November, 1980

A MIN-MAX PROBLEM

by

W.L. Steiger

DCS-TR-95

W. L. Steiger
Department of Statistics, IAS,
The Australian National University,
P.O. Box 4,
Canberra. A.C.T. 2600

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, N.J. 08903

Problem 79-17 *

A MIN-MAX PROBLEM

W.L. STEIGER

Department of Statistics, IAS,
The Australian National University,
P.O. Box 4,
Canberra. A.C.T. 2600

Problem 79-17 * by W.R. Utz (University of Missouri)

"Determine an algorithm, better than complete enumeration, for the following problem: given a non-negative integer matrix, permute the entries in each column independently so as to minimize the largest row sum. This problem had arisen in determining an optimal scheduling for a factory work force."

Write the given matrix as $M = (m_{ij})$ and let $\Pi = (\underline{\pi}_1,\ldots,\underline{\pi}_n)$ be a matrix, $\underline{\pi}_j$ being a permutation of $\{1,\ldots,n\}$ that acts on column j of M. Applying Π to M gives $M(\Pi) \equiv \Pi M$ with row sums $r_i(\Pi) = \sum_{j=1}^{n} m_{\pi_j(i)\,j}$ and $\rho(\Pi) = \max r_i(\Pi)$, the largest row sum. We seek $\mu = \min \rho(\Pi)$ the min taken over the $(n!)^{n-1}$ distinct sets of column permutations, Π.

Suppose $\mu = \rho(\Pi)$ and that $\rho(\Pi) = r_p(\Pi)$; i.e., the p^{th} row of $M(\Pi)$ is the minimax row sum. The following algorithm exploits a (family of) necessary condition(s) for Π to be optimal: choose any k columns $j_1,\ldots,j_k$, $1 \le k \le [n/2]$, and any row $q \ne p$. Swapping the entries in rows p and q in columns $j_1,\ldots,j_k$ yields a matrix with $r_p \ge \rho(\Pi)$ or $r_q \ge \rho(\Pi)$. The greater k, the stronger the condition and the closer it is to sufficing for $\rho(\Pi) = \mu$. However there seems to be no simply exploitable sufficient condition for $M(\Pi)$ to be optimal.

The algorithm attains the necessary condition for optimality by using a heuristic principal of a statistical nature:

AT OPTIMUM, THE ROW SUMS MUST BE NEARLY EQUAL, SO THE ROW SUM VARIANCE IS SMALL.

Thus the algorithm iterates through permutations Π, at each step seeking to reduce row sum variance. It does this by selecting a pair

of rows, i_1, i_2 with $r_{i_1} > r_{i_2}$, and k columns, $j_1, \ldots, j_k$. It interchanges $m_{i_1 j}$ with $m_{i_2 j}$, $j = j_1, \ldots, j_k$ as long as row sum variance is reduced. If not, it tries another set of k columns. If none of the $\binom{n}{k}$ sets can reduce variance, it selects another pair of rows.

The procedure is "greedy" in three ways. A swap of some entries in rows i_1 and i_2 will reduce row sum variance greatly when $r_{i_1} - r_{i_2}$ is large. It thus examines the row pairs roughly in order of decreasing $r_{i_1} - r_{i_2}$. When a successful swap is found, the row sums are re-ranked and row pair enumeration begins afresh.

Secondly, the k columns are selected in a way that is most likely to have a large impact on $r_{i_1} - r_{i_2}$: the variance of entries in each column is constant throughout the algorithm. A pair of entries in a column are likely to be most different when that column has large variance. Thus the columns are ranked by variance. The original k columns chosen are those with the k largest variances and enumeration proceeds according to the variance rankings.

Finally, with row swaps in certain k columns, the coarsest set of row sum changes occur when $k = 1$ (there are only n possible changes); the finest set of changes arise when $k = k_{max}$ (there are $\binom{n}{k_{max}}$ possible changes). Thus the algorithm iterates from $k = 1$ through $k = k_{max} \leq [n/2]$ each successive k reducing variance less, but requiring more column enumeration in investigating potential row swaps.

The algorithm is <u>very</u> cheap, in absolute terms and especially as compared to $(n!)^{n-1}$ maximum row sum evaluations. Presumably this is due to the fact the row sum variance reduction is a potent heuristic (in comparison

to reducing $\rho(\Pi))$ and the fact that each iteration is extremely simple: the next Π is obtained from the present one by interchanging k corresponding elements from a pair of rows.

The following examples depict the performance of the algorithm on some test problems. The matrices are of random integers in the range 1,...,10000.

Problem 1

$$M = \begin{pmatrix} 850 & 4931 & 133 & 8920 \\ 9010 & 5382 & 6162 & 8214 \\ 160 & 8780 & 9505 & 4413 \\ 9202 & 5765 & 4620 & 2752 \end{pmatrix}$$

In 4 iterations, each a row interchange in a single column, the algorithm halted at

$$M(\Pi) = \begin{pmatrix} 850 & 5382 & 6162 & 8920 \\ 160 & 4931 & 9505 & 8214 \\ 9010 & 8780 & 133 & 4413 \\ 9202 & 5765 & 4620 & 2752 \end{pmatrix}$$

with row sums $\begin{pmatrix} 21314 \\ 22810 \\ 22336 \\ 22339 \end{pmatrix}$

This is an optimal $M(\Pi)$, as verified by exhaustive enumeration, and used .4 sec. of CPU time on the DEC KA10., as compared to 6.1 sec for enumeration.

Problem 2

$$M = \begin{pmatrix} 4574 & 9769 & 8063 & 8278 & 4606 & 523 & 9207 & 7722 \\ 2246 & 6699 & 7060 & 5361 & 2707 & 6125 & 921 & 6595 \\ 8339 & 3642 & 3294 & 81 & 4046 & 464 & 2001 & 1557 \\ 1668 & 7596 & 6183 & 7576 & 7756 & 478 & 472 & 6594 \\ 6858 & 5888 & 6820 & 8386 & 8988 & 3758 & 1174 & 5725 \\ 957 & 659 & 8412 & 5033 & 8198 & 537 & 217 & 6689 \\ 9059 & 3988 & 7445 & 9667 & 4960 & 7712 & 2538 & 9956 \\ 9413 & 6729 & 3390 & 606 & 7810 & 6460 & 3356 & 3021 \end{pmatrix}$$

The algorithm terminated after 46 iterations. Of these steps, 33 were row pair interchanges within a single column, 4 interchanges within a pair of columns, 5 within a trio of columns, and 4 were interchanges within $4 = n/2$ columns. The final tableau is:

$$M(\Pi) = \begin{pmatrix} 6858 & 3642 & 3294 & 7576 & 4960 & 7712 & 217 & 6595 \\ 9059 & 7596 & 3390 & 606 & 4606 & 6460 & 3356 & 5725 \\ 8339 & 5888 & 6183 & 8386 & 4046 & 478 & 921 & 6594 \\ 4574 & 6699 & 8412 & 81 & 7756 & 6125 & 472 & 6689 \\ 957 & 659 & 8063 & 9667 & 8988 & 523 & 2001 & 9956 \\ 1668 & 3988 & 7445 & 5033 & 8198 & 3758 & 9207 & 1557 \\ 2246 & 9769 & 7060 & 8278 & 2707 & 537 & 2538 & 7722 \\ 9413 & 6729 & 6820 & 5361 & 7810 & 464 & 1174 & 3021 \end{pmatrix}.$$

with row sums

$$\begin{pmatrix} 40854 \\ 40798 \\ 40835 \\ 40808 \\ 40814 \\ 40854 \\ 40857 \\ 40792 \end{pmatrix}$$

It must be close to optimum because the maximum and minimum row sums differ by only 65. It required 4.2 sec. of CPU time.

Finally, it is worth mentioning (without detail) the performance on a 12 x 12 example. Here, the algorithm terminated after 78 row swaps, 49 in a single column, 7 in a pair of columns, 18 in a trio of columns, and 4 in 4 columns. It took 28 sec. of CPU time. The final row sums were

$$\begin{pmatrix} 64085 \\ 64089 \\ 64083 \\ 64084 \\ 64083 \\ 64085 \\ 64087 \\ 64092 \\ 64084 \\ 64083 \\ 64084 \\ 64085 \end{pmatrix}$$

This must be very nearly optimal, since the max and min row sums differ by only 9. It is worth pointing out that in a sense, the task for this algorithm BECOMES EASIER as n increases. That is because, with more columns to manipulate, it is more likely that the max and min row sums will be very close, both at optimum and at the termination of the algorithm.

The details of the algorithm appear below.

.../6

THE ALGORITHM

Setup

(1) Get $r_i = \sum_{j=1}^{n} m_{ij}$, $i = 1,\ldots,n$

Get $V_j = \sum_{i=1}^{n} (m_{ij} - m_j)^2/n$, $m_j = \sum_{i=1}^{n} m_{ij}/n$,

and $j_1,\ldots,j_k : V_{j_1} \geq \ldots \geq V_{j_n}$

Set $k = 0$

Column Search Complexity

(2) $k = k + 1$; stop if $k > k_{max}$

Row Ranking

(3) Get $i_1,\ldots,i_n : r_{i_1} \geq \ldots \geq r_{i_n}$

Row Enumeration

(4) $p = 0$

(5) $p = p + 1$; if $p = n$, GO TO 2

(6) $q = n + 1$

(7) $q = q - 1$; if $q = p$, GO TO 5

Column Enumeration

(8.1) For $\ell_1 = 1$ thru $n - k + 1$

(8.2) For $\ell_2 = \ell_1 + 1$ thru $n - k + 2$

$\vdots$

(8.k) For $\ell_k = \ell_{k-1} + 1$ thru n

Test

(9) Set $d = \sum_{t=1}^{k} (m_{i_p j_{\ell_t}} - m_{i_q j_{\ell_t}})$

(10) if $r_{i_p} - r_{i_q} > d > 0$ GO TO 11, ELSE GO TO 8.k. UNTIL loops in 8, satisfied. Then GO TO 7.

Swap

(11) $m_{i_p j_{\ell_t}} \longleftrightarrow m_{i_q j_{\ell_t}}$, $t = 1,\ldots,k$

Update

(12) $r_{i_p} = r_{i_p} - d$, $r_{i_q} = r_{i_q} + d$. GO TO 3.