Rapid Optimal Scheduling for Time-Multiplex Switches Using a Cellular Automaton

CHRISTOPHER ROSE

Abstract—**Many time-multiplex switching systems require that the incoming traffic be scheduled in order to avoid conflict at the switch output (two or more users converging simultaneously upon a single output). Optimal scheduling provides a means to assign traffic on demand such that either blocking probability is minimized (unbuffered system) or packet waiting time is minimized (buffered system). However, computation of an optimal schedule for switches of a reasonable size (i.e., $N = 100$) may require many seconds or even minutes, whereas the traffic demand may vary much more rapidly. Since the computation time varies as $O(N^2)$, the problem becomes readily intractable for large N. This computational bottleneck is overcome by using a scheduling algorithm which is run on a simple special-purpose parallel computer (cellular automaton). A schedule is produced in $O(N)$ time if signal propagation time in the automaton is considered negligible, and therefore, increases in computation speed by several orders of magnitude should be possible; the time to compute a schedule for a 1000 input switch would be measured in milliseconds rather than minutes.**

I. Introduction

A time-multiplex switch routes communication traffic from its input to output by providing intervals of duration τ called *time slots* during which a fixed-length message packet is transferred from its source to its destination. The switch may not send two packets to the same output during the same time slot without the loss of one or both packets. However, since user requests for time slots are assumed to arise randomly, there is the possibility of *conflict* (two or more users converging simultaneously upon a single output). Thus, to avoid conflict, some scheduling of the packets must be done. If certain intuitive conditions on the distribution of user requests are met, then a *conflict-free* schedule may be found. Specifically, given a group of C time slots (*frame*) in which to route user packets, a total of C packets may be routed to any given output and a total of C packets may be handled by any given input. If the user requests comply with this *scheduling criterion* (i.e., no user requests more than C time slots for transmission and no output is the destination of more than C packets) then every request can be serviced [1], [2]. Optimal scheduling of the packets is a means to maximize system throughput while minimizing either packet blocking (unbuffered system) or packet delay (buffered system). For some systems, throughput improvements of 10–20 percent can be realized [3].

Given that the scheduling criterion is met, providing conflict-free (optimal) schedules[1] in response to a stochastically varying load has proven difficult using current computational techniques. For example, the time required to optimally schedule an $N \times N$ (N inputs, N outputs) switch with $N = 1000$ and $C = 100$ time slots per frame would be measured in tens of seconds and a sequential computer using an efficient algorithm.[2] In many applications such long computation time renders assignment of traffic on demand impossible.

Derived in this paper is an optimal scheduling algorithm which may be implemented with a special-purpose parallel computer (cellular automaton) and allows extremely rapid computation of schedules; the scheduling time for an $N = 1000$, $C = 100$ switch is measured in milliseconds. Owing to the parallel structure, the computation time varies approximately linearly with switch size. In what follows, the problem of finding an optimal schedule is introduced. An algorithm is then developed to find such optimal schedules and it is then shown how a cellular automaton may be used to realize the algorithm.

In addition, two interesting results emerge from this study. The first is a new proof for the existence of sets of distinct representatives [2]. The second is the discovery of a graphical form for Hall's algorithm [4] for deriving sets of distinct representatives.

II. Problem Statement

A. The Traffic Matrix and Diagonals

If the communication requirements (in packets) of N fully connected users are tabulated in matrix form, the result is a "traffic matrix," T. Each t_{ij} denotes the number of packets destined to output j from input i. A possible traffic matrix is shown in Fig. 1(a) for an $N = 3$ switch. The constraints on any $N \times N$ switch are that no two inputs may be connected to the same output simultaneously and no one input may be connected to more than one output simultaneously. These constraints suggest that no two packets sharing the same row or column of the traffic matrix may be transmitted (travel) in the same time slot. Thus, conflict-free scheduling is the problem of finding a set of "diagonals" through the traffic matrix where a diagonal is a group of elements no two of which share either a row or a column.[3] The boxed elements in the matrix of Figure 1a form a diagonal.

B. Optimal Scheduling and Maximal Diagonals

Given a traffic matrix which meets the scheduling criterion (the sum of entries in each row and the sum of entries in each column do not exceed C, the number of time slots available for transmission) there exists a schedule which carries all the offered traffic in the allotted time slots [1]. Consider then a matrix each of whose rows and columns sum to C. Each diagonal in the optimal schedule must contain an entry from every row and column. If the matrix is $N \times N$ then each

Paper approved by the Editor for Communications Switching of the IEEE Communications Society. Manuscript received September 17, 1987; revised March 31, 1988.

The author is with AT&T Bell Laboratories, Holmdel, NJ 07733.

IEEE Log Number 8926994.

[1] It is worthwhile to note that the problem considered here differs slightly from that considered by some previous workers [4]–[7]. Specifically, previous workers were also concerned with minimizing the number of different switching configurations required to support a given traffic demand. This consideration is important in scanning spot beam satellite systems [1], [4]–[7] wherein changing the switch state requires a nonnegligible amount of time; repeated unnecessary changes in switch configuration could degrade switch performance. In this study, however, a terrestrial system is assumed wherein the time required to change the switch state is assumed negligible.

[2] This rough estimate is based upon a personal conversation with T. Inukai of COMSAT Laboratories in October 1986. He recalls that his algorithms [4], implemented in Fortran IV for $N = 8$ users with C roughly equal to 4000 required tens of milliseconds to produce a schedule. The computation time is proportional to CN^2. Thus, scheduling a 1000×1000 switch with $C = 100$ would require tens of seconds. These figures, of course, are only order-of-magnitude estimates.

[3] The "diagonal" is also known as a "system of distinct representatives" in the literature [2], [4]–[7] and stems from P. Hall's [2] analysis of a similar combinatorial problem.

$$\begin{matrix} 0 & \boxed{1} & 1 \\ 1 & 0 & \boxed{1} \\ \boxed{1} & 1 & 0 \end{matrix}$$

(a)

$$\begin{array}{r|ccc} & & & \text{critical column} \\ & & & \downarrow \\ \text{critical row} \longrightarrow & 1 & 1 & 1 \\ & 1 & 0 & 1 \\ & 0 & 1 & 1 \end{array}$$

(b)

$$\begin{array}{r|ccc} & & & \text{critical column} \\ & & & \downarrow \\ \text{critical row} \longrightarrow & 1 & 1 & 1 \\ & 0 & 0 & 1 \\ & 0 & 0 & 1 \end{array}$$

(c)

Fig. 1. Illustration of valid diagonals, critical rows and columns and extraction of critical rows and columns. (a) A valid diagonal, (b) a critical row and column, (c) traffic matrix containing only the critical row and column from the matrix of part (b). All other entries are set to zero. See text for full description.

diagonal will contain N elements (have length N) and the schedule will be composed of N such diagonals. For such a matrix, all of whose rows and columns sum to C, finding diagonals of length N is a pivotal problem in procuring an optimal schedule.

For a matrix not all of whose rows and columns sum to C, finding an optimal schedule still involves finding maximum length diagonals. Specifically, let a column or row that sums to C be called *critical*. Each diagonal in the schedule must contain elements from all critical rows and columns. For an $N \times N$ matrix a diagonal of length N will certainly satisfy this condition, but a diagonal of length N may not exist. Therefore, the optimal scheduling problem becomes one of finding a diagonal which covers all the critical rows and columns. This may be done by deriving a new matrix T' whose only nonzero entries are those elements contained in the critical rows and columns of the original traffic matrix. A diagonal of maximum length through T' will cover all critical rows and columns of T.[4] Thus, the problem of optimal scheduling is intimately tied to the problem of finding maximal diagonals. Fig. 1(b) and (c) illustrate the process of deriving T'.

Finding a maximal diagonal visually in a small matrix is simple. The difficulty of finding a maximal diagonal in larger matrices may be appreciated be referring to Fig. 2(a) wherein a 10×10 matrix with only 0 and nonzero (X) entries is presented.[5] Under visual inspection, typically eight or nine

[4] If there are c critical columns and r critical rows ($r, c \leq N$) then the a diagonal of at least length max (r, c) must exist [1], [2].

[5] The values of these nonzero entries may be chosen to make the row and columns sums almost arbitrary ($\geq$ the number of nonzero entries in that row or column).

column number

row number	0	1	2	3	4	5	6	7	8	9
0	X	X	X	X	0	0	X	0	X	0
1	0	X	X	X	X	X	0	X	0	0
2	X	0	0	0	X	X	X	0	0	X
3	0	X	X	0	X	X	0	X	0	X
4	X	X	0	X	X	0	0	0	0	0
5	X	0	0	0	0	X	X	0	X	0
6	0	0	X	X	X	0	0	0	X	X
7	0	X	X	0	X	0	X	0	0	0
8	0	X	0	X	X	X	0	0	X	X
9	0	X	0	0	X	0	X	X	0	0

(a)

column number

row number	0	1	2	3	4	5	6	7	8	9
0	X	X	$\boxed{X}$	X	0	0	X	0	X	0
1	0	X	X	X	$\boxed{X}$	X	0	X	0	0
2	X	0	0	0	X	X	X	0	0	$\boxed{X}$
3	0	X	X	0	X	X	0	$\boxed{X}$	0	X
4	$\boxed{X}$	X	0	X	X	0	0	0	0	0
5	X	0	0	0	0	$\boxed{X}$	X	0	X	0
6	0	0	X	$\boxed{X}$	X	0	0	0	X	X
7	0	$\boxed{X}$	X	0	X	0	X	0	0	0
8	0	X	0	X	X	X	0	0	$\boxed{X}$	X
9	0	X	0	0	X	0	$\boxed{X}$	X	0	0

(b)

Fig. 2. A larger traffic matrix (10×10), (a) the reader is directed to find diagonals of length 10 in this matrix (see text for description). (b) One possible diagonal of length 10 for the matrix of part (a).

entries will be chosen and the remaining admissible element(s) will be zero. A diagonal of length 10 through the matrix of Fig. 2(a) is shown in Fig. 2(b) to assure the reader that such a diagonal exists.

III. Extending Submaximal Length Diagonals: A Graphical Approach

A. Introduction

For the following development let the traffic matrix T be a nonnegative integer $N \times N$ matrix whose column and row sums are $\leq C$. As illustrated by the example of Fig. 2(a), a maximum length diagonal is difficult to find in a large sparsely populated matrix. Thus, the approach taken is to start with a submaximal length diagonal and extend it an element at a time until a maximal diagonal is found. The algorithm presented in this section will be developed sequentially, ending in a general approach to finding diagonals in an $N \times N$ matrix. The algorithm is then extended to include matrices in which a diagonal of length N does not exist but which have *critical* rows and columns (columns or rows whose elements sum to C) which must be represented in the diagonal to allow all the traffic to be cleared in C time slots [1], [2]. The modified

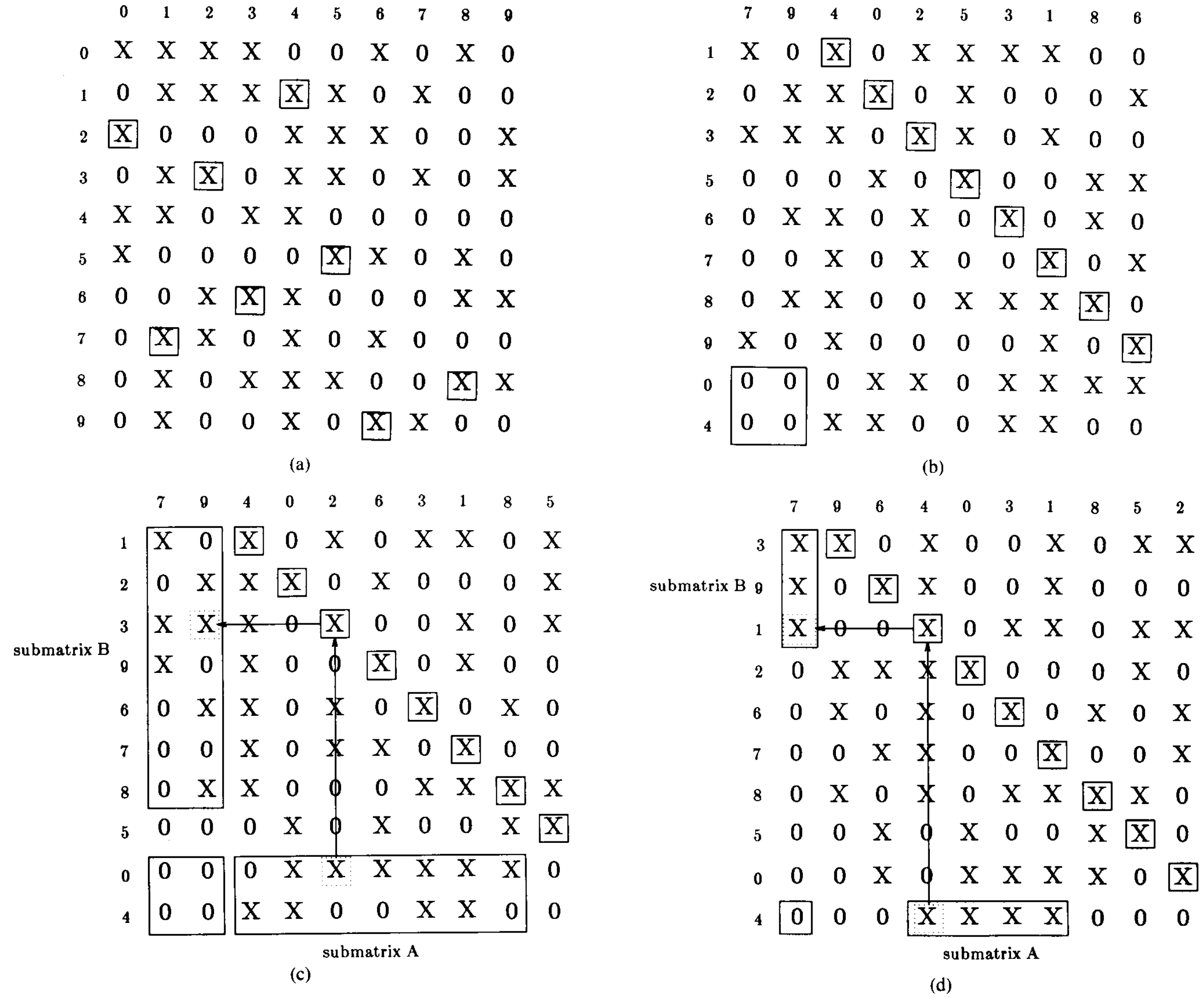


Fig. 3. Illustration of matrix rearrangement and extention of current diagonals using single exchanges. (a) A diagonal of length 8 through a 10 × 10 traffic matrix. This diagonal cannot be extended by choice of a nonzero nexus. (b) A rearrangement of the traffic matrix of part (a) [obtained by row and column exchanges]. See text for description. (c) A rearrangement of the traffic matrix in part (b) to help illustrate a *single exchange*. See text for description. (d) A rearrangement of the traffic matrix of part (c) used to illustrate the final *single exchange* which produces a maximal diagonal.

algorithm produces a diagonal which covers all critical rows and columns.

B. Extension by Single Exchange

Let a row or column be called *uncovered* if it does not currently contain a diagonal element. A *nexus* is defined as a nonzero element which exists at the intersection of an uncovered row and an uncovered column. Diagonal elements may be sequentially chosen from *nexi* until either a maximal diagonal is found or all remaining *nexi* are zero. For example, consider the 10 × 10 traffic matrix of Fig. 3(a) in which X denotes nonzero entries. The boxed elements comprise a diagonal of length 8 which cannot be extended by choosing elements in the remaining uncovered rows and columns. This fact becomes obvious through row and column exchanges[6] which render the original diagonal as a chord spanning from the top row to the right-hand column [Fig. 3(b)]. The leftmost columns (7, 9) and the bottom-most rows (0, 4) are not spanned by the original diagonal and all the entries which do not conflict with this diagonal (lower left-hand submatrix) are zero.

A simple way to extend the current diagonal is by exchanging an element in the current diagonal for two elements in the uncovered rows and columns. To simplify the illustration of this procedure and prepare the way for illustration of later procedures, the matrix of Fig. 3(b) is rearranged to form the matrix of Fig. 3(c). The lower left-hand submatrix is still zero and the diagonal is in the same position. The difference is that submatrix A is the smallest matrix which contains all the nonzero elements of the uncovered rows and likewise, submatrix B contains all the nonzero elements of the uncovered columns (see Appendix for an existence proof of this matrix form). A *single exchange* is accomplished by substituting two elements, one in A and one in B (dotted boxes), for the diagonal element at which they intersect thereby extending the current diagonal by one element. The lines drawn between the three elements in question constitute a *path* between a nonzero element in submatrix A and a nonzero element in submatrix B. After rearranging the resulting diagonal and rendering the resulting matrix in the form of Fig.

[6] These exchanges do not alter the traffic pattern represented by the traffic matrix. They only renumber the inputs and outputs relative to the switch, i.e., the communication request pattern remains the same as viewed by the users.

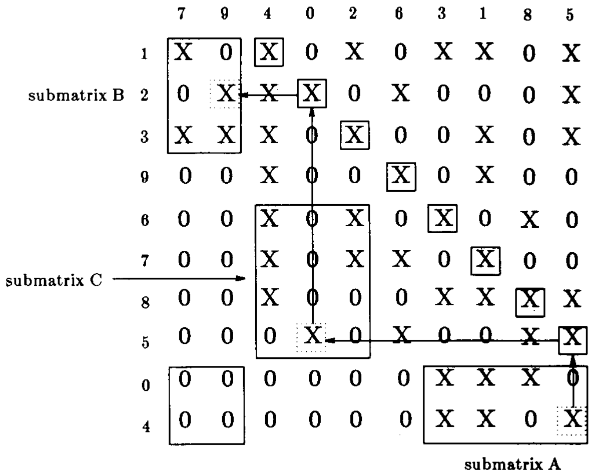


Fig. 4. A traffic matrix in which submatrix A and submatrix B do not intersect along the current diagonal thereby necessitating a double exchange. See text for complete description.

3(c), Fig. 3(d) is obtained. To complete the diagonal, the same procedure as before is used as illustrated [Fig. 3(d)].

C. Extension by Double Exchange

Consider the matrix of Fig. 4. Matrix A and matrix B contain no elements which intersect on the current diagonal.[7] Therefore, no *single exchange* is possible. However, the basic principle may be applied to perform a *double exchange* as follows. The submatrix C is formed by the intersection of the rows whose current diagonal elements intersect A and the columns whose current diagonal elements intersect B. Thus, if the submatrix C is nonzero, then a *double exchange* may always be performed as illustrated in Fig. 4. The three elements in dotted boxes are exchanged for the two current diagonal elements and the diagonal length is increased by one. A *path* is formed from A to B in this manner. The resulting matrix may be rearranged and a *single* or *double exchange* performed to lengthen the diagonal unless C is identically zero.

D. Extension by Multiple Exchanges

In the case where C is identically zero, a *double exchange* is impossible. Such a matrix is illustrated schematically in Fig. 5(a). A notable feature of this matrix is that the embedded submatrix T_1 (dashed outline), which shares no common rows or columns with the elements of A or B, respectively, is exactly of the form of Fig. 3(b); the lower left submatrix is zero and the current diagonal is a chord which spans from the top to the right side. Thus, it will be possible to rearrange T_1 as in Fig. 5(b). Submatrices A_1 and B_1 are the analogs of submatrices A and B in Fig. 3(c). Since such a rearrangement of T_1 need only involve the columns covered by $R1$ and the rows covered by $R2$ in Fig. 5(a), the positions of the current diagonal elements outside T_1 need not be disturbed.

If A_1 and B_1 have an intersection along the current diagonal then a *path* may be formed between A_1 and B_1. If no intersection exists [Fig. 5(c)] then a *path* may still be formed unless C_1 is zero. In either case, since a *path* always exists between A and A_1 and between B and B_1, a *path* will exist between A and B via the *path* between A_1 and B_1. This enables the extension of the current diagonal by a *multiple* exchange. If $C_1 = 0$ then the same pathfinding procedure may

[7] Any matrix for which A and B do not intersect may be rendered as in Fig. 4; i.e., all zeros to the left of A and all zeros below B (see Appendix). This arrangement allows a clearer illustration of extension by *double exchange*.

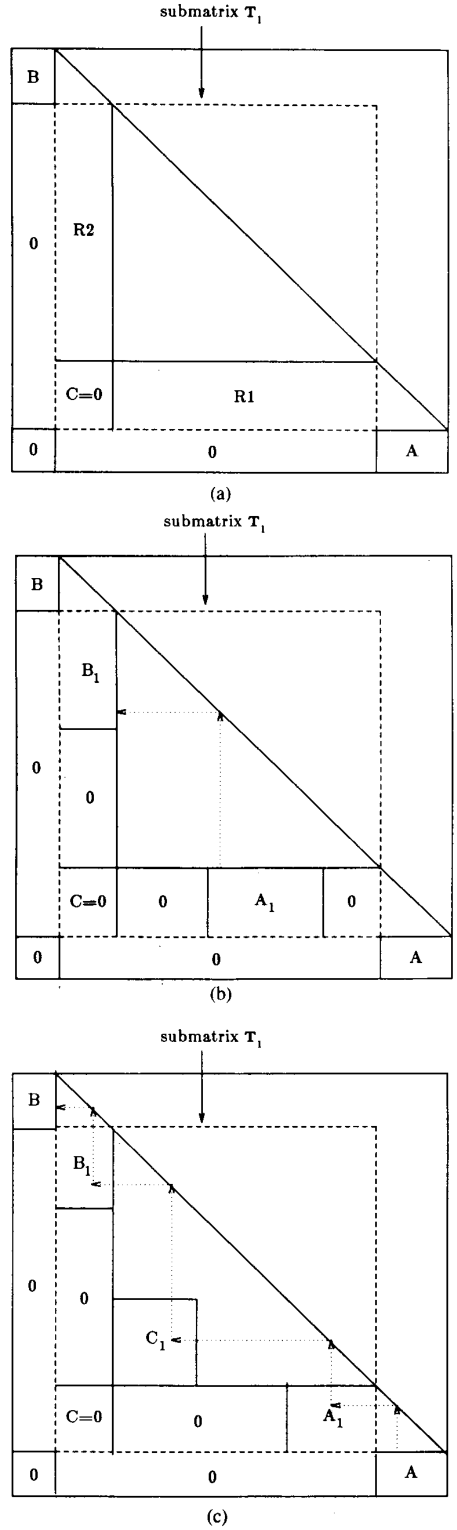


Fig. 5. Matrix schematics used to illustrate extention of diagonals by multiple exchanges. (a) A schematic of a traffic matrix for which submatrix C is identically zero. The submatrix T_1 is outlined by a dashed box. (b) A rearrangement of the embedded matrix T_1 to conform to Fig. 3(c) A *path* between A_1 and B_1 is traced by the dotted line. A schematic depicting a submatrix T_1 in which A_1 and B_1 do not overlap but for which C_1 is not identically zero. A *path* is traced by the dotted line between A and B.

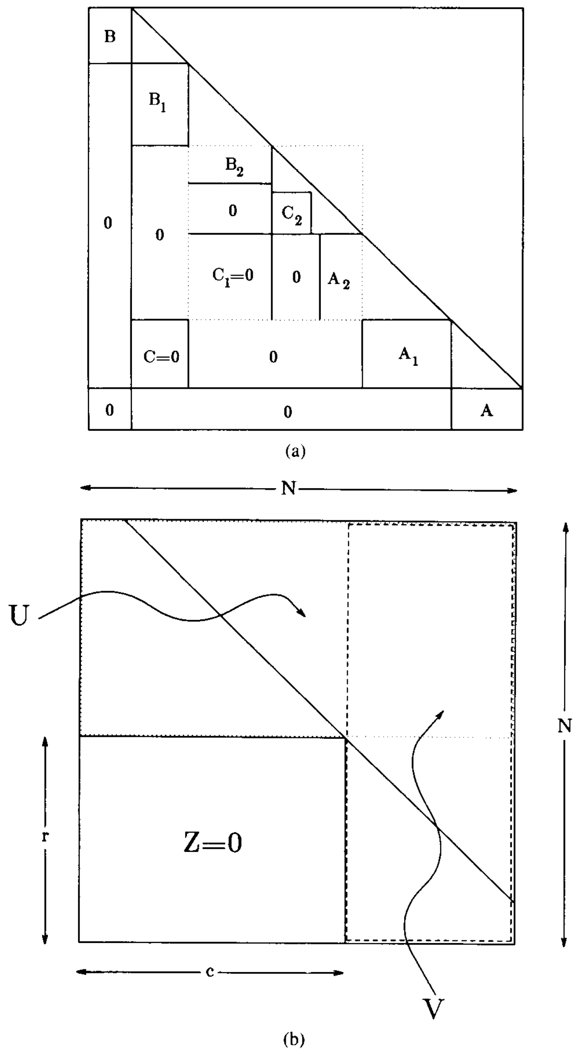


Fig. 6. Schematic matrices used to illustrate the stopping rule for diagonal extension. (a) A rearranged traffic matrix at the $i = 2$ stage (the second embedded submatrix T_2 with dotted outline). This schematic illustrates that by construction, all A_i and B_i border on the current diagonal when A_i and B_i do not overlap on the current diagonal, i.e., no type-3 diagonal elements exist in submatrix T_i (see Appendix: Fig. 15, also Fig. 4). (b) Maximum length diagonals in a matrix containing a zero submatrix of dimension $r \times c$. An $r \times c$ zero matrix Z borders the current diagonal. U spans the $N - r$ rows above Z and V spans the $N - c$ columns to the right of Z. See text for full description.

be performed on the resulting submatrix T_2 contained in submatrix T_1 (just as T_1 was contained in T). The procedure can continue until a path between A_i and B_i in the T_i submatrix is found.

E. The Stopping Rule

Consider the matrix of Fig. 6(a). If A_2 and B_2 do not overlap along the current diagonal (comparable to Fig. 4) and either A_2 or B_2 is identically zero then, schematically, the matrix may be represented by Fig. 6(b). Notice that an identically zero submatrix borders the current diagonal. Extending this concept, if at any stage in the search for a *path* between A and B the submatrix A_i and B_i do not overlap on the current diagonal and either A_i or B_i is zero, then an identically zero submatrix will also border the current diagonal. This fact may be used to define a stopping rule for the algorithm.

Theorem I:

$$A_i = 0 \text{ or } B_i = 0$$

implies the current diagonal cannot be extended.

Proof of Theorem I: Let all the rows above the $r \times c$ zero submatrix Z of Fig. 6(b) be defined as submatrix U (dotted box), and all the columns to the right of Z be defined as V (dashed box). Thus, U is of dimension $(N - r) \times N$ and V is of dimension $N \times (N - c)$. The upper bound on the length of a diagonal in any matrix is the minimum dimension of the matrix. Therefore, the largest diagonal which could be contained in U is of length $N - r$ and likewise the largest diagonal in V is of length $N - c$. Thus, that the largest diagonal which could be contained in $U \cup V$ is of length $l \leq 2N - r - c$.[8] However, since $2N - r - c = d$ is the length of the current diagonal, the current diagonal is the largest possible. The stopping rule is, therefore, very simple. If at some point $A_i = 0$ or $B_i = 0$ then the current diagonal is maximal. Q.E.D.

An important concept that will be used in the parallel implementation of this algorithm is the equivalence of the nonexistence of a *path* between A and B and the existence of an i for which $A_i = 0$ or $B_i = 0$.

Theorem II:

$A_i = 0$ or $B_i = 0$ is equivalent to the nonexistence

of a *path* between submatrices A and B.

Proof of Theorem II: If $A_i = 0$ or $B_i = 0$ then no *path* between A and B can exist since a *path* must include only nonzero elements. Conversely, if an $A \rightarrow B$ *path* exists, then none of the $A_i = 0$ and $B_i = 0$ which contributed elements to the *path* can be zero. Thus, the nonexistence of an $A \rightarrow B$ *path* is exactly equivalent to A_i or B_i being zero for some i. Q.E.D.

Combination of Theorems I and II yields the following equivalence.

- The nonexistence of an $A \rightarrow B$ *path* implies that the current diagonal is maximal.

It is interesting to note that this result defines a new but equivalent condition for the existence/nonexistence of a system of distinct representatives (P. Hall [2]). Specifically, iff at some point during the extension procedure no $A \rightarrow B$ path exists, then no complete system of distinct representatives (diagonal of length N) exists. Thus, it is not too surprising that the previous set-theoretic algorithm by Hall (stated precisely in [4]) which sequentially deletes and adds members to a set of distinct representatives is virtually identical to the method presented here when it is recast in graphical form.

F. Ensuring Coverage of Critical Rows and Columns

A critical row or column of a traffic matrix $\boldsymbol{T}$ is one which sums to C where C is the number of remaining time slots allocated for transmission. If all the packets represented by the traffic matrix are to be sent during these C time slots then these critical rows and columns must be covered by the diagonal. Otherwise, at the next step when only $C - 1$ time slots remain the traffic matrix will violate the *scheduling criterion* in that there will exist a row or column in $\boldsymbol{T}$ whose sum C is greater than $C - 1$.

In $N \times N$ traffic matrices where a diagonal of length N exists, the exchange algorithm will produce a diagonal of length N which covers all the rows and columns. Thus, for such a matrix the problem of critical row and column coverage

[8] It is assumed that $r + c \geq N$ since l cannot exceed the minimum dimension of the matrix.

$$\begin{array}{rccccc} & \text{critical column} & & & & \\ & \downarrow & & & & \\ \text{critical row} \longrightarrow & 1 & 1 & 1 & 0 & 0 \\ & 0 & \boxed{1} & 0 & 0 & 0 \\ & 0 & 0 & \boxed{1} & 0 & 0 \\ & 1 & 0 & 0 & \boxed{1} & 0 \\ & \boxed{1} & 0 & 0 & 0 & 0 \end{array}$$

Fig. 7. A matrix in which a maximal diagonal of length 4 exists but which does not cover all critical rows and columns ($C = 3$).

for diagonals generated by the exchange algorithm does not exist. It is possible, however, for the maximal diagonal to be of fewer elements than the dimension of the traffic matrix. In such a case it is also possible for a maximal diagonal to leave a critical row or column uncovered. A maximal diagonal which does not cover all critical rows and columns is shown in Fig. 7. The exchange algorithm, as currently proposed, could produce such a diagonal. A simple modification, however, will preclude this possibility.

Let the initial diagonal be composed of only elements from critical rows and columns. If any critical rows or columns are left uncovered then they will reside in submatrix A and submatrix B, respectively. In extending the diagonal through *exchanges* consider only elements in the critical rows of A and critical columns of B. Since all the elements involved in each *exchange* will be elements contained in critical rows or columns, the extended diagonal will cover only critical rows and columns until there are no critical columns left in B and no critical rows left in A. Such a diagonal is guaranteed to exist [1]. At this point the diagonal may be extended by the original means without regard for critical rows and columns since all the rows and columns covered by the current diagonal will be covered by any extended diagonal as well. Thus, all the critical rows and columns will remain covered.

IV. Implementation of the Extension-by-Exchange Algorithm

A. Introduction

The previous theoretical development described a method to extend a randomly chosen diagonal, an element at a time, through various exchange procedures. The basis of this extension-by-exchange algorithm was the location of *paths* between elements in uncovered rows and elements in uncovered columns. Three simple observations enable the realization of a simple parallel computer which when given a traffic matrix as input will produce maximal diagonals as output.

• *Observation 1)* Let a *vertex* be defined as an element at which the *path* changes from horizontal to vertical. *Vertices* along a given *path* are invariant under row and column exchanges. For exmaple, the matrix in Fig. 8(a) with a *path* as shown may be transformed by row and column exchanges into the matrix of Fig. 8(b). The *path* elements remain the same as does their connectivity. Thus, the arrangement of current diagonal elements, uncovered rows and uncovered columns in the traffic matrix is irrelevant to the task of finding a *path* between an uncovered row and an uncovered column.

•*Observation 2)* If a *path* is allowed to "wrap around" the edges of the matrix, any *path* may be composed of solely right → left and down → up links [Fig. 8(c)]. The leftward links begin at current diagonal elements and end on nonzero off-diagonal elements. The upward links begin on nonzero off-diagonal elements and end on current diagonal elements. Likewise, a reverse *path* may be composed solely of left → right and up → down links.

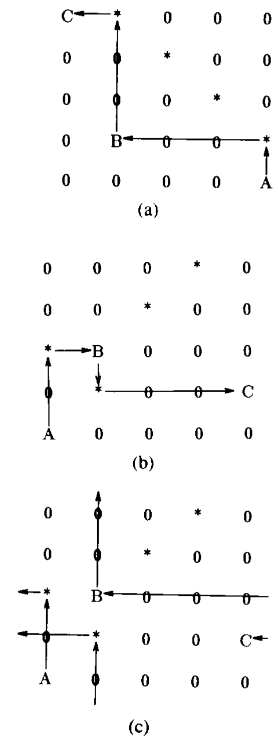


Fig. 8. Illustration of *path* invariance under row and column exchanges, (a) original *path*, (b) *path* of part (a) after row and column exchanges, (c) *path* of parts (a) and (b) executed using only right → left and down → up links. The *paths* extend around the edges of the matrix ("wrap around").

•*Observation 3)* For a matrix in which the current diagonal is of zero length, each nonzero element is a zero length *path*; i.e., each nonzero element is a *nexus* between an uncovered row and an uncovered column.

B. Locating Paths with a Cellular Automaton

Suppose a set of switches were arranged in a matrix with each switch corresponding to an element in a traffic matrix. Connections with neighboring elements would occur on the **r**ight, **l**eft, below (**d**own) and above (**u**p).[9] A structure such as this wherein simple elements communicate directly with only their nearest neighbors is called a *cellular automaton* and is illustrated in Fig. 9. The routing function performed by each switch would depend upon whether it was a current diagonal element, zero or an off-diagonal element. The switches representing current diagonal elements would route their lower input to their left output in consonance with Fig. 8(c). Nonzero off-diagonal elements would route $\mathbf{r} \rightarrow \mathbf{L}$, $\mathbf{U}$, and $\mathbf{d} \rightarrow \mathbf{U}$. Zero elements would always route $\mathbf{r} \rightarrow \mathbf{L}$ and $\mathbf{d} \rightarrow \mathbf{U}$ also as in Fig. 8(c).

A signal injected at the right of an uncovered row would be routed through the network according to the above routing rules. If a *path* existed between that row and an uncovered column, the signal would eventually propagate to the top of an uncovered column as illustrated in Fig. 10. Thus, if signals were injected into the right of every uncovered row, the appearance of a signal at the top of an uncovered column would guarantee the existence of a *path* between an uncovered row and an uncovered column. There are several problems remaining, however; there may be multiple conflicting

[9] To simplify notation let upper case bold letters denote outputs and lower case bold letters denote inputs.

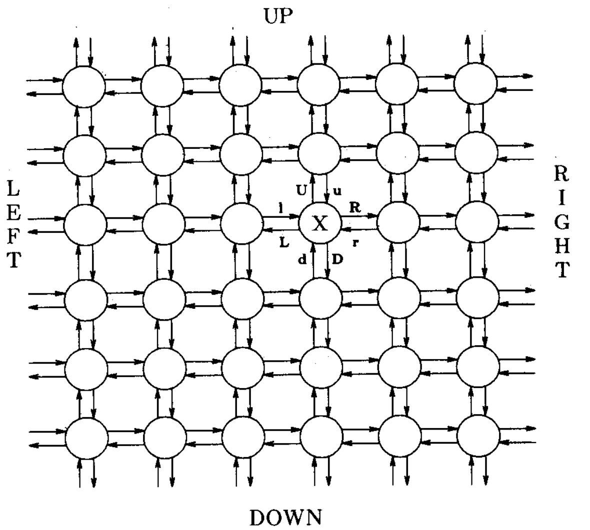


Fig. 9. A cellular automaton composed of a matrix of switches. Each switch is mutually connected to its four nearest neighbors.

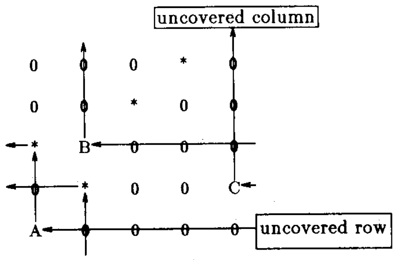


Fig. 10. An illustration of *path* propagation through an uncovered row ending in a *path* through an uncovered column.

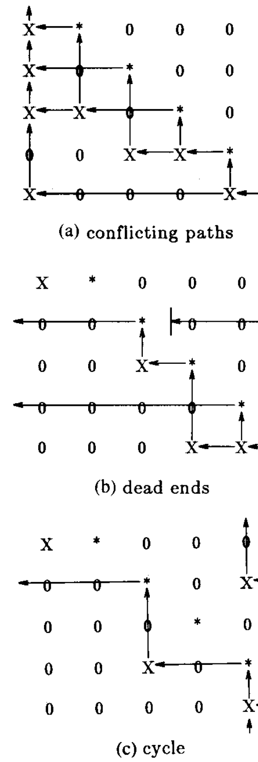


Fig. 11. An illustration of *path* types through the switching mesh. (a) For most matrices, multiple *conflicting paths* exist between an uncovered row and an uncovered column. (b) *Dead ends* terminate on the right side of a current diagonal element. Since current diagonal elements do not accept inputs from the right, the *path* abruptly ends. (c) A *cyclic path* occurs when a *path* overlaps itself.

paths,[10] some *paths* may end in a *cyclic* path which never reaches the uncovered columns or some paths may simply be *dead ends*, again never reaching an uncovered column. *Conflicting paths* are illustrated in Fig. 11(a), two *dead ends* are shown in Fig. 11(b) and a *cyclic path* is shown in Fig. 11(c). Obviously some method of choosing which elements form a valid path is necessary.

Elements comprising *cyclic paths* and *dead ends* may be easily identified by allowing the network to propagate signals in the backward direction via injection of signals downward into uncovered columns. Specifically, let the current diagonal elements also route $l \rightarrow D$, nonzero off-diagonal elements route $u \rightarrow D$, R, and $l \rightarrow R$, and zero elements route $l \rightarrow R$ and $u \rightarrow D$. In that case, only *vertices* of both ascending and descending *paths* would be valid *vertices*. *Vertices of cyclic paths* and *dead ends* will be contained in only an ascending or a descending *path* but not both. The reader may look to Fig. 11(a)–(c) for verification.

The first step in an algorithm which finds valid *paths* is one which removes *dead end* and *cyclic paths*. Thus,

- *Step 1)* Remove from consideration all elements which do not receive both an ascending and descending *path* (*dead end* and *cyclic path* removal). Specifically, impose a $r \rightleftarrows l$ and $d \rightleftarrows u$ routing pattern on all such off-diagonal elements

A remaining problem in the location of valid *paths* is the removal of *conflicting paths*. Since conflict occurs when *paths* share the same row or column, a natural scheme to resolve conflict is to first resolve row conflict and then column conflict. The following three steps illustrate this concept.

[10] *Path conflict* occurs when two or more *paths* travel along the same row or column.

- *Step 2)* After STEP 1 all the remaining off-diagonal *vertices* are members of a valid *path*. Impose a $r \rightarrow U$ and $d \rightarrow U$ routing pattern on these *vertices* thereby allowing only ascending *paths* and one *vertex* per row (one *path* per row).
- *Step 3)* All elements which have ceased being *vertices* should be removed from consideration by imposing the routing pattern of Step 1). Current off-diagonal *vertices* should impose a $u \rightarrow R$ routing pattern thereby allowing only descending *paths* and one *vertex* per column (one path per column).
- *Step 4)* All remaining current diagonal elements which are also *vertices* should become off-diagonal elements and all off-diagonal *vertices* should become new current diagonal elements.

In summary, Step 1) removes from consideration *paths* which end or cycle before spanning the gap between an uncovered row and an uncovered column. Step 2) resolves row conflict, Step 3) resolves column conflict and step 4) allows the exchange of off-diagonal *vertices* for *vertices* on the current diagonal thereby extending the diagonal. This four-step procedure can be applied until a maximal diagonal is found. Since each application guarantees the extension of the diagonal by at least one element (if possible), this procedure need be applied a maximum of N times to find a maximal diagonal. Coverage of critical rows and columns is guaranteed by only injecting signals into critical rows and columns until the current diagonal covers all critical rows and columns. The extension procedure could then proceed as before with signals being injected into all uncovered rows and columns.

Application of the algorithm is illustrated sequentially in Fig. 12(a)–(e). Fig. 12(a) depicts all possible *paths* which

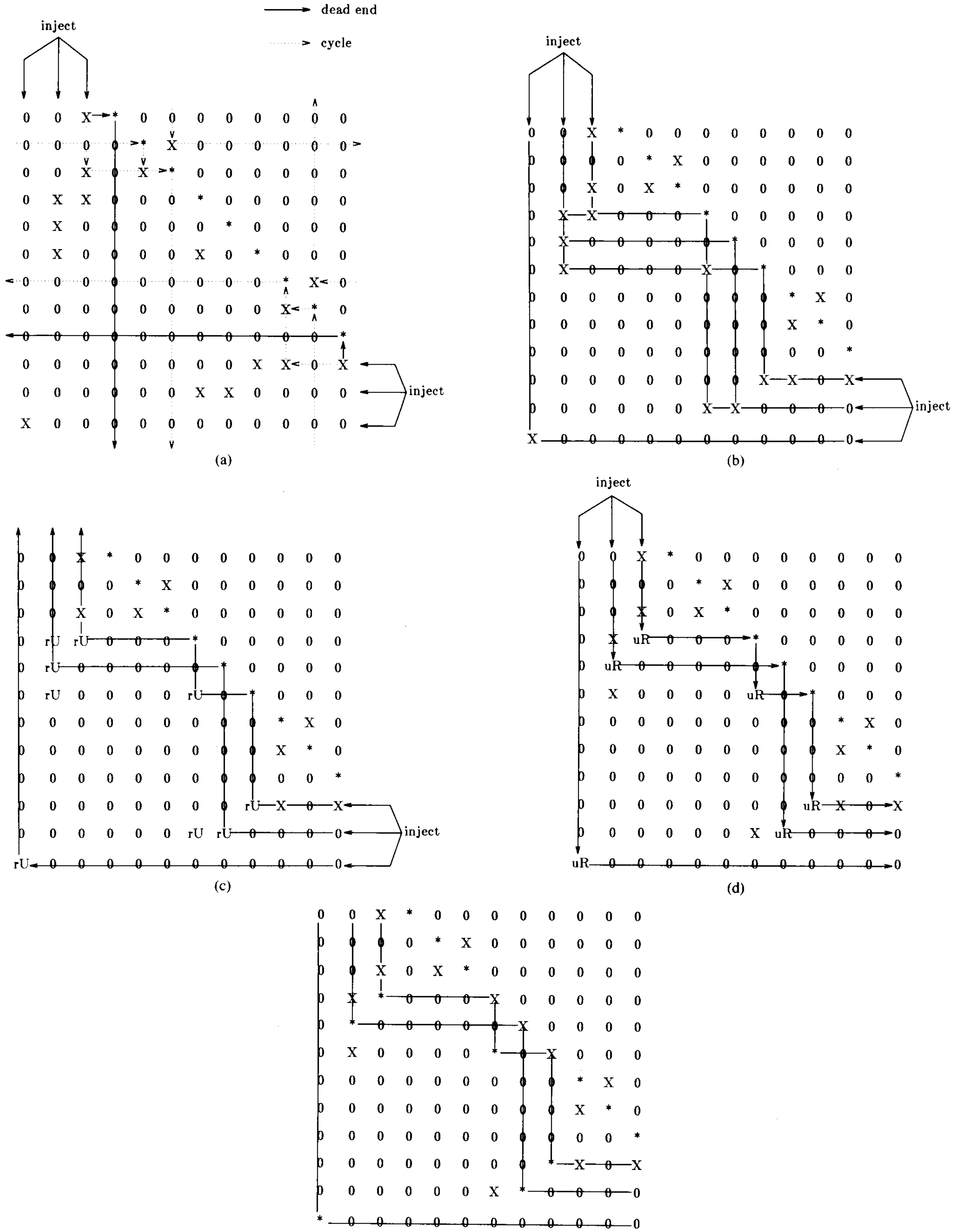


Fig. 12. Application of the extention by exchange algorithm to a 12 × 12 traffic matrix. (a) An illustration of *dead-end* and *cyclic paths*. Notice that each *vertex* (element at which the *path* changes direction) mediates only an ascending or descending *path*. (b) An illustration of *conflicting paths*. Each line in the mesh denotes a bidirectional *path*. (c) The effect of imposing a $r \rightarrow U$, $d \rightarrow U$ routing pattern on the possible *vertices* from Fig. 11(b) and the elimination of row conflict. (d) The effect of imposing a $u \rightarrow R$ switching pattern on the off-diagonal *vertices* to remove column conflict and the resulting nonconflicting *paths*. (e) Exchange of the surviving off-diagonal *vertices* of the current diagonal elements for the *paths* of Fig. 11(d) and the resultant extention of the diagonal (in this case by three elements).

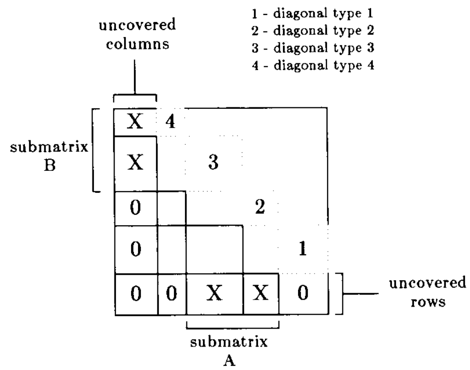


Fig. 13. Grouping of four different types of current diagonal elements into four subblocks. See Appendix for description.

contain elements that are *vertices* for only an ascending or descending *path* (*dead ends* and *cyclic paths*). Fig. 12(b) depicts several possible valid but *conflicting paths*. Fig. 12(c) shows the effect of the routing pattern $\boldsymbol{r} \rightarrow \boldsymbol{U}$, $\boldsymbol{d} \rightarrow \boldsymbol{U}$ on the remaining possible off-diagonal *vertices* and the resolution of row conflict. Fig. 12(d) shows the effect of a $\boldsymbol{u} \rightarrow \boldsymbol{R}$ routing pattern and the resolution of column conflict. Finally, Fig. 12(e) shows the extended diagonal after exchange of current diagonal *vertices* with off-diagonal *vertices*.

V. Discussion and Conclusion

The extension-by-exchange algorithm offers an intuitive and graphical approach to the problem of finding maximal diagonals in an arbitrary matrix. A set-theoretic approach to this problem has previously been presented by Hall (precisely stated in [4]). It is interesting to note that if Hall's set-theoretic algorithm is recast in matrix form, and we observe how elements are added to and deleted from the set of distinct representatives, then the procedure bears a striking resemblance to the extension by exchange algorithm. It is this graphical approach to the problem which offers the conceptual framework that leads directly to a special purpose computer capable of providing diagonals rapidly.

The parallel implementation of the extension by exchange algorithm allows the rapid calculation of optimal schedules by using many simple processing elements (cells for short). The speed of computation may be found by first assuming that the total time taken for signal propagation through the network is negligible (<1 ns). Such an approximation is valid if all the switches are located close to one another. If logic devices, operating in the 20–40 ns range are assumed, then each switch will be able to perform the necessary calculations[11] well within 50 ns. Thus, assume each STEP of the procedure outlined in Section IV-B requires 50 ns. 200 ns would be necessary to produce an extension of the current diagonal. Therefore, for a switch of size N, at most $N \times 200$ ns would be required to produce a maximal diagonal. It is noteworthy that scheduling time is virtually a linear function of N rather than quadratic as would be the case for a sequential computer. Let each switch be capable of storing the value of its associated t_{ij} element of the traffic matrix and when appropriate be able to decrement this value at the end of eac 200 ns cycle [Step 4]. A complete schedule could then be generated sequentially in $CN \times 200$ ns. For $C = 100$ and $N = 10^3$, the scheduling time would be 20 ms. If the decision time for each switch of 50 ns could be lowered to <10 ns using higher speed logic then the scheduling time would be under 4 ms. In either case, such speed is impossible even using current state-of-the-art sequential computers. Of course, an $N = 10^3$ parallel scheduler of the type described would require 10^6 cells and may therefore be impractical. However, for smaller N (say $N = 100$) scheduling machine should be possible using current VLSI technology.

In summary, the graphical and intuitive extension-by-exchange algorithm provides the basis for a high-speed time-division multiplex switch scheduler. Approximate scheduling times for a $N = 1000$ input, $C = 100$ time slot switch should be in the range of milliseconds as compared to seconds or minutes using current computing technology. In addition, owing to the parallel structure of the scheduling network and the assumption that signal propagation within the network is rapid compared to the response times of the individual cells, scheduling time varies only linearly with N rather than as the square of N. Thus, the benefits of optimal scheduling could be theoretically extended to a switch of virtually any size thereby achieving performance improvements of 10-20 percent in some networks [3].

[11] Each "switch" representing a nonzero element in the traffic matrix must be able to decide whether it should change from/to a diagonal element and whether it is a valid *vertex*.

Appendix

It is always possible to render a matrix such as that in Fig. 3(a) in the form of that in Fig. 3(c). Specifically, a submatrix A with no zero columns, a submatrix B with no zero rows and the current diagonal as shown in Fig. 3(c) may be formed by column and row exchanges. This may be shown as follows.

Consider that each column and each row of the traffic matrix $\boldsymbol{T}$ will have either zero or nonzero intersections with the uncovered rows and columns, respectively. Thus, there are four types of current diagonal elements, d_{ij} where i is the row of the diagonal and j is its column.

1) The intersection of row i and column j with the uncovered columns and rows yields only zero elements.

2) The intersection of row i and the uncovered columns is zero while the intersection of column j and the uncovered rows is nonzero.

3) The intersection of row i and column j with the uncovered columns and rows is nonzero.

4) The intersection of row i and the uncovered columns is nonzero while the intersection of column j and the uncovered rows is zero.

By simple row and column exchanges each of these diagonal types may be grouped into four homogeneous subblocks as shown in Fig. 13. This grouping yields a submatrix A with no zero columns and a submatrix B with no zero rows. The remaining rearrangement (to render the current diagonal as a chord spanning from the upper edge to the right edge of the matrix) may be accomplished by arranging the diagonal

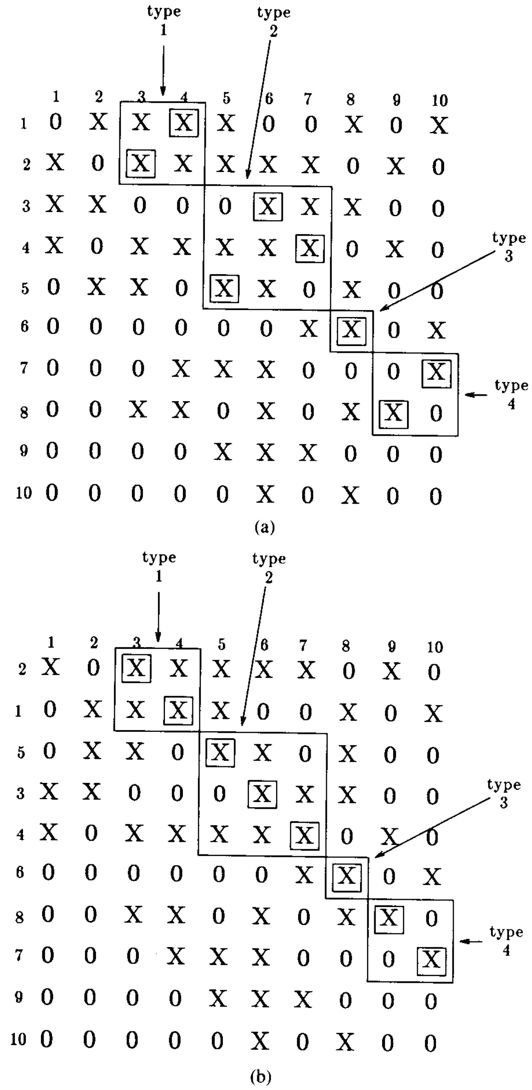


Fig. 14. A specific example of a matrix in the form of Fig. 13 rearranged to form a diagonal which spans from the top to the right-hand side of the matrix. (a) Diagonal elements grouped by type (see Appendix and Fig. 13); (b) matrix of part (a) with diagonal elements rearranged to conform with that of Fig. 3(c). Notice that the rearrangement did not require any column exchanges and that all diagonal "types" remained in their original subgroup.

elements within each subblock along the main diagonal of the given subblock. Since the rows (and columns) associated with each subblock do not intersect those of any other subblock, this rearrangement may be done independently for each subblock by exchanging the rows (or columns) appropriately. The resulting matrix will be identical in form to that in Fig. 3(c). A simple example is provided in Fig. 14.

It is also readily seen that if no type-3 diagonal elements exist (submarix A and B do not overlap on the current diagonal), then the matrix can be put in the form of that in Fig. 4. First the rows associated with type-2 diagonal elements must be extracted and appended to the bottom of the matrix. Then the columns associated with these elements must be extracted and appended to the right side of the matrix. The resulting matrix as shown in Fig. 15 is identical in form to that of Fig. 4.

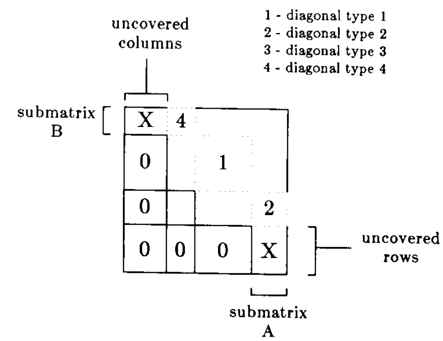


Fig. 15. Matrix form resulting when no type-3 diagonal elements exist. See Appendix for complete description.

Acknowledgment

I would like to thank A. S. Acampora, K. Y. Eng, H. V. Jagadish, and B. Wittner for helpful discussions of this work.

References

[1] A. S. Acampora and B. R. Davis, "Efficient utilization of satellite transponders via time-division multibeam scanning," *Bell Syst. Tech. J.*, vol. 57, no. 8, pp. 2901-2914, 1978.

[2] M. Hall, Jr., *Combinatorial Theory, Ed. II.* New York: Wiley, 1986, ch. 5, Distinct Representatives.

[3] C. Rose and M. G. Hluchyj, "The performance of random and optimal scheduling in a time-multiplex switch," *IEEE Trans. Commun.*, vol. COM-35, pp. 813-817, 1987.

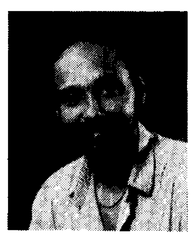
[4] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. 27, pp. 1449-1455, 1979.

[5] G. Bongiovanni, D. Coppersmith, and C. K. Wong, "An optimum time slot assignment algorithm for an SS/TDMA system with variable number of transponders," *IEEE Trans. Commun.*, vol. 29, pp. 721-726, 1981.

[6] I. Gopal, D. Coppersmith, and C. K. Wong, "Minimizing packet waiting time in a multibeam satellite system," *IEEE Trans. Commun.*, vol. COM-30, pp. 305-316, 1982.

[7] R. Ramaswamy and P. Dhar, "Comments on an efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. COM-32, pp. 1061-1065, 1984.

★



Christopher Rose was born on January 9, 1957, in New York City, NY. He received the B.S. (1979), M.S. (1981), and Ph.D. (1985) degrees all from the Massachusetts Institute of Technology in Cambridge, MA.

He is currently with AT&T Bell Laboratories in Holmdel, NJ, as a member of the Network Systems Research Department. His current technical interests include adaptive connectionist (neural) networks, cellular automata, communications systems and, most recently, applications of high-temperature superconductivity to communications problems.