Edgar H. Sibley
Panel Editor

Decimal-oriented error detection schemes are explored in the context of one particular company project.

ERROR DETECTING DECIMAL DIGITS

NEAL R. WAGNER and PAUL S. PUTTER

We were recently engaged by a large mail-order house to act as consultants on their use of check digits for detecting errors in account numbers. Since we were not experts in coding theory, we looked in reference books such as *Error Correcting Codes* [7] and asked colleagues who were familiar with coding theory. Uniformly, the answer was: There is no field of order 10; the theory only works over a field. This article relates our experiences and presents several of the simple *decimal*-oriented error detection schemes that are available, but not widely known.

CHECK DIGITS FOR DECIMAL ERRORS

Human beings still use *decimal* numbers in everyday affairs, and when working with these numbers they make *decimal* mistakes—such as copying a single digit incorrectly or transposing two adjacent digits. Machines that read decimal numbers also make decimal-oriented mistakes—mainly just misreading a digit. To help protect against errors, decimal numbers used in business (such as account numbers) often have an extra *check digit*. There is usually a *check equation* which all digits, including the check digit, must satisfy. In this way the check digit itself is also protected against errors. If the check equation is not true then there is definitely an error present. If the check equation is true, however, one or more errors may still be present. A check digit also gives some protection against an attempt to use a random account number, since this act would be caught 90 percent of the time.

As an example, the check equation might add the digits, take the remainder on division by 10, and see if the result is zero. If the digits are $a_1, a_2, \ldots, a_n$, this says

$$(a_1 + a_2 + \cdots + a_n) \bmod 10 = 0.$$

Of course, if a_n is the check digit, then one must choose a_n so that the equation is true. This method will always detect an error in a single digit, and it will detect 90 percent of all errors. Nevertheless, it will not detect transpositions. If human beings are involved, transposing adjacent digits is a common error, so this scheme is not ideal in such a case. This method might be suitable for an optical scanning system that might occasionally have trouble reading a digit, but knows the position of the digit in error. If x is the unknown digit, and s is the check sum without x, then one has $(x + s) \bmod 10 = 0$, or $x = 10 - s$, if $s \neq 0$, and $x = 0$ if $s = 0$.

The ISBN number of a book uses a mod 11 check [5]. Here the check equation is

$$(a_1 + 2*a_2 + 3*a_3 + \cdots + n*a_n) \bmod 11 = 0, \quad n \leq 10.$$

If $n > 10$, just keep repeating weights from 1 to 10. For ISBN numbers, $n = 10$, and the digits are written in reverse order, with a_1 used as the check digit to make its calculation easier. Unfortunately, this check digit has a value from 0 to 10, and the ISBN number uses an X to represent 10. Thus, it is not really a *decimal* check digit. The mod 11 check detects any single error and transposition of any two digits, whether adjacent or not (assuming $n \leq 10$).

Many account numbers (such as MasterCard) use a special mod 10 check, often called the "IBM check." The check equation looks like

$$(2\#a_1 + a_2 + 2\#a_3 + a_4 + \cdots) \bmod 10 = 0,$$

with the understanding that $2\#a_i$ is to be calculated by

multiplying a_i by 2 and adding the decimal digits, that is, $2\#a_i = (2*a_i) \text{ div } 10 + (2*a_i) \bmod 10$. For example, if the account number is 54996, then the check equation without the check digit is

$$(2\#5 + 4 + 2\#9 + 9 + 2\#6) \bmod 10$$
$$= (1 + 4 + 9 + 9 + 3) \bmod 10 = 26 \bmod 10 = 6,$$

so that the check digit must equal 4. This scheme detects all single-digit errors as well as all adjacent transpositions except for 09 and 90. Thus it catches 97.777 percent of adjacent transposition errors (88 out of 90).

Other schemes are used in practice. For example, United States banks put an eight-digit processing number on checks and add a check digit with check equation

$$(3*a_1 + 7*a_2 + a_3 + 3*a_4 + 7*a_5 + a_6 + 3*a_7 + 7*a_8 + a_9) \bmod 10 = 0.$$

These repeated factors of 3, 7, and 1 mean that adjacent transpositions of digits that differ by 5 are not caught. Thus 10 percent of adjacent transposition errors would go undetected. The authors were unable to discover the origin of this scheme.

THE CONSULTATION

As mentioned earlier, we were consulted about the process of evaluating an error detection scheme for new account numbers being introduced. The proposed scheme would use an eight-digit account number followed by four additional digits. Digit 9 would be chosen so that digits 1 through 9 would satisfy the ISBN mod 11 check. Digits 10 and 11 would be random, and digit 12 would be chosen so that all 12 digits would satisfy the IBM mod 10 check. Specifically, they asked for the *error rate*, and also asked if there was anything to be gained by adding the two-digit random number.

In turn, we had more questions than answers. What if the mod 11 check required a 10 for the ninth digit? What kind of errors were visualized? Were there security threats from malicious individuals? Finally, what was the purpose of the two random digits? We could only imagine that they would make it harder (100 times as hard) to guess a legitimate account number, even for someone who knew the method of checking. We wondered why the firm was asking *us* about the purpose of these two extra digits.

A senior official answered our questions and described the environment. The firm had a mailing list of 19 million names and addresses—one of their important resources. At that time, they had employees keying in each name and address by hand, perhaps 30–40 keystrokes, followed by the order itself. In the new proposed system, most customers would paste a mailing label on their order form with the new account number on it. The operator would key in only the account number, 12 keystrokes, followed by the order. They hoped to improve accuracy and eliminate keystrokes. They saw no external threats but were especially concerned about errors. (An error would mean merchandise incorrectly sent, and subsequent customer problems.)

In response to other questions, they said they would not use account numbers that had the ISBN mod 11 check digit come out to 10. The errors they envisioned as most common were what we expected: single errors or adjacent transpositions. The extra two random digits were there to *scatter* the account numbers. They would process the accounts in alphabetical order and were afraid that there might be many cases of "Joe Smith's" with very similar addresses, and they did not want the corresponding account numbers too close together.

FIRST RECOMMENDATIONS

We first recommended not using any check digits. The account number would be keyed in, looked up, and the associated name and address displayed to the operator, who could just compare the display with the mailing label.

We proposed to scatter account numbers in one of several ways (without extra random digits) and still process the accounts in alphabetical order. One could store successive account numbers in a small buffer (e.g., size 100), and pseudorandomly select them for assignment to a name and address, replacing the buffer location with the next account number. One could get a similar effect by simply using a fixed pseudorandom rearrangement of the numbers from 00 to 99, and assigning accounts in blocks of 100.

We also considered using a fixed pseudorandom sequence of length equal to the largest prime less than 100 million (99999989). Account numbers would be pseudorandomly scattered throughout the range, and the current *latest* account number (the seed) would show where to take up the sequence for new account numbers. In the end, we did not mention this scheme to the firm.

In one meeting, officials from the firm explained that they had a fixed hardware setup and would not be able to modify it. As configured, the names and addresses were not available online. Order information was keyed in and transferred on tape to a remote batch system where the account information was kept. They used schemes for matching the keyed-in name and address with the form that was actually in their system. These schemes were error prone, however, and there was no operator feedback about keying errors. They would be able to carry out the account number checks online and let the operator know immediately that the number was keyed in incorrectly, unless the errors happened to create another account number with valid check digits. They again emphasized their desire for a low error rate.

They never did seem to understand our proposals for scattering account numbers with no extra digits. However, they did agree with our suggestion that if two extra digits were used, the digits might as well also be

check digits. Such check digits would essentially be pseudorandom and should scatter account numbers as well as random digits. We tentatively suggested a mod 97 check for these two digits.

They also mentioned that the leading zeros of account numbers would not be keyed in. This caused us some concern because a dropped digit might be a fairly common error. We mentioned our discovery that the mod 10 check would not catch adjacent transpositions of 09 and 90. The senior official said he had heard this before, however.

LATER RECOMMENDATIONS

We gave several recommendations, depending on the desired error rate. One system would use two check digits. We had simulated errors to give evidence that the mod 11 and mod 10 checks together would yield a 1 percent error rate for random errors. Of course, the mod 10 check, by itself, would catch all single errors and all adjacent transpositions except for 09 and 90. With the mod 11 check in place, only terminal 09 or 90 transpositions would escape detection.

With four check digits, we recommended using a mod 97 check between the mod 11 and the mod 10. The mod 97 check equation just treats the whole 11-digit number as an integer and takes the remainder on division by 97, getting zero if valid. In effect, the weights are successive powers of 10, mod 97. We investigated other possible weights for a mod 97 check and concluded that successive powers of 10 are optimal (though not unique) for detecting adjacent double errors.

Using all four check digits, we simulated 370,000 random double errors and got no single valid account number, though 37 would have been expected from true random errors. This made it look as if all (or almost all) double errors would be caught, though we had no proof of this fact. Simulated random triple errors produced the expected 0.01 percent error rate. We recommended keying in leading zeros since we were able to produce simulation results showing approximately a 1 percent error rate from a single dropped digit if two check digits were used.

We wanted to propose a more elegant single mod 997 or mod 9973 check. This would be simpler to compute and would not require dropping account numbers, as the mod 11 check did.

We did not make this recommendation for several reasons. The firm was all geared up to use the mod 11 and mod 10 checks, and they believed in them. They were willing to add another check since that could not make the system weaker. Also, we did not know the best choice of weights to use for mod 997 or mod 9973 checks, and we had run out of time for further investigation. We could not *guarantee* that a single mod 9973 check was superior to the more complex setup, nor could we *guarantee* that it had no hidden weaknesses.

In the end, they decided they wanted the extremely low error rate provided by four check digits. They also decided to start account numbers at 10000000, so that there would be no leading zeros. Soon after, one of the programmers called with questions about implementing the mod 97 check, but seemed to get back on track. In a follow-up six months later, the senior official said they had implemented exactly what we had discussed and were very pleased with the outcome.

ANOTHER SCHEME: VERHOEFF'S WORK

Months after making our recommendations, we had stopped looking for a single check decimal that would detect all single errors and all adjacent transpositions, and instead we were working on a proof that no such check existed. Then we located the interesting 1969 monograph by J. Verhoeff [2, 8]. In this work we found that others had published such proofs, implicitly assuming a fixed weight or permutation applied to each digit, followed by addition modulo 10. But Verhoeff discovered more complex methods that will detect all these errors. (During final revisions of the present article, [3] appeared, which describes the rediscovery of Verhoeff's methods by Gumm [4].)

Verhoeff also gives a thorough discussion of various alternative checks. In addition to many methods for detecting all single errors and all adjacent transpositions, he presents other common kinds of human errors, along with checks to detect as many of these errors as possible. Among errors mentioned as occurring in a specific study of 12000 errors are the following, with percentages in parentheses. (In each case, a is not equal to b, but c can be any decimal digit.)

- single errors: $a \rightarrow b$ (60 to 95 percent of all errors)
- adjacent transpositions: $ab \rightarrow ba$ (10 to 20 percent)
- twin errors: $aa \rightarrow bb$ (0.5 to 1.5 percent)
- jump transpositions: $acb \rightarrow bca$ (0.5 to 1.5 percent)
- jump twin errors: $aca \rightarrow bcb$ (below 1.0 percent) (Longer jumps are even rarer.)
- phonetic errors: $a0 \rightarrow 1a$ (0.5 to 1.5 percent) (Phonetic, because in some languages the two have similar pronunciation, e.g., thirty and thirteen.)
- omitting or adding a digit (10 to 20 percent)

One of Verhoeff's best checks, which we present in complete detail, will catch all single errors and all adjacent transpositions. It will also catch 95.555 percent of twin errors, and 94.222 percent of jump transpositions and jump twin errors (or 94.1666 percent of these jump errors if a, b and c must all be distinct). Finally, it will catch 95.3125 percent of phonetic errors, assuming that a ranges from 2 to 9. We programmed this check, exactly as we will describe, and obtained these figures by simple exhaustive search. (Verhoeff gives the same percentages.)

Verhoeff's clever idea was to use a method for combining the integers from 0 to 9 that is different from addition module 10. This method is based on a group known as the *dihedral group* D_5, represented by the symmetries of a pentagon [1]. Multiplication in this group is not *commutative*, that is, $a*b$ is not always

equal to $b*a$. (We use * for the group operation and 0 for the identity element.) Just combining digits over this group works better than adding digits mod 10 since in both cases all single errors are caught, yet in D_5 two-thirds of the adjacent transpositions are caught (whereas none are caught mod 10). Verhoeff considers check equations of the form

$$f_1(a_1)*f_2(a_2)* \cdots *f_n(a_n) = 0,$$

where multiplication is over D_5 and $f_1, f_2, \ldots, f_n$ are permutations of the ten digits. He is able to get an excellent check in the special case where f_i is just the ith iteration of a fixed permutation f.

This check is not hard to program and is quite efficient, but it does employ several tables. First is the matrix $Op[i, j]$, $0 \le i \le 9$, $0 \le j \le 9$, giving the result of multiplying i by j in D_5 as shown in Figure 1. Next we need an array $Inv[i]$, $0 \le i \le 9$, giving the inverse of i in D_5, i.e., $i*Inv[i] = 0$ in D_5.

$Inv[i]$ $(0 \le i \le 9)$ = [0, 4, 3, 2, 1, 5, 6, 7, 8 and 9].

Finally, the check needs an array $F[i, j]$, $0 \le i \le 7$, $0 \le j \le 9$, where $F[i, *]$ is the ith permutation function, used on digit a_i. Define this by

$$F[0, j] = j, \quad 0 \le j \le 9,$$

and

$$F[1, j]\ (0 \le j \le 9) = [1, 5, 7, 6, 2, 8, 3, 0, 9 \text{ and } 4].$$

Then, the $F[i, j]$, for $2 \le i \le 7$, are defined by the formula

$$F[i, j] = F[i-1, F[1, j]].$$

Note that $F[8, *] = F[0, *]$, so the table entries start over again. With these tables the check equation is simple to define. Assume the account number is an array $a[i]$, $0 \le i \le n$, with $a[0]$ used for the check digit.

```
begin (* check that array a is valid *)
  Check := 0;
  for i := 0 to n do
    Check := Op[Check, F[i mod 8, a[i]]];
  if Check <> 0 then Error
end
```

In order to create the check digit $a[0]$, use

```
begin (* compute check digit for position a[0] *)
  Check := 0;
  for i := 1 to n do
    Check := Op[Check, F[i mod 8, a[i]]];
  a[0] := Inv[Check]
end
```

This works because $F[0, *]$ is the identity permutation. If one wanted a general $a[i]$ as the check digit, one would also need the inverse of $F[i, *]$.

ERROR CORRECTION

A year later we learned that for two mod 11 check digits, a single-error *correcting* code is available: a Hamming mod 11 code, discovered in 1949. This and related codes are discussed at length in a recent coding theory book [6]. The code uses two check equations, with up to ten data digits and two check digits.

$$\begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
1 & 2 & 3 & 4 & 0 & 6 & 7 & 8 & 9 & 5 \\
2 & 3 & 4 & 0 & 1 & 7 & 8 & 9 & 5 & 6 \\
3 & 4 & 0 & 1 & 2 & 8 & 9 & 5 & 6 & 7 \\
4 & 0 & 1 & 2 & 3 & 9 & 5 & 6 & 7 & 8 \\
5 & 9 & 8 & 7 & 6 & 0 & 4 & 3 & 2 & 1 \\
6 & 5 & 9 & 8 & 7 & 1 & 0 & 4 & 3 & 2 \\
7 & 6 & 5 & 9 & 8 & 2 & 1 & 0 & 4 & 3 \\
8 & 7 & 6 & 5 & 9 & 3 & 2 & 1 & 0 & 4 \\
9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0
\end{pmatrix}$$

FIGURE 1. $Op[i, j]\ (0 \le i \le 9)(0 \le j \le 9)$

$$(a_1 + 2*a_2 + 3*a_3 + \cdots + 10*a_{10} + a_{12}) \bmod 11 = 0,$$

and

$$(a_1 + a_2 + \cdots + a_{10} + a_{11}) \bmod 11 = 0.$$

To correct any single error, get the amount of the error from the second check and the position of the error from the first. (Position 12 is a special case.)

Unfortunately, this approach will also erroneously correct a transposition by changing a single digit. One could be satisfied with correcting about 94 percent of all single errors, detecting the remaining single errors, and detecting all adjacent transpositions. As an alternative, one could use only nine data digits plus two check digits, leaving off a_{12} in the first check equation. Now the system corrects all single errors and detects all transpositions (adjacent or not).

With mod 11 checks one must be prepared to use an X to represent 10 in the check digits, or else one must throw out 17.36 percent of the possible sequences of data digits (since 21 out of each 121 sequences have one or both of the two check digits equal to 10). Our firm was willing to discard some account numbers, but would not have been interested in this approach because they had no need for error correction.

CONCLUSIONS

The most common checks for error detection using a single digit are the ISBN mod 11 check and the IBM mod 10 check, but each of these has a significant disadvantage. The ISBN mod 11 check needs an extra character to represent 10, or else requires that one eliminate account numbers with check digit 10. The IBM mod 10 check does not catch all adjacent transpositions, failing for 09 and 90. A fairly simple and efficient check (Verhoeff's check) eliminates these disadvantages.

For a very low error rate, firms might consider *two* check digits and a mod 97 check, with weights succes-

sive powers of 10. This check catches 100 percent of each of the errors on Verhoeff's list, and 99.94 percent of adjacent double errors. Overall, it catches nearly 99 percent of all errors, compared with 90 percent for the single digit checks, and it is easy to calculate by hand.

Two mod 11 check digits perform even better than a mod 97 check, including single error correction with ten or fewer data digits, but this approach has the disadvantages mentioned above for mod 11 checks: Either an *X* is needed to represent 10, or account numbers with one of the check digits equal to 10 must be eliminated.

Finally, in dealing with commercial firms as a consultant, one should avoid excessively technical or academic recommendations. Companies want workable, simple solutions that they understand and believe in. We regarded our consulting firm's final scheme with its three separate checks as an overly complicated solution, but it satisfied the company's needs. In the same way, the IBM mod 10 check, because of its simplicity, is superior to Verhoeff's check for most commerical uses.

REFERENCES

1. Birkhoff, G., and MacLane, S. *A Survey of Modern Algebra.* 4th ed. Macmillan, New York, 1977.
2. Brinn, L.W. Algebraic coding theory in the undergraduate curriculum. *Am. Math. Mon. 91*, 8 (Oct. 1984), 509–573.
3. Gallian, J.A., and Winters, S. Modular arithmetic in the marketplace. *Am. Math. Mon. 95*, 6 (June–July 1988), 548–551.
4. Gumm, H.P. A new class of check-digit methods for arbitrary number systems. *IEEE Trans. Inf. Theory 31*, 1 (Jan. 1985), 102–105.
5. Hamming, R.W. *Coding and Information Theory.* 2d ed. Prentice-Hall, Englewood Cliffs, N.J., 1986.
6. Hill, R. *A First Course in Coding Theory.* Clarendon Press, Oxford, 1986.
7. Peterson, W.W., and Weldon, E.J. *Error-Correcting Codes.* 2d ed. MIT Press, Cambridge, Mass., 1972.
8. Verhoeff, J. *Error Detecting Decimal Codes.* Mathematical Centre Tract 29, The Mathematical Centre, Amsterdam, 1969.

CR Categories and Subject Descriptors: B.4.5 **Input/Output and Data Communications**]: Reliability, Testing and Fault-Tolerance—*error checking*; E.4 [**Data**]: Coding and Information Theory—*nonsecret encoding schemes*

General Terms: Reliability

Additional Key Words and Phrases: Check digits, decimal errors, *error detection*

ABOUT THE AUTHORS:

PAUL PUTTER is a member of the technical staff of AT&T Bell Laboratories, where he works on compilation systems. His research interests include programming languages and compilers. Author's present address: AT&T Bell Laboratories (SF D-321), 190 River Road, Summit, NJ 07901.

NEAL R. WAGNER is an associate professor of computer science at the University of Texas at San Antonio. His research interests are cryptography and database security. Author's present address: Division of Mathematics, Computer Science and Statistics, University of Texas at San Antonio, San Antonio, TX 78285.