It should also be noted that overpacking is allowed in the above described optimal histogram matching problem. If overpacking is not allowed and the monotone property is dropped, then the problem becomes the classical bin-packing problem [4] which is known to be *NP*-complete.

Subsequent to this paper, Chow and Kou found a dynamic programming algorithm for optimal histogram matching which also has time complexity $O(k1 \times k2)$ [1].

Acknowledgments. The authors are indebted to G. Manacher for his many helpful suggestions and comments, and to the referee who pointed out an error in an earlier version of this paper.

Received June 1977; revised April 1978

References

1. Chow, W.M., and Kou, L.T. Matching two digital pictures. IBM Res. Rep. RC6870, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., Nov. 1977.
2. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations,* R.E. Miller and J.W. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85–103.
3. Rosenfeld, A., and Kak, A.C. *Digital Picture Processing.* Academic Press, New York, 1976, pp. 173–175.
4. Yao, A. Concrete computational complexity. Ph.D. Diss., U. of Illinois at Urbana-Champaign, 1975.

Programming Techniques

S.L. Graham, R.L. Rivest Editors

Counting Large Numbers of Events in Small Registers

Robert Morris
Bell Laboratories, Murray Hill, N.J.

It is possible to use a small counter to keep approximate counts of large numbers. The resulting expected error can be rather precisely controlled. An example is given in which 8-bit counters (bytes) are used to keep track of as many as 130,000 events with a relative error which is substantially independent of the number *n* of events. This relative error can be expected to be 24 percent or less 95 percent of the time (i.e. $\sigma = n/8$). The techniques could be used to advantage in multichannel counting hardware or software used for the monitoring of experiments or processes.

Key Words and Phrases: counting
CR Categories: 5.11

A Counting Problem

An *n*-bit register can ordinarily only be used to count up to $2^n - 1$. I ran into a programming situation that required using a large number of counters to keep track of the number of occurrences of many different events. The counters were 8-bit bytes and because of the limited amount of storage available on the machine being used, it was not possible to use 16-bit counters. Using an intermediate size counter on a byte-oriented machine would have considerably increased both the complexity and running time of the program.

The resulting limitation of the maximum count to 255 led to inaccuracies in the results, since the most common events were recorded as having occurred 255 times when in fact some of them were much more

Author's address: Bell Laboratories, Murray Hill, NJ 07974.

frequent. I looked in many directions for a solution to the problem with the following constraints in mind. Running time was important because the program was already painfully slow. There was no significant additional space available in the machine. On the other hand, precise counts of the events were not necessary since the processing was statistical in nature and a reasonable margin of error was tolerable.

A Simple Solution

The most obvious way to count more than 255 events in an 8-bit register is to count only every other event. This can be done with a modest amount of error by simply flipping a coin at every event to decide whether or not to make the count. Not only is the expected error small, but it can be precisely described. In particular, if the number of events that have occurred is n, then the expected value for the value v in the counter is $n/2$ and the standard deviation is

$$\sigma = \sqrt{v/2}$$

so that by the time 400 events have occurred, $v = 200$ and $\sigma = 10$. One can expect that 95 percent of the time, the number of events estimated from $2v$ is within 40 of the actual count, an error of 10 percent.

This approach can be extended in the obvious way to count yet larger numbers of events with correspondingly increased expectations of error by simply using an appropriately weighted coin. This appealing approach did not solve my problem because even though the absolute error was relatively small, the relative error was intolerably high for small counts. In fact if one event had occurred, I was guaranteed a 100 percent error.

A mixed approach is possible and overcomes this problem by keeping actual counts up to some preset number and using the coin-flipping approach above that point. It is a generalization of this approach that led to the successful solution to my problem.

A Generalization

Suppose that instead of trying to keep track of the number n of events or of some constant multiple of n, that we keep in the register the logarithm of the number of events and devise some sort of approach as was set out above. Then if the value (which is a logarithm) suffers from a given absolute error, then the number of events which this logarithm reflects is affected only by a relative error. This is what is generally desired.

The simplest approach that uses this idea is to proceed as follows: The value v stored in the register is imagined to represent

$$v(n) = \log(1 + n)$$

where n is the number of events that have occurred. Zero value corresponds to zero events.

Suppose that in the midst of counting, we have the value v stored in the register. Whenever we get another event, we attempt to modify the contents of the register in the most appropriate way. All we have is the value v. As far as we know, the best estimate of the number of events so far is

$$n_v = e^v - 1$$

so that the number of events including the current one is e^v. The value that we would like to return to the register is

$$v' = \log(1 + e^v)$$

but this is not in general an integer and we cannot just truncate or round the quantity for fear of accumulating serious error. Instead we let $v' = k + f$ where k is an integer and $0 \leq f < 1$, and return either the value k or the value $k + 1$ to the register as the new value of v with appropriately chosen probabilities so that the expected value of v is correct.

The correct thing to do is to compute

$$1/\Delta = n_{v+1} - n_v.$$

In every case of interest, Δ will be between 0 and 1 and so we can ignore the possibility of an integer part. We then obtain a random number r from a random number generator uniformly distributed in the interval $0 \leq r < 1$ and

if $\Delta > r$, set $v = v + 1$
if $\Delta \leq r$, set $v = v$.

It is easy to prove that we have not by this procedure disturbed the expected value of the contents of the register. To prove this, observe that the random procedure substitutes for the correct value v' either

v with probability $1 - \Delta$, or
$v + 1$ with probability Δ.

Then the expected value of n represented by the contents of v is equal to

$$\begin{aligned}\Delta n_{v+1} + (1 - \Delta)n_v &= \Delta(n_{v+1} - n_v) + n_v \\ &= \frac{e^{v+1} - 1 - e^v + 1}{e^v(e - 1)} + n_v \\ &= n_v + 1.\end{aligned}$$

After one event has occurred, the register contains the value 1 with probability .59, or 0 with probability .41. When we come to interpret this value we would conclude that the number of events was

1.7 59 percent of the time, and
0 41 percent of the time.

The expected value in the register is correct, but the actual value is way off.

We can jiggle the parameters of the method so that a count of one results in a register value of one and then the random rounding procedure has no effect on the first count. The function

$$v = \log(n + 1)/\log(2)$$

has just the property we want. This formula is of course independent of the base of the logarithms.

A General Solution

The class of functions that I have used and analyzed are the functions

$$v(n) = \log(1 + n/a)/\log(1 + 1/a)$$

where the parameter a controls both the maximum count that can be held in the register and the expected error. The constant $\log(1 + 1/a)$ in the denominator serves only to force $n = 1$ to correspond to $v = 1$ so that the random procedures have no effect on the first count and counts of 0 and 1 are represented exactly. It is in this way that good relative accuracy is preserved for small counts. The maximum value n that can be represented using the parameter a can be calculated from the inverted formula

$$n_v = a((1 + 1/a)^v - 1).$$

The expected error in the estimated value of n after n counts can be calculated from the formula

$$\sigma^2 = n(n - 1)/2a.$$

This formula can be proved by induction on n.

Let us inspect the performance of this method using the parameter $a = 30$. The largest value that can be represented in an 8-bit counter is about 130,000. The standard deviation σ is approximately equal to $n/8$ which implies that the relative error is nearly independent of n and that 95 percent of the time the relative error will be less than 24 percent. Larger values of a will lead to smaller maximum counts and, of course, to smaller relative error.

There is no need to compute any of the logarithms or powers during the counting process. A table containing the 255 values of Δ can be precomputed by the formula $\Delta_j = (a/(a + 1))^j$ and accessed when a new count is made. The random number generator can be of the simplest sort and no great demands are made on its properties.

The distribution of errors is somewhat asymmetric for small counts, but as n becomes larger, the distribution closely approximates the normal distribution. In the example above where $a = 30$, the normal error curve gives a useful estimate of the error distribution for counts greater than about 20.

Received June 1975; revised December 1977

Programming Techniques
S.L. Graham, R.L. Rivest Editors

An Analysis of Algorithms for the Dutch National Flag Problem

Colin L. McMaster
University of California

Solutions to the Dutch National Flag Problem have been given by Dijkstra [1] and Meyer [3]. Dijkstra starts with a simple program and arrives at an improved program by refinement. Both of the algorithms given by Dijkstra are shown to have an expected number of swaps which is $\frac{2}{3}N + \theta(1)$ and that these values differ at most by $\frac{1}{3}$ of a swap and asymptotically by $\frac{1}{4}$ of a swap. The algorithm of Meyer is shown to have expected swap complexity $\frac{5}{9}N$.

Key Words and Phrases: algorithmic analysis, Dutch National Flag Problem, refinement, structured programming

CR Categories: 4.0, 5.24, 5.25, 5.3

Introduction

Dijkstra [1] has defined a problem which he calls the Problem of the Dutch National Flag. It may be stated as follows. There is a row of N buckets numbered from 1 to N. The buckets are arranged in numerical order with bucket 1 on the left and bucket N on the right. Each bucket contains exactly one pebble and each pebble is

Research sponsored by Joint Services Electronics Program Grant F 44620-76-C-0100 and National Science Foundation Grant MCS 76-15036.

Author's address: Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.