GRAY CODES FOR RANDOMIZATION PROCEDURES

BY

PERSI DIACONIS and SUSAN HOLMES

TECHNICAL REPORT NO. 444

JANUARY 1994

PREPARED UNDER THE AUSPICES

OF

NATIONAL SCIENCE FOUNDATION GRANT DMS92–04864

DEPARTMENT OF STATISTICS

STANFORD UNIVERSITY

STANFORD, CALIFORNIA
Gray Codes for Randomization Procedures

By

Persi Diaconis and Susan Holmes

Technical Report No. 444

January 1994

Prepared Under the Auspices

Of

National Science Foundation Grant DMS92-04864

Also Supported by Teledyne Electronics and issued as Technical Report No. 10, Department of Statistics, Stanford University.

Department of Statistics
Stanford University
Stanford, California
GRAY CODES FOR RANDOMIZATION PROCEDURES

PERSI DIACONIS  SUSAN HOLMES
Department of Mathematics  INRA
Harvard University  Unité de Biométrie
One Oxford Street  2, Place Pierre Viala
Cambridge, MA 02138  34060 Montpellier
FRANCE
(visiting Stanford University)

Abstract

We introduce a simple combinatorial scheme for systematically running through a complete enumeration of sample reuse procedures such as the bootstrap, Hartigan’s subsets, and various permutation tests. The scheme is based on Gray codes which give “tours” through various spaces, changing only one or two points at a time. We use updating algorithms to avoid recomputing statistics and achieve substantial speedups. Several practical examples and computer codes are given.

1. Introduction

This paper introduces Gray codes as systematic procedures for complete enumeration of sample reuse procedures such as the bootstrap and permutation tests. Gray codes are a well developed part of combinatorial algorithms. They allow a “tour” of a set of combinatorial objects making minimal changes at each step. Updating algorithms, allow us to avoid complete recomputation of statistics of interest.

As an example, consider the law school data used by Efron (1982). The data consists of 15 pairs of numbers (GPA, LSAT) for a sample of American law schools. The correlation coefficient is \( \hat{\rho} = .776 \). The bootstrap gives a perturbation analysis as explained in Section 3. We carried out an exact computation of the bootstrap distribution using the Gray codes. A histogram of the values of \( \rho \) is shown in Figure 1.1. There is a “corner” in this figure at about .9 which does not show up in even large Monte Carlo replications, Figure 1.2. We discuss this corner further in Section 4 (the examples section). The point for now is that there are features of the distribution picked up by the present analysis not available by routine Monte Carlo or asymptotic analysis.

Section 2 presents the original Gray code for binary \( n \)-tuples and gives applications to Fisher’s randomization test and Hartigan’s subsets. These problems can also be treated exactly using the fast Fourier transform on binary \( n \)-tuples. We show that the Gray code provides an improvement on these approaches in many situations.
Figure 1.1 Exhaustive Bootstrap for the Correlation Coefficient of the Law School Data
Figure 1.2 Monte Carlo Bootstrap for the Law School Data, $B = 40,000$
Section 3 develops Gray codes for the bootstrap. There the combinatorial space consists of compositions of \( n \). We explain the gray codes, review the updating literature, discuss algorithms for \( m \) and \( n \) versions of the bootstrap and discuss a variety of uses of these complete enumeration procedures.

Section 4 presents some comments and examples, section 5 discusses other non-parametric procedures where suitable Gray codes can be applied.

One can only hope to enumerate completely for moderate sample sizes \( (n \leq 20 \text{ today}) \). For larger sample sizes partial enumeration through carefully spaced points is discussed in Diaconis and Holmes (1994a). Another idea is to use a small dose of randomness, not as much as Monte Carlo, by doing a random walk between close points so as to be able to use updating procedures all the same, this is detailed in the case of exploring the tails of a bootstrap distribution in Diaconis and Holmes (1994b).

As will emerge from the examples below, the exact results give a good feel for the usefulness of asymptotics.

2. The original Gray code, Fisher’s test, and Hartigan’s subsets.

A. The original Gray code.

Let \( Z^n_2 \) be the set of binary \( n \)-tuples. This may be identified with the vertices of the usual \( n \)-cube or with the set of all subsets of an \( n \) element set. The original Gray code gives an ordered list of \( Z^n_2 \) with the property that successive values differ only in a single place. For example, when \( n = 3 \) such a list is

\[
000, 001, 011, 010, 110, 111, 101, 100.
\]

It is easy to give a recursive description of such a list, starting from the list for \( n = 1 \) (namely 0,1). Given a list \( L_n \) of length \( 2^n \), form \( L_{n+1} \) by putting a zero before each entry in \( L_n \), and a one before each entry in \( L_n \). Concatenate these two lists by writing down the first followed by the second in reverse order. Thus from 0,1 we get 00,01,11,10 and then the list displayed above for \( n = 3 \).

For \( n = 4 \) the list becomes:

\[
0000, 0001, 0011, 0010, 0110, 0101, 0100, 1100, 1110, 1111, 1101, 1010, 1011, 1001, 1000.
\]

Gray codes were invented by F. Gray (1939) for sending sequences of bits using a frequency transmitting device. If the ordinary integer indexing of the bit sequence is used then a small change in reception, between 15 and 16, for instance, has a large impact on the bit string understood. Gray codes enable a coding that minimizes the effect of such an error. A careful description and literature review can be found in Wilf (1989). One crucial feature: there are non-recursive algorithms for providing the successor to a vector in the sequence in a simple way. This is implemented through keeping track of the divisibility by 2 or the step number, see algorithm in appendix.
We study two uses for such Gray codes here: they provide a way of complete enumeration for Fisher’s randomization procedure and for Hartigan’s subset procedure. Similar procedures can be used for signed rank statistics and the usual two sample (m out of n) tests.

B. Fisher’s randomization test.

Fisher (1935) gave a permutation justification for the usual test for n paired observations \((x_1, y_1), \ldots, (x_n, y_n)\). In his example (Darwin’s Zea data) \(x_i\) and \(y_i\) were real numbers representing plant height for treated and untreated plants. Darwin had calculated the mean difference. Fisher gave a way of calibrating this by calculating \(d_i = |x_i - y_i|\) and considering all \(2^n\) possible sums \(S_n = \epsilon_1 d_1 + \cdots + \epsilon_n d_n\) with \(\epsilon = \pm 1\).

We observe that a Gray code allows us to systematically run through all \(2^n\) patterns. One need only update the present value of the sum by changing a single value. Furthermore the symmetry of the problem saves a factor 2 of the computations. Figure 2.1 shows a histogram of the randomization distribution based on this procedure applied to the Zea data. The original value of the sum \(S_0\) is 39.25 inches. This clearly falls in the tail of the distribution. In fact the exact proportion of sign patterns with a larger absolute sum is .0518 (as Fisher also calculated).

The sum \(S_n = \epsilon_1 d_1 + \epsilon_2 d_2 + \cdots + \epsilon_n d_n\) has an approximate normal distribution as a sum of independent not identically distributed random variables.

Figure 2.2a shows a plot of \((F_{\text{true}}(z) - \Phi(z))\) vs \(z\). The normal approximation is quite good. It gives a p-value of 0.0538. Fisher himself suggested using the studentized statistic \(T_n = \frac{S_n}{\hat{s}_n}\) (where \(\hat{s}_n^2\) is the variance of the vector \((\epsilon_1 d_1, \epsilon_2 d_2, \cdots, \epsilon_n d_n)\)). This is a one-to-one increasing function of \(S_n\), so the number of sign patterns such that \(T_n\) is strictly larger than the observed \(T_n\) (denoted by \(T_{n0}\) ) equals the number of sign patterns such that \(S_n\) is strictly larger than the observed \(S_n\) (\(S_{n0}\)). Call this common number \(M_n\) (it is 835 in the Zea data case), on the whole the errors in Figure 2.2b seem smaller than the errors in Figure 2.2a.

Figure 2.2b shows the difference between the true distribution of \(T_n\) and the distribution of a t-statistic with \(n - 1\) degrees of freedom versus \(x\) (which we will call \(G_{\text{true}}(x) - F_{n-1}\)). The errors are again quite small. Using the t-approximation gives a p-value of .0497 for the Zea data.

It is natural to ask which of the two approaches for approximating \(M_n\) (use of \(S_n\) and a normal approximation to its distribution or use of \(T_n\) and its t-approximation) is more accurate. We can give an asymptotic answer using second order corrections to the two permutation distributions.

Albers, Bickel and Van Zwet (1974) give a rigorous treatment proving that the Edgeworth correction to the true distribution (Theorem 2.1):

\[
F_e(z) \approx \Phi(z) + \phi(z) \frac{\kappa}{4n} (z^3 - 3z) \quad \text{with} \quad \kappa = n \frac{\sum x_i^4}{(\sum x_i^2)^2} \quad \text{and} \quad z = \frac{x}{\sigma_n}
\]

is accurate to \(O(n^{-4})\) under mild regularity conditions.

Figure 2.2c shows \((F_{\text{true}}(z) - F_e(z))\) versus \(z\) for the Zea data. This is an improvement over the already good fit of Figure 2.2a.
In joint unpublished work with Tom DiCiccio we have also computed the Edgeworth expansion for the studentized statistic $T_n$:

$$G_{est}(z) \simeq \Phi(z) - \frac{\phi(z)}{12n} (z^3(\kappa - 6) - 3z(2 - \kappa))$$

the difference $G_{true}(z) - G_{est}(z)$ is plotted in Figure 2.2d. Again the Edgeworth approximation gives an improvement on the t-approximation.

Combining our approximations we have the following general result:

**Lemma 2.1**: With our present notation, the normal distribution (approximation to $S_n$) provides a more accurate estimate for $M_n$ than the t-approximation to second order if and only if $\kappa \leq 3/2$.

Where $\kappa$ is as defined in equation (2.1).

In Darwin's Zea data, $\kappa = 1.93$; as predicted the t-approximation is better.

---

*Figure 2.1 Randomization Distribution of $S_n$ for Zea Data*
Figure 2.2 Comparisons between the true and the approximate distributions
All of these plots are based on complete enumeration using Gray coding.

We note that there is no reason to use the mean (Pearson (1937), Efron (1969)). Fisher used it because Darwin had. The Gray code can be useful with any statistic and savings increase if efficient updating for single changes are available. We discuss this in more detail in Section 3D.

Fisher's test and this data set have been extensively discussed in the statistics literature. Fisher himself avoided complete computation by looking at the tails and coming in carefully. Usual approximations proceed by Monte Carlo, however Pagano and Trichler (1983) give a fast Fourier algorithm for complete enumeration. We give a different Fourier implementation and further discussion in Section 2D. Lehmann (1975) and Trichler (1984) discuss exact confidence intervals derived from Fisher's randomization test. Manly (1992) carries out the exact computation without using an updating procedure. Basu (1980) and his discussants debate the logic of such "permutation" tests. Maritz (1981) gives a clear presentation in his book of more general two-sample location problems. The Gray code approach can be used for \( n \leq 30 \). It also adapts to the vector valued \( T^2 \) statistic studied by Eaton and Efron (1970).

C. Hartigan's subsets.

Hartigan (1969) developed an exact confidence interval for a location parameter. This is explained in Maritz (1979) and Efron (1982). If \( \mathcal{X}_n = \{x_1, x_2, \cdots, x_n\} \) is a sample observed from \( F_{\theta} \) with \( \text{Prob}(X \in A) = \int_A f(x - \theta) \text{dx} \) an \( M \)-estimate \( \hat{\theta} \) for \( \theta \) is any solution of \( \Sigma \psi(x_i - z) = 0 \). Here \( \psi \) is anti-symmetric and increasing.

There are \( 2^n - 1 \) nonempty subsets of the original \( n \) data points. Any of these can be used to generate a new \( \tilde{\hat{\theta}}(s), s = 1, \cdots, 2^n \). Hartigan (1969) showed that the \( 2^n - 1 \) values, \( \tilde{\hat{\theta}}(s) \), once ordered, partition the line into \( 2^n \) intervals and that each of these has probability "exactly" \( \frac{1}{2^n} \) of containing the true value of \( \theta \). This leads to exact confidence intervals for \( \theta \).

It is straightforward to run through all subsets via a Gray code. Figures 2.4–2.6 below show examples based on subsets of a sample of size \( n = 10 \) generated by Maritz (1979) from a Cauchy distribution. Three location estimates have been made, the median, the \( M \)-estimate based on Huber’s Psi and the one based on Tukey’s bi-weight. As a comparison we have also provided the bootstrap of the median for this sample (Figure 2.7). The subsets procedures lead to distributions peaked at a few values. The bootstrap distribution seems slightly smoother.

Hartigan thought of the subset procedure much more broadly. One can induce a distribution for very general statistics using subsets. The subsampling procedures introduced by Hartigan (1969) were studied further by Forsythe and Hartigan (1970) who proved the method asymptotically efficient with respect to Student's \( t \) in the case of iid normal variables.

Gordon (1974a and 1974b) studied the efficiency in the long tailed case. He also gave convenient combinatorial conditions for classes of subsets to yield efficient procedures. Developing Gray codes for these classes is a challenging open problem.
Figure 2.4 Subsets for the median

90 % Confidence Interval : [-7.4 , 9]

Figure 2.5 Subsets with Huber’s Psi

90 % Confidence Interval : [-11.3 , 9.9]

90 % Confidence Interval : [-16.9 , 8.5]

90 % Confidence Interval : [-7.5 , 8.5]

Figure 2.6 Subsets with Tukey’s biweight

Figure 2.7 Bootstrap for the median
D. A Fourier connection.

This section amplifies a remark in Hartigan (1969). Let $Z_2^n$ be the binary $n$-tuples thought of as a group with coordinatewise addition (mod 2). For $x, y \in Z_2^n$, the inner scalar product (mod 2) is denoted $x \cdot y$.

Let $f: Z_2^n \to \mathbb{R}$ be a function. The Fourier transform of $f$ is defined by:

$$
\hat{f}(y) = \sum_x (-1)^{x \cdot y} f(x).
$$

Naive computation of this for a fixed value of $y$ takes $n \times 2^n$ operations. Allowing all values of $y$ will thus take order $n \times 2^n$ operations. The fast Fourier transform (FFT) is a collection of algorithms for computing $\hat{f}(y)$ for all $y$ which only needs $n2^n$ operations. It was originally conceived by Yates for statistical applications and Good (1957) abstracted to the modern FFT. A comprehensive review can be found in Elliott and Rao (1982). Varying a suggestion of Hartigan (1969) we observe that the FFT for this case allows exact computation of all $2^n$ possible sums in Fisher’s randomization test.

**Lemma:** Let $z_i, 1 \leq i \leq n$ be $n$ real numbers. Define

$$
f: Z_2^n \to \mathbb{R} \quad \text{by} \quad \begin{cases} f(e_i) = z_i & 1 \leq i \leq n \\ f(x) = 0 & \text{otherwise} \end{cases}
$$

$$
e_i = \begin{cases} (\delta_{ij})_{j=1,\ldots,n} \\ (0,\ldots,0,1,0,\ldots,0) \end{cases} \quad \text{the } i\text{th basis vector}.
$$

Then $\{\hat{f}(y)\}_{y \in Z_2^n}$ takes all $2^n$ values $\epsilon_1 z_1 + \cdots + \epsilon_n z_n; \epsilon_i \in \{-1, +1\}$.

**Remarks:** The FFT gives an order $n2^n$ algorithm for these problems. The Gray code gives an order $2^n$ algorithm. There is another approach to exact computation for this problem developed by Pagano and Trichler (1983). This uses the FFT on a different group and, in a sense that is developed below, works in polynomial time in $n$. The idea is to consider the distribution of $\epsilon_1 z_1 + \cdots + \epsilon_n z_n$ as a convolution of $n$ independent non-identically distributed random variables and then compute the law of the sum using the FFT. To implement this, $z_i$ must be taken as integers. We will give a running time analysis of the Pagano-Trichler algorithm.

Let $N = |z_1| + \cdots + |z_n|$ this is a measure of the size of the input. To do convolutions exactly one must work on $Z_{3N}$ (to avoid wrap-around). Then one must compute $n$ transforms of $\pm |z_i|$ (cost $n(3N)\log(3N)$). These must be multiplied together pointwise (cost $n \cdot 3N$) and then an inverse transform is needed (cost $3N\log 3N$). Thus, this FFT approach works in order $n \cdot (3N)\log(3N)$ operations. If all $|z_i| \leq B$, then $N \leq nB$ and one may say the algorithm works in a polynomial in $n$.

In practice, much depends on the size of $|z_i|$. As an example, Darwin’s data are originally measured in eighth’s of an inch. Multiplying by 8 gives $N = |z_1| + \cdots + |z_{18}| = 545$. Then order $15 \cdot (3N)\log(3N) = 181,470$ steps are needed for this FFT approach.
By contrast, Gray codes require an order of \(2^{14} = 16,384\) operations. (One update of a sum costs one operation at each step, generation of the Gray code costs at each step). This gives a nice class of examples where an exponential algorithm seems preferable to a quadratic algorithm.

Clearly the FFT on \(Z_{2N}\) will perform well when the \(z_i\) numbers are small. Pagano and Spino (1991) have shown how to adapt the \(Z_{2N}\) ideas to lightly trimmed means. Their approach is limited to examples of statistics that are close to linear and at present are not extensible to medians or general \(M\)-estimates.

In summary, for Fisher's randomization test and Hartigan's subsets, Gray coding is a useful procedure for samples up to size 30 or so. It works for general location estimates. For larger sample sizes on linear statistics the convolution approach is better.

3. The Bootstrap

A. Introduction. The bootstrap is a widely used tool for nonparametric procedures for bias correction, construction of confidence regions and estimation of sampling variability under minimal assumptions on the functional form of estimators or underlying distribution. Efron and Tibshirani (1993) give an accessible account with many examples and extensive references. Hall (1992) gives a theoretical development.

Bickel and Freedman (1981) carried out exact enumeration of the bootstrap distribution for the mean using the fast Fourier transform, Bailey (1991) used a similar approach for simple functionals of means. Fisher and Hall (1991) suggest exact enumeration procedures that we will compare to the Gray code approach in Section B below.

Let \(\mathcal{X}_n = \{x_1, x_2, \ldots, x_n\}\) be the original data supposed to be independent and identically distributed from an unknown distribution \(F\) on a space \(\mathcal{X}\). The bootstrap proceeds by supposing that replacement of \(F\) by \(F_n\), the empirical distribution, can provide insight on sampling variability problems.

Practically one proceeds by repeatedly choosing from the \(n\) points with replacement. This leads to bootstrap replications \(\mathcal{X}_n^* = \{x_1^*, \ldots, x_n^*\}\). There are \(n^n\) such possible replications, however these are not all different and grouping together replications that generate the same subset we can characterize each resample by its weight vector \((k_1, k_2, \ldots, k_n)\) where \(k_i\) is the number of times \(x_i\) appears in the replication. Thus \(k_1 + \cdots + k_n = n\).

Let the space of compositions of \(n\) into at most \(n\) parts be

\[
C_n = \{k = (k_1, \ldots, k_n), k_1 + \cdots + k_n = n, k_i \geq 0, k_i \text{ integer}\}.
\]

Thus \(|C_n| = \binom{2n-1}{n-1}\). We proceed by running through all compositions in a systematic way. Note that the uniform distribution on \(\mathcal{X}_n^*\) induces a multinomial distribution on \(C_n\)

\[
m_n(k) = \frac{1}{n^n} \binom{n}{k_1 \cdots k_n}.
\]

11
To form the exhaustive bootstrap distribution of a statistic $T(\mathcal{X}_n)$ one need only compute each of the $\binom{2^n-1}{n}$ statistics and associate a weight $m_n(k)$ with it. The shift from $\mathcal{X}_n$ to $C_n$ gives substantial savings. For the law school data, $n = 15, 15^{15} \approx 4.38 \times 10^{17}$ while $\binom{29}{14} \approx 7.7 \times 10^7$.

Efficient updating avoids multiplying such large numbers by factors of $n$. This is what makes the computation feasible. Gray codes generate compositions by changing two coordinates of the vector $k$ by one up and one down. This means that $m_n(k)$ can be easily updated by multiplying and dividing by the new coordinates. Similar procedures, discussed in Section C below allow efficient changes in the statistics of interest.

**B. Gray codes for compositions.**

Following earlier work by Nijenhuis, Wilf and Knuth, Klingsberg (1981) gave methods of generating Gray codes for compositions. We will discuss this construction briefly here, details can be found in Klingsberg (1982) and Wilf (1989).

For $n = 3$, the algorithm produces the 10 compositions of $C_3$ in the following order:

$$300, 210, 120, 030, 021, 111, 201, 102, 012, 003.$$  

The easiest way to understand the generation of such a list is recursive, construction of the $n$-compositions of $n$ can actually be done through that of the $(n - 1)$ compositions of $(n - i), i = 1, \ldots, n$.

For any $n$, the 2-composition is just provided by the list $n0, (n - 1)1, \ldots, 0n$, which is of length $n + 1$. So the 2-compositions out of 3 are $L(n, n - 1) = L(3, 2) = 30, 21, 12, 03$

the 2-compositons out of 2 are $L(n - 1, n - 1) = L(2, 2) = 20, 11, 02$

and 2-compositions out of 1 are $L(n - 2, n - 1) = L(1, 2) = 10, 01$

Finally there is only one 2-composition of 0: $L(0, 2) = 00$

The 3-out of 3 list is obtained by appending a 0 to the $L(3, 2)$ list, a 1 to the $L(2, 2)$ list, a 2 to the $L(1, 2)$ list and a 3 to the $L(0, 2)$ list. These four lists are then concatenated by writing the first, the second in reverse order, the third in its original order followed by the fourth in reverse order. This is actually written:

$$L(3, 3) = L(3, 2) \oplus 0, \quad \overline{L(2, 2)} \oplus 1, \quad L(1, 2) \oplus 2, \quad \overline{L(0, 2)} \oplus 3$$

and more generally

$$L(n, n) = L(n, n - 1) \oplus 0, \quad \overline{L(n - 1, n - 1)} \oplus 1, \quad L(n - 2, n - 1) \oplus 2, \ldots .$$

Here is an idea of the hamiltonian path on the simplex that the method generates:
Figure 3.1: Gray Code path on 3-simplex in perspective.
The same procedure leads to the 35 compositions of \( n = 4 \) in the following order:

\[
\begin{align*}
4000, &\quad 3100, \quad 2200, \quad 1300, \quad 0400, \quad 0310, \quad 1210, \\
2110, &\quad 3010, \quad 2020, \quad 1120, \quad 0220, \quad 0130, \quad 1030, \\
0040, &\quad 0031, \quad 0121, \quad 1021, \quad 2011, \quad 1111, \quad 0211, \\
0301, &\quad 1201, \quad 2101, \quad 3001, \quad 2002, \quad 1102, \quad 0202, \\
0112, &\quad 1012, \quad 0022, \quad 0013, \quad 0103, \quad 1003, \quad 0004.
\end{align*}
\]

The lists generated in this way have the property that two successive compositions differ only by \( \pm 1 \) in two coordinates.

Klingsberg (1982) provides a simple nonrecursive algorithm that generates the successor of any composion in this Gray code. This is crucial for the implemention in the the present paper. It requires that one keep track of the whereabouts of the first two non-zero elements and an updating counter. Both a \( S \) and a \( C \) version of the algorithm are provided in the appendix.

We conclude this subsection by discussing a different algorithm due to Nijenhuis and Wilf (1975, pp. 40-46) which runs through the compositions in lexicographic order (reading from right to left). This algorithm was suggested for bootstrapping by Fisher and Hall (1991).

**N - W algorithm to run through compositions \( C_n \).**

1. Set \( k_1 = n, k_i = 0, 2 \leq i \leq n \).
2. Let \( h = \) first \( i \) with \( k_i \neq 0 \). Set \( t = k_n, k_n = 0, k_1 = t - 1, k_{n+1} = k_{n+1} + 1 \).
3. Stop when \( k_n = n \).

For example, the following list gives all 35 composions in \( C_4 \) in the order produced by the N.W. algorithm:

\[
\begin{align*}
4000, &\quad 3100, \quad 2200, \quad 1300, \quad 0400, \quad 0310, \quad 1210, \\
2110, &\quad 3010, \quad 2020, \quad 1120, \quad 0220, \quad 0130, \quad 1030, \\
0040, &\quad 0031, \quad 0121, \quad 1021, \quad 2011, \quad 1111, \quad 0211, \\
0301, &\quad 1201, \quad 2101, \quad 3001, \quad 2002, \quad 1102, \quad 0202, \\
0112, &\quad 1012, \quad 0022, \quad 0013, \quad 0103, \quad 1003, \quad 0004.
\end{align*}
\]

The N-W algorithm (and so lexicographic order) changes at most 3 entries each time. Approximately half the time the first entry is non-zero. Then, two entries are changed, each by 1. On the other hand, there are some large changes. For example, following \( (0, \cdots, n, 0) \) is \( (n - 1, 0, \cdots, 1) \). The following lemma shows that the average move in the N-W algorithm involves switching 3 things. It is useful to observe that the number of switches in the N-W algorithm is even. It is \( 2j \) for steps of form \( (0, \cdots, 0, j, *, *, \cdots, *) \to (j - 1, 0, \cdots, 0, *, 1, *, \cdots, *) \) when \( j \geq 2 \).
Lemma 3.1 On \( C_n \) of (2.1) let \( P(2j) \) be the proportion of transitions in the N-W algorithm involving \( 2j \) switches, \( 1 \leq j \leq n \). Then, for fixed \( j \) and large \( n \)

\[
P(2) \sim \frac{3}{4}, \quad P(2j) \sim \frac{1}{2^{j+1}}.
\]

Further,

\[
\sum_{j=1}^{n} 2j P(2j) \sim 3.
\]

Proof. A step leads to two switches at the start if the step is \((k_1, k_2, \cdots)\) with \( k_1 > 0 \) or \((0, \cdots, 1, *, *, \cdots)\). The proportion of steps with \( k_1 > 0 \) is

\[
\frac{\binom{2n-1}{n-1} - \binom{2n-2}{n-1}}{\binom{2n-1}{n-1}} \rightarrow \frac{1}{2}.
\]

The proportion of steps of form \((0, 0, \cdots, 1, *, *, \cdots)\) where the initial 1 is in position \( a \), \( 1 \leq a \leq n-1 \), is

\[
\frac{\binom{2n-a-2}{n-2}}{\binom{2n-1}{n-1}} \rightarrow \frac{1}{2^{a+2}} \text{ for a fixed, } n \text{ large.}
\]

Summing in \( a \) shows that the proportion of such steps tends to \( \frac{1}{4} \) which gives \( P(2) \sim \frac{3}{4} \).

The argument for \( P(2j) \sim \frac{1}{2^{j+1}} \) is similar. Finally

\[
2 \cdot \frac{3}{4} + \sum_{j=2}^{\infty} (2j) \frac{1}{2^{j+1}} = 3.
\]

This suggests that the running time of the \( K-K \) algorithm is not so much better than the N-W algorithm. Of course, a factor of \( \frac{3}{2} \) becomes onerous when multiplied by 77 million. Further, the variable size of the N-W updating in bootstrap applications makes it complicated to take advantage of updating algorithms. The analysis here has some similarity to computation in Nijenhuis-Wilf (1978).

C. Updating.

The Gray code algorithms avoid complete recomputation of a statistic because only one or two values need be changed. In the numerical analysis literature these are often referred to as rank-one or rank-two updates. In discussing this we find it useful to introduce a notion of gain as:

\[
\text{gain} = (\text{number of floating point operations required for complete recomputation}) - (\text{number of operations required for the updated statistic}).
\]

For example suppose \( X_1, \cdots, X_n \) are real-valued and the statistic is \( \bar{X} = \frac{1}{n} \{X_1 + X_2 + \cdots, X_n\} \). It takes \( n-1 \) operations to recompute if the \( \frac{X_i}{n} \) are stacked instead of \( X_i \), and it takes 2 operations to delete a \( \frac{X_i}{n} \) and to add \( \frac{X_i}{n} \). Thus the gain is \( (n-1) - 2 = n - 3 \).
The simplest case of updating is for functions of means (like the correlation coefficient for instance). Here \(X_i\) takes values in \(R^p\) and the statistic of interest is \(f(X_1 + \cdots + X_n)\). The gain is \(p(n - 3)\). We note that most of the theoretical justification for the bootstrap takes place in this arena so there are many examples.

For regression problems, the "sweep" procedures in Dempster (1969) allow for up and down-dating. A more recent study of rank-two updating for Choleski decomposition is Bartels and Kaufman (1989). This allows gains for many multivariate methods that call for QR-decompositions, this includes discriminant analysis, multireponse regression and parts of principal components analysis. For the latter another classical procedure is available that is the rank-one update of singular values and eigenvalues described in Bunch and Nielsen (1979) and Bunch et al. (1978) that were built on the original idea of Golub and Reinsch (1970). They show that a rank one update for a singular value decompositioen can be obtained in order \(p^2\) steps for a \((n \times p)\) data matrix. Arbenz and Golub (1988) and Elhay et al. (1989) give recent variations. Updating techniques are used in the cross-validation literature, see for example Eastment and Krzanowski (1982) who use Bunch et al. (1978).

The problem of updating the median can be shown to provide a gain of at least \(\frac{n}{2}\), because if one uses the "efficient" algorithm for searching for the median as described, page 459 of Press et al. (1986) in their "Numerical Recipies", then the median can be seen as the solution of

\[
x_{med} = \frac{\sum_{i=1}^{n} \frac{x_i}{|x_i - x_{med}|}}{\sum_{i=1}^{n} \frac{1}{|x_i - x_{med}|}}.
\]

Half of the time the increasing and decreasing index are on the same side of \(\frac{n}{2}\), for instance \(x_i < x_{med}\) and \(x_d < x_{med}\), replacement of \(x_d\) by \(x_i\) will not change the median and two comparisons are sufficient to see this.

Even when \(i\) and \(d\) are each side of \(\frac{n}{2}\), the equation will only need one new passage to be done through the data to update the sums. Updating quantiles needs further study.

D. Uses for Exhaustive Bootstrap Distributions

In modern developments of the bootstrap the original data are re-used in complex ways to get more accurate estimates and intervals and indeed to assess the variability of bootstrap intervals themselves. Techniques like double bootstrapping and Jackknife-after-bootstrap are being actively explored. In principle, all of this information is contained in the exhaustive bootstrap. In the present section we spell this out in a simple case: the jackknife after bootstrap as developed in Efron (1992).

The object is to assess the sampling variability of a functional of the bootstrap distribution such as the length of a confidence interval. Efron used the Jackknife idea, considering separately each of the histograms \(H_i, 1 \leq i \leq n\), where \(H_i\) used bootstrap replications which were missing the \(i\)th of the original observations. The functional of interest can be calculated using the usual Jackknife weights. Efron (1992, section 3) contains details and examples.
Efron based his analysis on B bootstrap resamples (B=1,000). The same ideas apply for the exhaustive bootstrap. Here, while running through all compositions, one simply distributes the associated function value (and associated weight) into the n distributions being built up. Of course if a given composition is missing more than one value then it gets used once for each missing value.

4. Examples of the Exhaustive Bootstrap
   
   A. Correlation Coefficient for the Law-School Data

   The 15 data points are as follows:
   
   LSAT : 576 635 558 578 666 580 555 661 652 605 653 575 545 572 594
   GPA :  3.39 3.30 2.81 3.03 3.44 3.07 3.00 3.43 3.36 3.13 3.12 2.74 2.76 2.88 2.96

   Figure 4.1: Scatterplot of Law School Data

   Observation 1 is a potential outlier. To investigate we carried out the exhaustive bootstrap without observation 1. Figure 4.3 shows that the mode is definitely larger (around 0.92) when this outlier is left out; so that the bump occurs because of the translation of the modal class towards one. We also investigated the impact of point 11. Figure 4.2 shows the exhaustive bootstrap without either the points 1 or 11. This gives further evidence that point 1 is responsible for the corner.

   We carried out an exhaustive bootstrap analysis on the first 10 data points from the list above. Here, the effect of the first data point is enormous, creating a hole that is visible in both the exhaustive and the Monte Carlo Bootstrap (Figures 4.5 and 4.6). A final remark that we have to make about the corner is that nothing corresponding to it is visible if we replace the correlation coefficient by it's variance-stabilized Fisher transform (\(z = \arctanh(\rho)\)), as suggested by Hall, Martin and Schucany (1989). (see Figure 4.7)
Figure 4.2: Exhaustive Bootstrap with all n=15 points (right tail)

Figure 4.3: Exhaustive Bootstrap without point 1 (right tail)

Figure 4.4: Exhaustive Bootstrap without point 1 or 11 (right tail)
Figure 4.5: Histogram for Exhaustive Bootstrap: first 10 Law School Points

Figure 4.6: Histogram for Monte Carlo Bootstrap: first 10 Law School Points, (B=10,000)
Figure 4.7: Histogram of Arctanh(\(\tilde{\rho}\))
B. Bio-equivalence data

As described in Efron and Tibshirani (1993) or Efron (1992), these are data on 8 patients for which we have measurements on placebo, an approved and a new drug patch.

The FDA bioequivalence requirement is that a .90 central confidence interval for

\[
\rho = \frac{E(\text{New} - \text{Approved})}{E(\text{Approved} - \text{Placebo})}
\]

to lie within the range \([-0.2, 0.2]\).

The data points are:

<table>
<thead>
<tr>
<th>App-Pla</th>
<th>8406</th>
<th>2342</th>
<th>8187</th>
<th>8459</th>
<th>4795</th>
<th>3516</th>
<th>4796</th>
<th>10238</th>
</tr>
</thead>
<tbody>
<tr>
<td>New-App</td>
<td>-1200</td>
<td>2601</td>
<td>-2705</td>
<td>1982</td>
<td>-1290</td>
<td>351</td>
<td>-638</td>
<td>-2719</td>
</tr>
</tbody>
</table>

![Figure 4.8 Scatterplot of Bioequivalence Data](image)

The exhaustive bootstrap was computed for these data points: Figure 4.9 presents the histogram of the 6438 weighted ratios. Here the exhaustive picture is easier to look at but the conclusions drawn by Efron and Tibshirani (1993) don’t change appreciably.

One can note here that the computation of the exhaustive bootstrap distribution is actually cheaper than doing an ordinary Bootstrap with \(B=4000\) as in Efron and Tibshirani (1993).

This is due to the fact that here we have a function of two means: a complete computation of the statistic has a cost of around \(2n - 1\) "flops" (floating point operations), so for \(B= 4000\) the total cost would be 60,000 flops, an update of the statistic costs 5 "flops", and that of the weights 2 flops, so an exhaustive bootstrap costs 45,066 flops in this case.
Figure 4.9: Exhaustive Bootstrap: Bioequivalence Data

Figure 4.10: Monte Carlo Bootstrap: Bioequivalence Data, B = 4000.
5. Other Gray Codes

In this section we give a summary of other problems in statistics where Gray codes are available. We treat permutation tests, subgroups of the permutation group, and permutations of multisets (and so $m$ out of $n$ bootstrap replications). Finally, we give pointers to the available Gray code technology.

A. Permutation Tests. Let $S_n$ be the group of $n!$ permutations on $n$ letters. Pitman (1937) first investigated the permutation justification of tests like the $t$-test. There are any number of modern variants. Among our favorites are Friedman-Rafsky’s (1979) generalized measure of association: there one observes $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i$ and $y_j$ taking values in metric spaces $X$ and $Y$. Friedman and Rafsky proposed building nearest neighbor graphs such as the minimal spanning tree using the $x_i$ in $x$ space and again using the $y_i$ values in $y$ space. This gives two graphs on the index set $\{1, 2, \ldots, n\}$. Let the number of pairs $\{i, j\}$ which appear as an edge in both graphs. If $T$ is large, points close in $X$ tend to be close in $Y$. One can calibrate the observed value of $T$ using the permutation distribution. There are many other variations: Witztum et al. (1994) give a fascinating application to finding patterns in the Bible!

Classically, Monte Carlo and asymptotics are used to approximate these distributions. There are also other approaches in the work of Pagano and his co-author for linear rank statistics and in the work of Mehta and Patil. For $n$ small (e.g., $n \leq 10$ or 15) Gray codes are also a possibility. Nijenhuis and Wilf (1978) give the standard Gray code for $S_n$ using transpositions. There are also many variants available, see e.g. Miller (1970), Chapter 3 of Wilf (1989) and the references to permutation generation in bibliography of Good (1994).

B. Permutation Groups. Let $G$ be a finite group and $S$ a symmetric set of generators of $G$. Form a graph with vertex set $G$ and an edge from $t$ to $t'$ if $t^{-1}t' \in S$. This is called the Cayley graph of $(G, S)$. A Hamiltonian tour of $(G, S)$ is a sequence $t_1, t_2, \ldots, t_{|G|}$, with no repeated values and $t_i^{-1}t_{i+1} \in S$ for each $i$. This gives a Gray coding of $G$ with $S$ making up the basic steps. As an example, if $G$ is the symmetric group and $S$ the set of transpositions, one gets the problem discussed in section A above.

Gray codes for more general groups arise in several natural problems. Here are three examples: first, one has the sign change group $B_n = S_n \times Z_2^n$. This is the group of $n \times n$ signed permutation matrices. Second, one has the symmetry groups of designed experiments such as analysis of variance. Third, one may consider subgroups of the symmetric group $S_n$ as a way of getting a systematic set of points which are well distributed in $S_n$. These would be useful when $n$ is too large to permit complete enumeration. One would hope to avoid the “holes” of Monte Carlo sampling. As an example, the group $PSL_3(F_p)$ of $3 \times 3$ matrices mod $p$ operates transitively and faithfully on the lines of the vector space, $Z^3_p$. There are $n = p^3 + p$ such lines so one gets a subgroup of $S_n$ of size $(p^3 - 1)(p^2 - 1)p^3/2$. The classical combinatorial design literature presents many further examples.
Investigating how regularly such examples sit in the permutation group seems like a fascinating combinatorial problem.

For any of the examples above one may ask for a Gray code in a natural set of generators which hopefully do not involve too large a change in the permutation representation. While we have not actively tried this out, there is a literature on Hamiltonian paths for Cayley graphs. Entry may be had through Alspach and Godsil (1985) and Ruskey (1994) and the references cited there.

C. Permutations of Multisets. Let $X$ be a multiset with $k$ values labeled 1, $k_2$ values labeled 2, $\ldots$, $k_m$ values labeled $m$. Thus $|X| = k_1 + \cdots + k_m = n$. There are $\binom{n}{k_1 \ldots k_n}$ distinct arrangements of $X$. Such arrangements arise in analysis of variance (for a one way layout). When $m = 2$, an arrangement of $X$ may be regarded as subsets of size $k_1$ out of $n$. These arise in the basic two sample test of Fisher (1936). They also arise from the $m$ out of $n$ bootstraps. See Politis and Romano (1992) for applications and history. Of course when $k_1 = n - 1$, $k_2 = 1$ one has the usual bootstrap of section 3.

Joan Miller (1970) has given an early Gray code for these cases using the set of transpositions. Ko and Ruskey (1992) discuss this problem and give pointers to the computer science literature.

D. General Theory. It is an unfortunate fact of life that while Gray codes usually exist, there is no general recipe for finding them. Indeed, the problem of finding Hamiltonian cycles in general graphs is $N - P$ complete. Nonetheless, with special neat cases or small $n$, one can often find suitable codes. The graph theory approach is usefully laid out in Nijenhuis and Wilf (1978). A survey of a rich variety of special cases can be found in Wilf and in the forthcoming monograph of Ruskey (1994). The cases where such Gray codes are useful in statistical problems involve small $n$ and may be solved by brute force.

Conclusion

We have seen that a careful ‘stepping through’ all bootstrap or randomized samples allows for use of updating procedures and thus enables considerable computational gain; especially when the statistics are reused for correction procedures. However as sample sizes grow, the number of resamples or permutations DOES grow exponentially so another scheme has to be introduced to use updating while still studying the randomization distributions, the idea of only allowing randomness to seep in the changes in the sample through low rank updates leads to doing random walks on the resamples, this is studied in Diaconis and Holmes (1994b).

Acknowledgements

We thank Brad Efron for several inspiring conversations, Tom DiCiccio, Steve Stern and Michael Martin for helpful remarks, and Frank Ruskey, Gene Golub and Ron Graham for bibliographical pointers.

We particularly acknowledge extensive programming help from Brigitte Charnomordic without which nothing would have been possible.
A S Functions for Gray codes

A.1 Gray Codes for subsets

grays_function(n = 5)
{
  #Generates the vector of elements whose signs are to change
  #An example of a randomization test using this follows
  vec <- 0
  for(i in 1:(n + 1)) {
    d <- i
    p <- 1
    while(!as.logical(d %% 2)) {
      d <- d %% 2
      p <- p + 1
    }
    vec <- c(vec, p)
  }
  return(vec[-1])
}

A.2 Randomization Test

pair.gr_function(dif = abs(dife), fn = sum, grayc)
{
  #
  n <- length(dif)
  S <- rep(0, 2^n)
  if(missing(grayc))
    grayc <- grays(2^(n - 1) - 1)
  signs <- rep(1, n)
  S[1] <- fn(dif)
  for(i in 2:(2^(n - 1))) {
    gc <- grayc[i]
    signs[gc] <- ( - signs[gc])
    S[i] <- S[i - 1] + 2 * signs[gc] * dif[gc]
    cat("S :", S[i], "\n")
  }
  S[2^(n - 1) + 1):(2^n) <- ( - S[1:(2^(n - 1))])
  return(S)
}

A.3 Gray Codes for Combinations

gray <- function(x = 0, n = sum(x), k = length(x), point = c(0, 0, 0))
{
  #is.list(x)) {
    point <- x$point
    x <- x$x
  }
}

final <- F  

# 1er tour
if(sum(x) == 0) {
  x <- c(n, rep(0, k - 1))
  point <- c(1, 2, 1)
  return(x, point, final)
}

# c'est fini
if(x[k] == n)
  stop("already last")  
if(point[3] == 1) {
  b <- (1:k)[x > 0][2]  # position du 2ème élément non nul
  if(is.na(b)) b <- 2  # au lieu de 0
  if(b == 2) {
# attention : ne pas changer d si d différent de 1
    if((point[1] == i) && (x[1] == 1)) point <- c(1, 2, 2)
# et ne pas oublier i
    else {
      if(((n - x[1]) %% 2) == 0)
        point <- c(1, 2, 2)
      else if((x[b] %% 2) == 1)
        point <- c(1, b, b)
      else point[1:2] <- c(b, 1)
    }
  }  # fin du cas b==2
}
else {
  # fin du cas p==1
  if(((n - x[point[3]]) %% 2) == 1) {
    point[1:2] <- c(point[3], point[3] - 1)
    if(x[point[3]] %% 2 == 0)
      point[2] <- 1
  }
  else {
    if((x[point[3] + 1] %% 2) == 0) {
      point[1:2] <- c(point[3], point[3] + 1)
      if(x[point[3]] == 1)
    }
    else point[1:2] <- c(point[3] + 1, point[3])
  }
}
x[point[2]] <- x[point[2]] + 1
x[point[1]] <- x[point[1]] - 1
if(x[1] > 0)
  point[3] <- 1
if(x[k] == n)
  final <- T
return(x, point, final)
callgray <- function(n = 5, k = 5, point = c(0, 0, 0), x = 0, pasapas = F)
{
    bi <- matrix(0, nrow = right(n), ncol = n)
    final <- F
    i <- 1
    while(!final) {
        x <- gray(x, n, k, point)
        bi[i, ] <- x$x
        i <- i + 1
        final <- x$final
        if(pasapas) {
            rep <- scan(n = 1)
            if(length(rep) > 0)
                break
        }
    }
    cat("finished...")
    return(bi)
}
B  C program for Gray Code for Combinations

#include <stream.h>
#include <String.h>
#include <stdlib.h>
#include <math.h>

//prototypage
void initab(int *, int, int);
int cgray(int *, int *, int *, int *)
int pos2(int *, int);
inpair(int);
void fixpoint(int *, int, int, int);
void affichetab(int *, int);
in sommetab(int *, int);

// call de la fonction cgray partir de Splus
int cgray(int *x, int *point, int *n, int *k)
{
    // appele :
    // pos2, fixpoint, affichage, initab, sommetab, pair

    int fini, b;
    fini=0;
    //premier tour on appele cette fonction avec x=c(0,0,...0)
    if(sommetab(x, k)==0)
    {
        initab(x, k, 0); //init tableau z ro pour tres x r
        x[0]=n;
        //cout "\npremier vecteur\t";
        affichetab(x, k);
        fixpoint(point, 0, 1, 0); //fixe premier (d, i, p)
        return(0);
    }
    if(x[k-1]==n)
    
    fini=1;
    if(point[2]==0)
    {
        b=pos2(x, k); //indice du second elt non nul
        if(b<=0)
        {
            //cout "erreur"
            return(-1);
        }
        if(b==1)
        {
            if((point[0]==0)&&(x[0]==1)) fixpoint(point, 0, 1, 1);
        }
        else
{  
    if(pair(n-x[0])) fixepoint(point,0,1,1);  
    else if(!pair(x[b])) fixepoint(point,0,b,b);  
    else fixepoint(point,b,0,point[2]);  
}
} // fin p==0
else
{
    if(!pair(n-x[point[2]]))
    {
        fixepoint(point,point[2],point[2]-1,point[2]);
        if(pair(x[point[2]])) point[1]=0;
        point[2]=point[1];
    }
    else
    {
        if(pair(x[point[2]+1]))
        {
            fixepoint(point,point[2],point[2]+1,point[2]);
            if(x[point[2]]==1) point[2]++;
        }
        else fixepoint(point,point[2]+1,point[2],point[2]);
    }
    // fin du boulot
    x[point[1]]++;
    x[point[0]]--;
    if(x[0]>0) point[2]=0;
    if(x[k-1]==n)
    {  
        fini=1;
        affichetab(x,k); //affichage
        return(fini);
    }
}
References


[42] Pitman, E. (1937). Significance tests which may be applied to samples from any population. JRSS B. 4 119–130.