ANALYSIS OF ERROR DETECTION PERFORMANCE OF A CODE WITH DECIMAL CHECK DIGIT*

by

Dr. Efren F. Abaya**

INTRODUCTION

Numeric or alphanumeric codes that are manually transcribed for off-line input to a computer are commonly protected against misreading or mistyping by a "check symbol" computed using decimal or integer arithmetic operations on the characters of the code. Examples of such codes include identification numbers, serial numbers, account codes, and mnemonic codes where the check digit provides protection at the point of data entry against the more common manual typing errors such as substitution, transposition, dropping or insertion of characters.

This paper discusses the use of decimal check symbols for detection of errors in numeric codes. The analytic methods used here and the types of errors encountered are somewhat different from those used for the binary error detection codes that are more common in the literature.

The paper first reviews the mathematical basis of codes using decimal arithmetic and the strengths and weaknesses of this class of codes as far as detecting errors is concerned.

The paper next describes an analytic method for determining the fraction of undetectable error patterns for various types of errors. The method is applied to a specific type of numeric account code described in Section 3.0 that uses alternate doubling of digits and a modulus of 10. For this code, it is shown that roughly 10% of all errors will not be detected, although for certain error types the percentage is even higher. The method of analysis described here appears to be applicable also to other kinds of check digit schemes.

Finally, the paper touches briefly on performance simulation as an alternative to mathematical analysis for evaluating detection performance of these codes.

---

* Paper presented at the PCS softrain '89 in June 1- 30, 1989

** Consultant, Computer Information Systems, Inc. and Associate Professor, University of the Philippines

DECIMAL CODES

In this paper, "decimal codes" will be used to refer to those codes which compute a check symbol (or check digit) using decimal arithmetic operations on integer values assigned to the various data characters in a codeword.

A codeword of length L characters will be denoted by

$$d_1\ d_2\ d_3\ d_4 \ldots d_{L-1}\ d_L$$

where d_1 to d_{L-1} are the data characters and d_L is the check symbol. (Since this paper focuses on numeric codes, the d_i are also called "digits".)

Distinct numerical values are assigned to the different characters in the codeword. For purely numeric codes, the digits "0" to "9" carry their respective numerical values. For alphanumeric codes, a common assignment for the letter symbols is "A" = 10, "B" = 11, "C" = 12, ... "Z" = 35 and "space" = 36.

Weights are assigned to each character position (usually the check symbol position carries a weight of unity) in order to compute a weighted sum defined by

$$WS = \sum_{i=1}^{L} w_i\ V(d_i)$$

where :

w_i = weight of the i-th character position

$V(d_i)$ = numerical value of character d_i

The weighted sum is reduced to a value corresponding to one of the code characters by the process of taking the remainder (or "residue") when WS is divided by a fixed integer M known as the "modulus" of the code. Mathematically, this remainder is defined by

$$RWS = \sum_{i=1}^{L} w_i\ V(d_i) \qquad (\text{mod } M) \qquad \text{(Eq. 1)}$$

Thus, $0 \leq RWS < M$.

For purposes of error detection, the check symbol is chosen so that each validly encoded codeword satisfies

$$RWS = 0 \qquad (\text{mod } M) \qquad \text{(Eq. 2)}$$

Violations of the above condition indicate detected errors in the combination being checked. An example of the computation involved in Eq. (1) is given in Section 3.0

Generally, a modulus that is a prime number such as 7, 11, 13, 37, etc. gives the best performance in detecting errors. However, the modulus 10 is also used and will be discussed in Section 3.0.

The following are some examples of decimal codes:

1. Symbols "0" to "9" and "A" to "Z", plus "space", carrying numeric values 0 to 36; progressive weighting of 1-2-3-4-etc.; modulus 37 [1];
2. Digits 0 to 9 (plus X = 10 for the check digit positions); progressive weighting of 1-2-3-4-etc.; modulus of 11 (used in International Standard Book Numbers) [1];
3. Either of the above two symbol sets with geometric weighting of 2-4-8-16-32-etc.;
4. The code presented in Section 3.0.

Decimal codes detect errors commonly encountered in typing such as alteration or interchange of digits by the fact that such changes usually lead to a different weighted sum that violates Eq. (2). This detection is performed without reference to any other data (not even the original digit combination) and so the code may be described as "self-checking". Other computationally simpler methods such as range checks and mnemonic combinations [4,5] can also be used to reduce the incidence of errors and to detect their presence, but the class of codes described above generally provides more power for detecting errors.

The detection performance of a code can be quantified by the fraction of undetectable errors defined as

$$\text{fraction not detected} = \frac{\text{\# error patterns not detected}}{\text{total \# error patterns}}$$

For example, if a fraction E of codewords manually encoded contain some errors, and the fraction not detected is F, then approximately E x F of encoded records will get through with errors in spite of the check digits. The detection percentage can also be computed for specific types of errors, e.g., single-digit, double-digit, random errors, transposition, etc.

If a code employs r independent check digits then approximately,

$$\text{fraction not detected} = 1/(\text{modulus})^r$$

The following things may be said in general about decimal codes that use one to check digit:

1. A single digit error will always be detected because of the resulting change in $w_i V(d)$ for the affected digit position;

2. Transposition of two digits in positions having <u>different</u> weights will usually be detected because changes in the $w_i V(d)$ do not cancel out, while transposition of digits in positions having <u>equal</u> weights will <u>never</u> be detected.

When the modulus M is a power of a prime number, well developed mathematical theories of Galois fields, integer rings and number theory can be applied for the design and analysis of decimal codes by treating them as cases of m-ary algebraic block codes [1,2]. However, these theories usually consider random errors or burst errors which are of a different nature than those occurring in manual transcription. Moreover, some codes such as the one to be discussed later employ a composite modulus and hence are not amenable to these algebraic models.

An alternative way to view the structure of a decimal code is that the mod M multiplication $w_i V(d_i)$ imposes a permutation or "scrambling" on the digits, as shown in Table I for the example of a mod 11 code (#2 above). From number theory, it is known that the mod M multiples of any number X are all different (except for cyclic repetition) if the numbers X and M have no common divisor except unity. (Mathematically, the two numbers are said to be relatively prime.) This situation is easiest to secure if M is prime, although this condition is not necessary.

Define the scrambled value taken by digit d in the i-th position as $V^*_i(d)$, which depends on the weight w_i and modulus M. That is

TABLE 1

VALUES OF V_i^* (d) FOR MOD 11 ALPHANUMERIC CODE

DATA DIGIT (d)	WEIGHT. MULTIPLIER (w_i)										
	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	X
2	0	2	4	6	8	X	1	3	5	7	9
3	0	3	6	9	1	4	7	X	2	5	8
4	0	4	8	1	5	9	2	6	X	3	7
5	0	5	X	4	9	3	8	2	7	1	6
6	0	6	1	7	2	8	3	9	4	X	5
7	0	7	3	X	6	2	9	5	1	8	4
8	0	8	5	2	X	7	4	1	9	6	3
9	0	9	7	5	3	1	X	8	6	4	2

NOTE: X = 10

$$V_i^{*}(d) = w_i \, V(d_i) \pmod{M}$$

Then the condition for acceptance of a codeword given in Equation (2) can be rewritten as

$$\sum_{i=1}^{L} V_i^{*}(d_i) = 0 \pmod{M} \quad \text{(Eq. 3)}$$

For example, if a mod 11 code has weight $w_8 = 5$ in position 8, then $V^*_8(d)$ is given by the fifth column of Table I. An example of the calculation in Eq. (3) will be given in Section 3.0.

A MODULUS 10 CODE

The rest of this paper will concentrate on a modulus 10 numeric code with alternate 1-2 weighting defined as follows:

1. The code is represented as
   $d_1 \; d_2 \; d_3 \; d_4 \ldots \; d_{L-1} \; d_L$
   where d_L is the check digit.

2. The code symbols are the digits 0 to 9 with their respective numeric values (i.e., $V(d_i) = d_i$).

3. The check symbol position and every other alternate position are assigned a weight of unity, while in-between positions are assigned a weight of 2.

4. The modulus M equals 10.

5. In computing the weighted sum WS in Eq. (1), if the product of a digit value and its assigned weight exceeds 9, then the sum of the units and tens digits of the product is taken instead.

EXAMPLE:

This example illustrates the computation of the weighted sum and the validation of a codeword.

codeword = 1 2 3 4 5 6 7 8 9 0 3

POSITION NUMBER, i	WEIGHT w_i	DIGIT d_i	w_i V (d_i)
1	1	1	1
2	2	2	4
3	1	3	3
4	2	4	8
5	1	5	5
6	2	6	1 + 2 = 3
7	1	7	7
8	2	8	1 + 6 = 7
9	1	9	9
10	2	0	0
11	1	3	3
		WS =	50

Since WS = 0 (mod 10) the codeword is acceptable.

Mathematically, the effect of rules (3) and (5) is to "scramble" the digits according to Table II. With these scrambled values, the RWS can be computed in an alternative way following Eq. (3) as shown by the example below.

EXAMPLE:

This example illustrates the computation of the weighted sum using scrambled values from Table II.

codeword = 1 2 3 4 5 6 7 8 9 0 3

POSITION NUMBER, i	WEIGHT w_i	DIGIT d_i	V_i^* (d_i) odd	even
1	1	1	1	
2	2	2		4
3	1	3	3	
4	2	4		8
5	1	5	5	
6	2	6		3
7	1	7	7	
8	2	8		7
9	1	9	9	
10	2	0		0
11	1	3	3	
		WS =	28 +	22 = 50

Since WS = 0 (mod 10) the codeword is acceptable.

This last method of describing the code in terms of the scrambling function $V^*_i(d)$ is convenient for the analysis of error performance that will follow.

ANALYSIS OF ERROR PERFORMANCE

The errors analyzed in this paper fall into the following (non-exclusive) types taken from [3]:

1. t-DIGIT RANDOM ERROR: Simultaneous errors in exactly t random positions of a codeword.
2. TRANSPOSITION ERROR: The digits in two positions (not necessarily adjacent) are switched.
3. DELETE ERROR: An omission of one digit of the codeword, causing the digits on the right side of the omitted digit to shift one position to the left, e.g.:

TABLE II

VALUES OF $V^*_i(d)$ FOR MODULUS 10 NUMERIC CODE

DIGIT	SCRAMBLED VALUES $V^*_i(d)$ POSITION OCCUPIED Weight 1 Position	Weight 2 Position	Difference* (mod 10)
0	0	0	0
1	1	2	9
2	2	4	8
3	3	6	7
4	4	8	6
5	5	1	4
6	6	3	3
7	7	5	2
8	8	7	1
9	9	9	0

*Mod 10 difference of scrambled values (column 2 minus column 3).

Refer to Appendix B.

codeword : d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11

error : d1 d2 d3 d5 d6 d7 d8 d9 d10 d11 X

where X is usually zero.

4. TRANSFER ERROR: A digit is moved to another position, causing a range of digits to shift one position right or left. Alternatively, this error can be characterized as a circular shift of a range of t digits, e. g. :

codeword: d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11

error (t = 4): d1 d2 d6 d3 d4 d5 d7 d8 d9 d10 d11

5. INSERT ERROR: An additional digit is inserted, causing the digits to the right of the insertion point to move one position to the right, e.g.

codeword : d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11

error: d1 d2 d3 X d4 d5 d6 d7 d8 d9 d10

where X is the additional digit, the old check symbol d11' is lost and d10 becomes the new check symbol.

ERROR PERFORMANCE FOR t-DIGIT RANDOM ERRORS

This section calculates the number of undetectable error patterns involving t digits for a mod 10 code of length L digits.

The analyses in this and the following Sections assumes that every pattern of digits satisfying Eq. (3) is allowed as a valid codeword. In many applications, there may be additional range constraints (e.g., on a date) and other structural limitations on the formation of valid codewords. Such additional constraints usually reduce the number of undetectable errors below the levels calculated here.

In general, the number of undetectable error patterns affecting exactly t digits in random positions is equal to the product of two factors:

-- the number of ways to select a combination of t positions from among L, designated by the combinatorial formula C(L, t); and

-- the number of ways that digits in t specific positions can be perturbed without affecting the equality of Eq. (3), designated Nt.

Therefore, the number of undetectable t-digit error patterns is given by

$$C(L, t) \times Nt \qquad \text{(Eq. 4)}$$

In Appendix A, it is proven that Nt is given by an expression of the form

$$Nt = (9^t + 9)/10 \quad \text{for } t \text{ even}$$
$$= (9^t - 9)/10 \quad \text{for } t \text{ odd} \qquad \text{(Eq. A5)}$$

TABLE III

ANALYSIS OF t-DIGIT RANDOM ERRORS NOT DETECTED BY MODULUS 10 CODE

NUMBER OF ERROR DIGITS	Multiplier for Number of ERRORS	UNDETECTABLE ERRORS	PERCENT OF ERRORS NOT DETECTED
t	9^t	N_t	(%)
1	9	0	0
2	81	9	11.1
3	729	72	9.9
4	6,561	657	10.0
5	59,049	5,904	10.0
6	531,441	53,145	10.0
7	4,782,969	478,296	10.0
8	43,046,721	4,304,673	10.0

The total number of error patterns affecting exactly t digits (i.e, exactly t incorrect keystrokes) is given by $C(L, t) \times 9^t$ because there are 9 incorrect digits that may be substituted for the correct digit in any position. The second factor of this product is tabulated in Table III.

Taking the ratio of the number of undetectable error patterns in Eq. (4) to the number of error patterns affecting exactly t digits yields $Nt/9^t$ which is the fraction that passes undetected by the mod 10 check when the encoder mistypes t digits. It may be seen from Table III that the ratio is practically equal to 10% for different numbers of error digits t greater than one.

The overall average fraction of t-digit errors that pass undetected is 10%.

ERROR PERFORMANCE FOR OTHER TYPES OF ERRORS

This section calculates the fraction of undetectable error patterns for error types #2 to #5 described in Section 4.0.

TRANSPOSITION ERROR. Transposition errors may be classified into two cases:

-- Digits in two even-numbered positions may be transposed without changing the weighted sum. The same is true for two odd-numbered positions.

-- A "0" and a "9" occupying any arbitrary positions may be transposed without detection.

This gives the following expression for the fraction of transposition errors not detected by the code:

$$\frac{C(q, 2) + C(L-q, 2) + 2q(L-q)/90}{C(L, 2)} \qquad \text{(Eq. 5)}$$

where

q = number of even-numbered digit positions

= largest integer not exceeding L/2.

The divisor 90 in the third term of the numerator is due to exclusion of codewords in which the digits in the positions to be exchanged are identical.

The expression above yields the same value for L = 2m and L = 2m-1 as shown in Table IV.

Note that the majority of undetectable transpositions involve non-adjacent positions. It may be seen in Table IV that for codes of length greater than 2 digits, a significant fraction of 2-digit transposition errors are not detected.

DELETE ERROR. A delete error may be considered to be the combination of the dropping of one digit and the subsequent left shift of the digits to the right. The effect of the left shift on the weighted sum is equivalent to the pairwise digit exchange discussed in Appendix B which increases the weighted sum by any number between 0 and 9. This change may be cancelled if the value of the digit dropped is the modulus 10 complement of the amount added by the left shift.

As long as the dropped digit is not in the leftmost position of the codeword (d_i), the constraint Eq. (3) does not apply to the sub-string of error digits, so that the range of positions affected by a delete error can begin with any digit from 0 to 9. Therefore, 1/10 of valid codewords are subject to undetectable delete errors.

TRANSFER ERROR. A transfer error changes digit positions from odd-numbered to even-numbered or vice-versa. In its effect on the reduced weighted sum in Eq. (2) this is equivalent to transposing several contiguous pairs of adjacent digits for which it is shown in Appendix B that the number of combinations that preserve the weighted sum is given by

TABLE IV

TRANSPOSITION ERRORS IN MOD 10 CODE

LENGTH, L	PERCENT OF ERRORS NOT DETECTED
2	2.22
3, 4	34.81
5, 6	41.33
7, 8	44. 13
9, 10	45.68
11, 12	46.67
13, 14	47.35
15, 16	47.85
17, 18	48.24
19, 20	48.54

TABLE V

TRANSFER ERRORS IN MOD 10 CODE

LENGTH OF ERROR PATTERN t	A_t	PERCENT OF ERRORS NOT DETECTED
2	12	2.22
3		11.11
4	1,008	9.99
5		10.07
6	100,032	10.00
7		10.00
8	10,000,128	10.00
9		10.00

the number A_t in Eq. (B7). For simplicity, we limit discussion to transfer errors of length t less than L.

Case: t Even. When t is even, there are A_t - 10 undetectable transfer errors (disregarding the starting position of the error string in the codeword). since the number At

derived in Appendix B includes 10 permutations in which all t digits are identical.

The total number of t-digit transfer patterns is $10^t - 10$. Hence the fraction not detected for a length t transfer error is $(A_t - 10)/(10^t - 10)$.

Case: t Odd. When t is odd, the digit that is transferred keeps the same weight. Hence, this is the same as a pairwise digit exchange of t-1 digits. Reasoning similarly as the previous case, the fraction of errors not detected is $(10\ A_{t-1} - 10)/(10^t - 10)$.

As shown in Table V the fraction of undetectable transfer errors is practically 10% for strings of at least 3 digits.

INSERT ERROR. Consider a digit inserted as in the example given previously in Section 4.0. Out of ten possible digits to insert, exactly one will retain the reduced weighted sum in Eq. (2) divisible by 10. Ignoring cases in which the digit inserted and the digits affected are all identical (which would not constitute an error), the fraction of insert errors not detected is very close to 0.10.

COMPUTER SIMULATION

An alternative method for investigating the performance of a code is by computer simulation. This section compares the results reported above with those obtained by some computer simulation studies on the code discussed in Section 3.0.

For purposes of simulation, the mod 10 code was taken to have a length L = 11 digits consisting of 10 data digits plus one check digit. Odd-numbered positions have a weight $w_i = 1$ while even-numbered positions have a weight of 2. Check digit verification follows the example given in Section 3.0.

Simulation of Random Errors. As discussed in Eq. (A3) of Appendix A, an error is associated with an "error pattern" $E_1E_2E_3...E_{11}$ where $E_i = 0$ if no error occurred in the i-th digit and $E_i = V^*_i(d_i') - V^*_i(d_i)$ is some number between 1 to 9 if digit d_i is erroneously converted to d_i'. An error is not detected if Eq. (A3) is satisfied.

That is

$$\sum_{i=1}^{t} E_i = 0 \pmod{10} \qquad \text{Eq. A3}$$

To simulate random errors in a way that is independent of the actual codeword, random 11 digit strings were generated using the built-in "random" function generator of the Turbo-Pascal programming language. For each string thus generated, Eq. (A3) was checked to

TABLE VI

COMPARISON OF MATHEMATICAL ANALYSIS AND
COMPUTER SIMULATION OF MOD 10 CODE PERFORMANCE

Type of Error	PERCENT OF ERRORS NOT DETECTED	
	Analytic Value	Simulation Value from [3]
1. t-Digit Random Errors (over all t)	10.0%	9.8%
2. Transposition Errors	46.7%	44.4%
3. Delete Errors	10.0%	0. %
4. Transfer Errors	10.0% (t > 4)	5.2% (over all t)
5. Insert Errors	10.0%	9.9%

determine what fraction would not have been detected. The result for 14,000 simulated errors shown in Table VI is quite close to the theoretical value.

Simulation of Other Error Types. Simulation of the other types of errors (#2 - #5) listed in Section 4.0 is different from the method described above because the errors depend on the actual digit sequence in the codeword. To simulate such errors, random codewords are generated and exhaustively searched for all undetectable errors of a specific type. One such study reported in [3] used six randomly generated codewords with results shown in Table VI.

For most error types, the results of simulation are in good agreement with the analytic values with the exception of delete and transfer errors which were underestimated by the simulation probably due to the limited number of codewords tested.

CONCLUSION

This paper has discussed an analytic method for determining the fraction of undetectable errors in a mod 10 numeric code. It was shown by mathematical analysis and verified by computer simulation that roughly 10% of errors of different types are not detected by the mod 10 code, although for certain errors types the percentage is even higher.

ACKNOWLEDGEMENT

The author wishes to thank Victor B. Gruet for providing inspiration and encouragement and Computer Information Systems, Inc. for supporting this work.

REFERENCES

RICHARD W. HAMMING, " Coding and Information Theory", New York, Prentice Hall, 1980, pp. 28-34.

T.R.N. RAO and E. FUJIWARA, "Error-Control Coding for Computer Systems", New Jersey, Prentice-Hall International, Inc., 1989.

MARI JO RUIZ, "A Comparative Study of Check Digit Schemes," Internal Report, Computer Information Systems, Inc.

K.B.C. SAXENA, "Enhancing System Reliability Through Humanized Codes", Proceeding 1st South East Asia Regional Computer Conference (SEARCC 76), Conference Ed., Amsterdam, North-Holland Publishing Co., 1976, pp. 543-547

L. SONNTAG, "Designing Human-Oriented Codes, " Bell Laboratories Record, Vol. 49 No. 2 (Feb. 1971), pp. 43- 49.

APPENDIX A

DERIVATION OF FACTOR N_t

It was shown in Section 5.0 that the number of undetectable t-digit errors in a modulus 10 code of length L digits is given by an expression of the form

$$C(L, t) \times N_t .$$

This Appendix derives an expression for the factor N_t.

Consider a codeword with digits d_i for which Eq. (3) holds. That is, using the scrambling function in Table II,

$$\sum_{i=1}^{L} V^*_i (d_i) = 0 \pmod{M} \qquad \text{(Eq. A1)}$$

Suppose that some digits are converted by errors to d_i' such that $d_i = d_i'$ if there is no error in the i-th position, while $d_i \neq d_i'$ if there is an error. Then the new residue of the (possibly incorrect) combination is

$$RWS' = \sum_{i=1}^{L} V^*_i(d_i) + \sum_{i=1}^{L} [V_i(d_i') - V^*_i(d_i)] \pmod{10} \qquad \text{(Eq. A2)}$$

Now RWS' = 0 (mod 10) and the error is not detected if and only if the second summation on the right side above is divisible by 10. Let

$$E_i = [V^*_i(d_i') - V^*_i(d_i)] \pmod{10}$$

Each E_i equals zero if no error occurred in the i-th digit, and is a non-zero digit between 1 to 9 (mod 10) if an error occurred. Therefore, an error is not detected if and only if

$$\sum_{i=1}^{t} E_i = 0 \pmod{10} \qquad \text{(Eq. A3)}$$

where E_i is between 1 to 9, inclusive.

It follows that N_t is equal to the number of permutations of t non-zero digits E_i which satisfy Eq. (A3) This number is calculated below.

Special Case: Single-digit error ($t = 1$). Here every error is detected because changing any one value in Eq. (A1) will result in a non-multiple of 10. That is, no single digit can satisfy Eq. (A3). Hence, $N_1 = 0$.

General Case: $t > 2$. If it happens that the first t-1 error digits by themselves constitute an undetectable error pattern, so that

$$\sum_{i=1}^{t-1} E_i = 0 \pmod{10}$$

then there is no way to chose a non-zero E_t to satisfy Eq. (A3). Therefore, the summation above must be non- divisible by 10, and the last E_t makes Eq. (A3) true. The number of ways to have the above sum not equal to zero is $9^{t-1} - N_{t-1}$.

This leads to the following recursion formula for N_t.

$$N_t = 9^{t-1} - N_{t-1}$$
$$N_1 = 0 \qquad \text{(Eq. A4)}$$

The solution of the recursion equation is:

$$N_t = (9^t + 9)/10 \quad \text{for } t \text{ even}$$
$$= (9^t - 9)/10 \quad \text{for } t \text{ odd} \qquad \text{(Eq. A5)}$$

Calculated values of N_t are given in Table III.

APPENDIX B

PAIRWISE DIGIT EXCHANGE IN MOD 10 CODE

This Appendix derives some results on permutation of digits in the modulus 10 code which are used in Section 6.0 to analyze detection performance for delete and transfer errors.

Consider a codeword with L digits for which Eq. (3) holds, i.e., using the scrambling function in Table II,

$$\sum_{i=1}^{L} V^*i\ (d_i) = 0 \pmod M \qquad \text{(Eq. B1)}$$

Every odd-numbered digit has a weight of 2 whereas every even-numbered digit has a weight of unity.

Suppose that a contiguous subset of t (t even) digits are exchanged in pairs, such that d_i is exchanged with d_i+1, d_i+2 is exchanged with d_i+3, etc. The problem considered here is how many codewords remain valid codewords under such pairwise digit exchange.

In the following development, it is convenient to assume that $t < L$ so that there is no constraint on the choice of code digits in the range of positions to be pairwise permuted.

The effect of the exchange of, say, d_1 and d_2 is that the weighted sum in Eq. (B1) changes to

$$\sum_{i=1}^{t} V^*i\ (d_i) + [V^*2\ (d1) - V^*1\ (d1)] - [V^*2\ (d2) - V^*1\ (d2)]$$

Each of the bracketed terms above represents an increase in the weighted sum due to moving a digit from an odd-numbered position with weight 2 to an even-numbered position with unity weight. These amounts are tabulated in the last column of Table II.

Let the increase due to digit d be denoted as

$$H(d) = [V^*2(d) - V^*1(d)] \pmod{10}$$

H (d1) - H (d2)	Number of Cases	
0	10	two digits equal
	2	(e.g., "55") two digits equal ("09", "90")
5	8	
1	10	
2	10	
3	10	
4	10	
6	10	
7	10	
8	10	
9	10	
TOTAL =	100	

Thus, when digits d1 and d2 are transposed, the weighted sum of the codeword changes, unless the bracketed terms H(d1) - H(d2) cancel (mod 10) in which case the change goes undetected.

Running through all the 100 pairs of two digit combinations for d1 and d2 gives the following changes:

In general case when an even number t of digits in contiguous positions within a codeword are transposed in pairs, the weighted sum increases by

$$H(d1) - H(d2) + H(d3) - H(d4) + \ldots + H(d_{t-1} - H(dt) \qquad \text{(Eq. B2)}$$

Define

At = number of permutations of t digits for which Eq. (B2) equals zero (mod 10), including permutations with repeated digits

B_t = number of permutations of t digits for which Eq. (B2) equals 5 (mod 10)

C_t = number of permutations of t digits for which Eq. (B2) equals any digit other than 0 or 5 (mod 10)

From the tabulation above,

$$A_2 = 12 \qquad C_2 = 80$$
$$B_2 = 8 \qquad \text{(Eq. B3)}$$

There are three ways for Eq. (B2) to reduce modulo 10 to zero:

(1) The first t-2 terms in Eq. (B2) reduce modulo 10 to zero, and the last two terms also reduce to zero. This occurs in $A_{t-2} \times 12$ ways.

(2) The first t-2 terms in Eq. (B2) reduce modulo 10 to 5, and the last two terms also reduce to 5. This occurs in $B_{t-2} \times 5$ ways.

(3) The first t - 2 terms in Eq. (B2) reduce modulo 10 to a number other than 0 or 5, and the last two terms reduce to the modulo 10 complement, giving an overall value of zero (mod 10). This occurs in $C_{t-2} \times 10$ ways.

Combining the above three cases gives an equation for the number of t-digit permutations which remain valid codewords under pairwise digit exchange:

$$A_t = 12A_{t-2} + 8B_{t-2} + 10C_{t-2} \qquad \text{(Eq. B4)}$$

By a similar derivation

$$B_t = 8A_{t-2} + 12B_{t-2} + 10C_{t-2} \qquad \text{(Eq. B5)}$$

Since Eq. (B2) must reduce to some value from 0 to 9, all possibilities are exhausted by A_t, B_t, and C_t so that

$$C_t = 10^t - A_t - C_t \qquad \text{(Eq. B6)}$$

The solution of Eqs. (B3) to (B6) is

$$A_t = 2^{t-1} + 10^{t-1}$$
$$B_t = 10^{t-1} - 2^{t-1}$$
$$C_t = 8 \times 10^{t-1} \qquad \text{(Eq. B7)}$$