant
Dim W(0 To 200, 0 To 200) As Variant
Dim WRowsum(0 To 200) As Variant, WColsum(0 To 200) As Variant
Dim u(0 To 200) As Variant, v(0 To 200) As Variant
Dim y(0 To 200) As Variant, z(0 To 200) As Variant
Dim test As Variant
Dim udiff(0 To 200) As Variant, vdiff(0 To 200) As Variant
Dim rowsum(0 To 200) As Variant, colsum(0 To 200) As Variant

m = Worksheets(1).Cells(11, 3) 'input number of rows'
n = Worksheets(1).Cells(12, 3) 'input number of columns'

test = 1
For i = 1 To m
  For j = 1 To n
    a(i, j) = Worksheets(1).Cells(15 + i, 2 + j) 'input original "A" matrix'
    result(i, j) = Worksheets(1).Cells(15 + i, 2 + j) 'initialize result matrix to "A" matrix'
    W(i, j) = Worksheets(1).Cells(39 + i, 2 + j) 'input weight matrix'
  Next j
Next i
For i = 1 To m
  u(i) = Worksheets(1).Cells(15 + i, 1) 'input desired row sums'
  y(i) = 0 'initialize iterative y-variable'
  WRowsum(i) = 0 'initialize weight row-sum variable'
Next i
For j = 1 To n
  v(j) = Worksheets(1).Cells(14, 2 + j) 'input desired column sums'
  z(j) = 0 'initialize iterative z-variable'
  WColsum(j) = 0 'initialize weight column-sum variable'
Next j

For i = 1 To m
  For j = 1 To n
    WRowsum(i) = WRowsum(i) + W(i, j) 'calculate weight matrix row sum'
  Next j
Next i
For j = 1 To n
  For i = 1 To m
    WColsum(j) = WColsum(j) + W(i, j) 'calculate weight matrix column sum'
  Next i
Next j

```

```

    'begin iterative loop'
For k = 1 To 100
test = 1
For i = 1 To m
    rowsum(i) = 0    'initialize variable for summing rows'
Next i
For j = 1 To n
    colsum(j) = 0    'initialize variable for summing columns'
Next j
For i = 1 To m
    For j = 1 To n
        rowsum(i) = rowsum(i) + result(i, j) 'calculate current row sum'
    Next j
Next i
For j = 1 To n
    For i = 1 To m
        colsum(j) = colsum(j) + result(i, j) 'calculate current column sum'
    Next i
Next j
For i = 1 To m
    udiff(i) = u(i) - rowsum(i) 'calculate the difference between desired and actual row sums'
Next i
For j = 1 To n
    vdiff(j) = v(j) - colsum(j) 'calculate the difference between desired and actual column sums'
Next j
For i = 1 To m
    y(i) = y(i) + (udiff(i) / WRowsum(i)) 'calculate the value for the iterative y-variable'
Next i
For j = 1 To n
    z(j) = z(j) + (vdiff(j) / WColsum(j)) 'calculate the value for the iterative z-variable'
Next j

For i = 1 To m
    For j = 1 To n
        result(i, j) = a(i, j) + W(i, j) * (y(i) + z(j)) 'update the A matrix'
    Next j
Next i
For i = 1 To m
    test = test + udiff(i)
Next i
For j = 1 To n
    test = test + vdiff(j)
Next j
test = test - 1
Next k
For i = 1 To m
    For j = 1 To n
        Worksheets(1).Cells(63 + i, 2 + j) = result(i, j) 'write results to worksheet'
    Next j
Next i
End Sub

```

```

Sub RAS()
' The RAS algorithm for balancing a matrix
' RAS Macro
' Maximum matrix size – 200 x 200
' Macro recorded 6/28/2000 by Barbara Carothers
' Keyboard Shortcut: Ctrl+r
Dim m As Integer, n As Integer, i As Integer, j As Integer, k As Integer
Dim a(0 To 200, 0 To 200) As Variant, result(0 To 200, 0 To 200) As Variant
Dim u(0 To 200) As Variant, v(0 To 200) As Variant
Dim rowsum(0 To 200) As Variant, colsum(0 To 200) As Variant

m = Worksheets(1).Cells(11, 3) 'input the number of rows'
n = Worksheets(1).Cells(12, 3) 'input the number of columns'

For i = 1 To m
  For j = 1 To n
    a(i, j) = Worksheets(1).Cells(15 + i, 2 + j) 'input the original "A" matrix'
    result(i, j) = Worksheets(1).Cells(15 + i, 2 + j) 'initialize the result matrix to A'
  Next j
Next i

For i = 1 To m
  rowsum(i) = 0 'initialize the variable for summing rows'
  u(i) = Worksheets(1).Cells(15 + i, 1) 'input the desired row sums'
Next i

For j = 1 To n
  colsum(j) = 0 'initialize the variable for summing columns'
  v(j) = Worksheets(1).Cells(14, 2 + j) 'input the desired column sums'
Next j

For k = 1 To 100
  For i = 1 To m
    rowsum(i) = 0
  Next i
  For j = 1 To n
    colsum(j) = 0
  Next j

  For i = 1 To m
    For j = 1 To n
      rowsum(i) = rowsum(i) + result(i, j) 'calculate current row sums'
    Next j
  Next i
  For j = 1 To n
    For i = 1 To m
      colsum(j) = colsum(j) + result(i, j) 'calculate current column sums'
    Next i
  Next j

```

```

For i = 1 To m
  For j = 1 To n
    result(i, j) = result(i, j) * (u(i) / rowsum(i)) 'adjust by rows'
  Next j
Next i

For i = 1 To m
  rowsum(i) = 0
Next i
For j = 1 To n
  colsum(j) = 0
Next j

For i = 1 To m
  For j = 1 To n
    rowsum(i) = rowsum(i) + result(i, j)
  Next j
Next i
For j = 1 To n
  For i = 1 To m
    colsum(j) = colsum(j) + result(i, j)
  Next i
Next j

For j = 1 To n
  For i = 1 To m
    result(i, j) = result(i, j) * (v(j) / colsum(j)) 'adjust by columns'
  Next i
Next j
Next k
For i = 1 To m
  For j = 1 To n
    Worksheets(1).Cells(87 + i, 2 + j) = result(i, j) 'write final result to worksheet'
  Next j
Next i

End Sub

```

```

Sub MRAS()
' The Modified RAS algorithm for balancing a matrix
' MRAS Macro
' Macro recorded 6/29/2000 by Barbara Carothers
' Maximum matrix size - 200 x 200

```

```

Dim m As Integer, n As Integer, i As Integer, j As Integer, k As Integer
Dim a(0 To 200, 0 To 200) As Variant, result(0 To 200, 0 To 200) As Variant
Dim u(0 To 200) As Variant, v(0 To 200) As Variant
Dim r(0 To 200) As Variant, s(0 To 200) As Variant
Dim rowsum(0 To 200) As Variant, colsum(0 To 200) As Variant

```

```

m = Worksheets(1).Cells(11, 3)
n = Worksheets(1).Cells(12, 3)

```

```

For i = 1 To m
  For j = 1 To n
    a(i, j) = Worksheets(1).Cells(15 + i, 2 + j)
    result(i, j) = Worksheets(1).Cells(15 + i, 2 + j)
  Next j
Next i

```

```

For i = 1 To m
  u(i) = Worksheets(1).Cells(15 + i, 1)
  rowsum(i) = 0
Next i
For j = 1 To n
  v(j) = Worksheets(1).Cells(14, 2 + j)
  colsum(j) = 0
Next j

```

```

For k = 1 To 100
  r(1) = 1
  For i = 1 To m
    rowsum(i) = 0
  Next i
  For i = 1 To m
    For j = 1 To n
      rowsum(i) = rowsum(i) + result(i, j)
    Next j
  Next i
  For i = 2 To m
    r(i) = u(i) / rowsum(i)
  Next i

```

```

For i = 1 To m
  For j = 1 To n
    result(i, j) = r(i) * result(i, j)
  Next j
Next i

```

```

For j = 1 To n
    colsum(j) = 0
Next j

For j = 1 To n
    For i = 1 To m
        colsum(j) = colsum(j) + result(i, j)
    Next i
Next j
For j = 1 To n
    s(j) = v(j) / colsum(j)
Next j
For j = 1 To n
    For i = 1 To m
        result(i, j) = result(i, j) * s(j)
    Next i
Next j

Next k

For i = 1 To m
    For j = 1 To n
        Worksheets(1).Cells(111 + i, 2 + j) = result(i, j)
    Next j
Next i

End Sub

```

```

Sub MTM()
'The Modified Transportation Model for Balancing a Matrix
'MTM Macro
'Macro recorded 7/6/2000 by Barbara Carothers

Dim m As Integer, n As Integer
m = Worksheets(1).Cells(11, 3)
n = Worksheets(1).Cells(12, 3)
Dim C(0 To 200, 0 To 200) As Variant, y(0 To 200, 0 To 200) As Variant
Dim u(0 To 200) As Variant, v(0 To 200) As Variant
Dim i As Integer, j As Integer, k As Integer
Dim Wp(0 To 200, 0 To 200) As Variant, Wn(0 To 200, 0 To 200) As Variant
Dim rowsum(0 To 200) As Variant, colsum(0 To 200) As Variant
Dim udiff(0 To 200) As Variant, vdiff(0 To 200) As Variant
Dim bv(0 To 200, 0 To 200) As Integer
Dim b(0 To 200) As Variant, a(0 To 200) As Variant
Dim ol As Integer, ow As Integer, st As Integer, ak(0 To 200) As Integer, bk(0 To 200) As Integer
Dim optimal As Integer, LoopSize As Integer
Dim lpv(0 To 200, 0 To 200) As Variant, entering(0 To 200, 0 To 200) As Variant
Dim checkCols As Integer, checkRow As Integer
Dim Posi As Integer, Posj As Integer, ni As Integer, nj As Integer
Dim LgstNegative As Variant, LgstPositive As Variant, SmlstPositive As Variant, SmlstNegative As Variant
Dim LoopSize As Integer
Dim previ As Integer, prevj As Integer
Dim used(0 To 200, 0 To 200) As Integer
Dim lasti As Integer, lastj As Integer, q As Integer
Dim lpi As Integer, lpj As Integer, spi As Integer, spj As Integer
Dim lni As Integer, lnj As Integer, sni As Integer, snj As Integer

For i = 1 To m
    u(i) = Worksheets(1).Cells(15 + i, 1) 'input desired row sums'
Next i
For j = 1 To n
    v(j) = Worksheets(1).Cells(14, 2 + j) 'input desired column sums'
Next j

optimal = 0
For i = 1 To m
    For j = 1 To n
        C(i, j) = Worksheets(1).Cells(15 + i, 2 + j) 'input original matrix'
        Wp(i, j) = Worksheets(1).Cells(39 + i, 2 + j) 'input positive weight matrix'
        If Worksheets(1).Cells(39 + i, 23 + j) < 0 Then
            Wn(i, j) = Worksheets(1).Cells(39 + i, 23 + j) 'input negative weight matrix, if given'
        Else
            Wn(i, j) = -Wp(i, j) 'if not given, negative weight matrix is the opposite of the positive weight matrix'
        End If
        Worksheets(2).Cells(30 + i, 2 + j) = Wn(i, j)
    End If
Next j
Next i

```

```

For i = 1 To m
    rowsum(i) = 0      'initialize row sum variable'
Next i
For j = 1 To n
    colsum(j) = 0     'initialize column sum variable'
Next j

For i = 1 To m
    For j = 1 To n
        rowsum(i) = rowsum(i) + C(i, j) 'calculate row sum of original matrix'
    Next j
Next i
For j = 1 To n
    For i = 1 To m
        colsum(j) = colsum(j) + C(i, j) 'calculate column sum of original matrix'
    Next i
Next j

For i = 1 To m
    udiff(i) = u(i) - rowsum(i) 'calculate the difference between original and desired row sums'
Next i
For j = 1 To n
    vdifff(j) = v(j) - colsum(j) 'calculate the difference between original and desired column sums'
Next j

'calculate initial basic feasible solution'
i = 1: j = 1
Do
    If udiff(i) < vdifff(j) Then 'use the smallest possible value for each yij until the last column/row is
reached'
        y(i, j) = udiff(i)
        vdifff(j) = vdifff(j) - udiff(i)
        bv(i, j) = 1          'indicates that this is now a basic variable'
        udiff(i) = 0
        i = i + 1
    ElseIf udiff(i) > vdifff(j) Then
        y(i, j) = vdifff(j)
        udiff(i) = udiff(i) - vdifff(j)
        bv(i, j) = 1
        vdifff(j) = 0
        j = j + 1
    Else
        y(i, j) = udiff(i)    'in the instance of a degenerate matrix, a zero must be used as a basic variable'
        vdifff(j) = 0
        bv(i, j) = 1
        udiff(i) = 0
        If (n - j) > (m - i) Then
            j = j + 1
        Else
            i = i + 1

```



```

End If
End If
Loop Until i > m - 1 Or j > n - 1

If i > m - 1 Then          'determine the final row/column of the initial basic feasible solution'
  ow = 1
  st = j
ElseIf j > n - 1 Then
  ol = 1
  st = i
End If
If ow = 1 Then
  For j = st To n
    y(i, j) = vdiff(j)
    bv(i, j) = 1
  Next j
ElseIf ol = 1 Then
  For i = st To m
    y(i, j) = udiff(i)
    bv(i, j) = 1
  Next i
End If

'begin the iterative loop'
Do
'determine the values for the dual variables a() and b()'
For i = 1 To m
  ak(i) = 0      'ak indicates that a is known, if its value is 1'
Next i
For j = 1 To n
  bk(j) = 0      'bk indicates that b is known, if its value is 1'
Next j

a(1) = 0
ak(1) = 1
For k = 1 To m + n - 1
  For i = 1 To m
    For j = 1 To n
      If bv(i, j) = 1 Then
        If ak(i) = 1 Then
          If bk(j) = 0 Then
            If y(i, j) >= 0 Then
              b(j) = Wp(i, j) - a(i)
              bk(j) = 1
              s = s + 1
            End If
          End If
        End If
      End If
    Next j
  Next i
  If ak(i) = 0 Then
    If bk(j) = 1 Then
      If y(i, j) >= 0 Then

```

```

    a(i) = Wp(i, j) - b(j)
    ak(i) = 1
    s = s + 1
  End If
End If
End If
If ak(i) = 1 Then
  If bk(j) = 0 Then
    If y(i, j) < 0 Then
      b(j) = Wn(i, j) - a(i)
      bk(j) = 1
      s = s + 1
    End If
  End If
End If
If ak(i) = 0 Then
  If bk(j) = 1 Then
    If y(i, j) < 0 Then
      a(i) = Wn(i, j) - b(j)
      ak(i) = 1
      s = s + 1
    End If
  End If
End If
End If
Next j
Next i
Next k

```

'determine the entering variable/check for optimality'

```

ei = 0
ej = 0
For i = 1 To m
  For j = 1 To n
    If a(i) + b(j) > Wp(i, j) Then
      ev = 1: ei = i: ej = j
    ElseIf a(i) + b(j) < Wn(i, j) Then
      ev = -1: ei = i: ej = j
    End If
  Next j
Next i
If ei = 0 And ej = 0 Then
  optimal = 1
  GoTo Line1
End If
Line2:
For i = 1 To m
  For j = 1 To n
    If ei = i And ej = j Then
      bv(i, j) = 2
    End If
  Next j
Next i

```

```
Next j
Next i
```

```
'find the loop'
For i = 1 To m
  For j = 1 To n
    If bv(i, j) = 2 Then
      lpv(i, j) = 1
    End If
  Next j
Next i
```

```
q = 1: lasti = ei: lastj = ej: k = 2
```

```
Do
  If q = 1 Then
    Do
      For j = 1 To n
        If bv(lasti, j) = 1 And used(lasti, j) = 0 And j <> lastj Then
          lpv(lasti, j) = k
          used(lasti, j) = 1
          lastj = j
          k = k + 1
          q = -q
        Exit Do
      End If
    Next j
    For i = 1 To m
      For j = 1 To n
        If lasti = i And lastj = j Then
          If bv(i, j) <> 2 Then
            lpv(i, j) = -1
            used(i, j) = -1
          End If
        Else
          If bv(i, j) = 1 And lpv(i, j) <> -1 Then
            lpv(i, j) = 0
            used(i, j) = 0
          End If
        End If
      Next j
    Next i
    k = 2
    q = 1
    lasti = ei
    lastj = ej
  Exit Do
Loop
```

```
ElseIf q = -1 Then
  Do
    For i = 1 To m
```

```

If bv(i, lastj) = 1 And used(i, lastj) = 0 And i <> lasti Then
  lpv(i, lastj) = k
  k = k + 1
  q = -q
  lasti = i
  used(i, lastj) = 1
Exit Do
ElseIf bv(i, lastj) = 2 Then
  endpt = 1
Exit Do
End If
Next i
For i = 1 To m
  For j = 1 To n
    If i = lasti And j = lastj Then
      used(i, j) = -1
      lpv(i, j) = -1
    ElseIf bv(i, j) = 1 And lpv(i, j) <> -1 Then
      used(i, j) = 0
      lpv(i, j) = 0
    End If
  Next j
Next i
k = 2
q = 1
lasti = ei
lastj = ej
Exit Do
Loop
End If
Loop Until endpt = 1
For i = 1 To m
  For j = 1 To n
    If lpv(i, j) > 0 Then
      LoopSize = LoopSize + 1
    End If
  Next j
Next i
'determine the variable leaving the basis'
If ev = 1 Then
  addme = 0
  LgstPositive = 0
  LgstNegative = -500000
  SmlstPositive = 500000
  SmlstNegative = 0
  For i = 1 To m
    For j = 1 To n
      used(i, j) = 0
    Next j
  Next i
  For k = 3 To LoopSize Step 2

```

```

For i = 1 To m
  For j = 1 To n
    If lpv(i, j) = k And y(i, j) >= 0 Then
      If y(i, j) > LgstPositive And used(i, j) = 0 Then
        LgstPositive = y(i, j)
        lpi = i
        lpj = j
        used(i, j) = 1
      End If
    End If
  Next j
Next i
Next k
For k = 3 To LoopSize Step 2
  For i = 1 To m
    For j = 1 To n
      If lpv(i, j) = k And y(i, j) < 0 Then
        If y(i, j) > LgstNegative And used(i, j) = 0 Then
          LgstNegative = y(i, j)
          lni = i
          lnj = j
          used(i, j) = 1
        End If
      End If
    Next j
  Next i
Next k
If LgstNegative = -500000 Then
  addme = LgstPositive
  li = lpi
  lj = lpj
Else
  addme = Abs(LgstNegative)
  li = lni
  lj = lnj
End If
For k = 2 To LoopSize Step 2
  For i = 1 To m
    For j = 1 To n
      If lpv(i, j) = k And y(i, j) >= 0 Then
        If y(i, j) < SmlstPositive And used(i, j) = 0 Then
          SmlstPositive = y(i, j)
          spi = i
          spj = j
          used(i, j) = 1
        End If
      End If
    Next j
  Next i
Next k
If SmlstPositive < addme Then

```

```

    addme = SmlstPositive
    li = spi
    lj = spj
End If
ElseIf ev = -1 Then
addme = 0
LgstPositive = 0
LgstNegative = -500000
SmlstPositive = 500000
SmlstNegative = 0
For i = 1 To m
    For j = 1 To n
        used(i, j) = 0
    Next j
Next i
For k = 3 To LoopSize Step 2
    For i = 1 To m
        For j = 1 To n
            If lpv(i, j) = k And y(i, j) < 0 Then
                If y(i, j) < SmlstNegative And used(i, j) = 0 Then
                    SmlstNegative = y(i, j)
                    sni = i
                    snj = j
                    used(i, j) = 1
                End If
            End If
        Next j
    Next i
Next k
For k = 3 To LoopSize Step 2
    For i = 1 To m
        For j = 1 To n
            If lpv(i, j) = k And y(i, j) >= 0 Then
                If y(i, j) < SmlstPositive And used(i, j) = 0 Then
                    SmlstPositive = y(i, j)
                    spi = i
                    spj = j
                    used(i, j) = 1
                End If
            End If
        Next j
    Next i
Next k
If SmlstPositive = 500000 Then
    addme = SmlstNegative
    li = sni
    lj = snj
Else
    addme = -(SmlstPositive)
    li = spi
    lj = spj

```

```

End If
For k = 2 To LoopSize Step 2
  For i = 1 To m
    For j = 1 To n
      If lpv(i, j) = k And y(i, j) < 0 Then
        If y(i, j) > LgstNegative And used(i, j) = 0 Then
          LgstNegative = y(i, j)
          lni = i
          lnj = j
          used(i, j) = 1
        End If
      End If
    Next j
  Next i
Next k
If LgstNegative > addme Then
  addme = LgstNegative
  li = lni
  lj = lnj
End If
End If


```

```

'adjust the loop'
For k = 1 To LoopSize Step 2
  For i = 1 To m
    For j = 1 To n
      If lpv(i, j) = k Then
        y(i, j) = y(i, j) + addme
      End If
    Next j
  Next i
Next k
For k = 2 To LoopSize Step 2
  For i = 1 To m
    For j = 1 To n
      If lpv(i, j) = k Then
        y(i, j) = y(i, j) - addme
      End If
    Next j
  Next i
Next k
'reset the loop variables'
For i = 1 To m
  For j = 1 To n
    If i = li And j = lj Then
      bv(i, j) = 0
    ElseIf i = ei And j = ej Then
      bv(i, j) = 1
    End If
  Next j
Next i

```

```
'end the iterative loop'  
Line1:  
Loop Until optimal = 1  
  
'print results to worksheet (1)'  
For i = 1 To m  
  For j = 1 To n  
    Worksheets(1).Cells(135 + i, 2 + j) = C(i,j) + y(i, j)  
  Next j  
Next i  
  
End Sub
```

Appendix B

Test Matrices and Results

MATRIX BALANCING – A COMPARATIVE STUDY

Test Matrices and Results

Matrix A: Sample Matrix

Matrix Balancing Algorithms

B. Carothers
July, 2000

Macros:

LSA -- Least Squares Algorithm (Deming/Stephan 1945)

RAS -- RAS Algorithm (Bachem/Korte 1972)

MRAS -- Modified RAS Algorithm (Bachem/Korte 1975)

MTM -- Modified Transportation Model (Ritchey 2000)

no of rows:



no of columns:

desired	col.sums:	1501	6849	5687	3188
rowsums:	A Matrix:				
15028		783	7476	4709	2145
2844		517	328	622	1703
1303		207	373	337	425

enter differing negative weights, if applicable, in the matrix to the right

Weight Matrix:

75	455	358	176
52	95	96	70
19	38	31	39

Result Matrix--Using LSA:

771.2163	7496.875	4710.899	2048.809
528.8828	379.4331	643.9081	1691.776
200.9009	372.6914	332.0325	437.8157

Result Matrix--RAS Algorithm:

771.3012	7603.8535	4709.117	2041.629
528.8357	373.7793	643.9095	1691.8011
200.8873	373.6889	331.8775	437.8239

Result Matrix--Modified RAS Algorithm:

771.9042	7503.953	1709.17	2020.829
628.8156	3747.779	645.31	837.111
200.8663	171.2889	131.277	248.171

Result Matrix--Modified Transportation Model:

748	7426	1709	2015
646	354	641	837
207	409	131	201

Comments/Observations:

Although all four algorithms return solutions, there are differences. It should be noted that the RAS and Modified RAS Algorithms return the same solution matrix. This is true for all applications.

The Least Squares Approximation returns a slightly different matrix, but the difference is not significant. The reason the matrices are so similar is because they minimize a similar penalty function, as noted earlier. The reason they are different is because the LSA uses the weight matrix, while the RAS and MRAS do not. It is interesting to note that difference is not significant, even with the weight matrix.

The Modified Transportation Model, on the other hand, returns a significantly different matrix. The cell entries are still integers, an important consideration. There are also changes to only six cells, keeping the alterations to the original matrix to a minimum.

Matrix B: Sample Matrix Without a Weight Matrix

**Matrix
Balancing
Algorithms**

**B. Carothers
July, 2000**

Macros:

LSA -- Least Squares Algorithm (Deming/Stephan 1945)

RAS -- RAS Algorithm (Bachem/Korte 1972)

MRAS -- Modified RAS Algorithm (Bachem/Korte 1975)

MTM -- Modified Transportation Model (Ritchey 2000)

no of rows:

3

no of columns:

4

desired col.sums:

1601.5 8349 6687 8138

rowsums: A Matrix:

15028
2844
1303

763 7436 4709 2145
517 1078 522 703
207 329 137 425

Weight Matrix: enter differing negative weights, if applicable, in the matrix to the right

[Empty matrix grid]

Result Matrix--Using LSA:

772.25 7457.917 4706.583 2091.25
533.6 987.1667 646.6667 676.6667
195.25 463.9167 331.9167 210.25

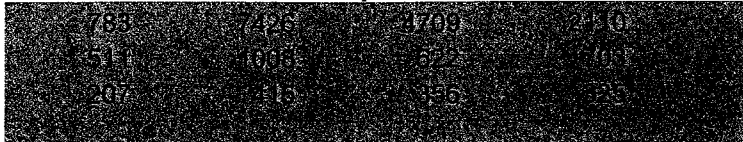
Result Matrix--RAS Algorithm:

771.3012 7603.953 4709.117 2041.629
528.6355 971.7579 645.9055 695.5011
200.6613 471.2889 335.9725 206.6702

Result Matrix--Modified RAS Algorithm:



Result Matrix--Modified Transportation Model:



Comments/Observations:

Without the weight matrix, the minor differences between the entries of the LSA result and the RAS/MRAS results seem to be larger. Perhaps that is because the weight matrix itself was estimated to be proportional to the individual cell entries.

The MTM returns a completely different matrix than with the weights added. It now makes the alterations to different cells than before, noting that the cost of changing one is the same as the cost of changing any other. It is also the only algorithm to return integer values.

Matrix C: Sample Matrix With a Varying Weight Matrix

**Matrix
Balancing
Algorithms**

**B. Carothers
July, 2000**

Macros:

LSA -- Least Squares Algorithm (Deming/Stephan 1945)

RAS -- RAS Algorithm (Bachem/Korte 1972)

MRAS -- Modified RAS Algorithm (Bachem/Korte 1975)

MTM -- Modified Transportation Model (Ritchey 2000)

no of rows:

3

no of columns:

4

desired

col.sums:

4501 8849 5687 6138

rowsums:

A Matrix:

15028
2844
1308

783	7426	4709	2145
517	928	622	703
207	378	331	425

Weight Matrix: enter differing negative weights, if applicable, in the matrix to the right

75	1	158	1
1	95	1	70
19	1	31	1

(negative only)

1	158	1	176
57	1	56	1
38	1	39	1

Result Matrix--Using LSA:

770.4922	7416.346	1718.935	12796.31
530.0235	3210.4137	1535.272	23910.01
200.2769	1061.608	32.932	10016.25

Result Matrix--RAS Algorithm:

771.3012	7501.953	1709.117	20433.29
528.8355	3737.573	1759.523	23950.01
200.8633	1112.889	31.9775	9863.62

Result Matrix--Modified RAS Algorithm:

771.3012	7501.953	1709.117	20433.29
528.8355	3737.573	1759.523	23950.01
200.8633	1112.889	31.9775	9863.62

Result Matrix--Modified Transportation Model:

626	7548	1709	9145
707	3928	1641	13568
168	1173	37	1035

Comments/Observations:

The first three algorithms cannot account for the negative weights/costs, and therefore must use only the positive costs. The only algorithm that was able to take the negative weights into account was the MTM. As expected, it also was the only algorithm to return integer values.

*Table I: Run Times for Random 50x50 Matrices
(20 random matrices and associated weights were run)*

<i>Algorithm Used:</i>	<i>Least Squares Approx.</i>	<i>RAS Algorithm</i>	<i>Modified RAS</i>	<i>Modified Transportation Model:</i>
<i>Average Run Time (in seconds):</i>	52	48.2	48.2	142.5