Pergamon

Comput. Biol. Med. Vol. 26, No. 1, pp. 41-52, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0010-4825/96 $15.00 + 0.00

0010-4825(95)00045-3

RANDOM SELECTION ALGORITHMS FOR SPATIAL AND TEMPORAL SAMPLING

JOHN A. BYERS

Department of Plant Protection, Swedish University of Agricultural Sciences, S-230 53 Alnarp, Sweden

(*Received* 6 *March* 1995; *received in revised form* 31 *August* 1995)

Abstract—Seven BASIC programs are presented that use algorithms for selection of treatments and samples in spatial and temporal contexts. Program (1) takes a natural sequence of samples (such as logs cut from a tree trunk) and divides them into groups (equal to the number of samples divided by treatments), and then selects non-redundantly from each group a sample at random for each treatment. Program (2) matches items from different categories equally to any number of treatments in proportion to the numbers of items of each category. Program (3) selects sampling times or segment lengths of specified interval and number from within a time period or perimeter distance. These samples can be spaced apart by at least a specified amount of time or distance but otherwise are chosen at random. In program (4), a series of sample coordinates (x, y) are chosen at random from a rectangular area so that no points are closer than a specified minimum distance to any other. For each of the sample points, the Cartesian and polar coordinates are given. Program (5) generates any possible Latin square, while program (6) generates Latin cubes, and program (7) makes Graeco–Latin cubes. Examples of program use and output are presented for experiments with bark beetles (Coleoptera: Scolytidae) responding to pheromone blends and colonizing host trees.

BASIC Algorithms Randomization Sampling Experimental design Pheromone

INTRODUCTION

There are numerous software programs and statistical books to aid researchers in their analysis of experimental data. However, relatively few of these textbooks and programs offer practical methods for randomization of treatments and sampling during the data gathering stages. Statistical analysis methods for experimental and natural observations presume that the selections of samples and treatments are unbiased [1–9]. The personal computer is ideally suited for selection of random numbers and their application to experimental design. Recently, five algorithms were implemented on the personal computer for use in randomization of treatments and for sampling at random [10]. The computer-programmed algorithms in the BASIC language allow construction of Latin squares (treatments represented only once in each row and column) and quasi–Latin squares (equal numbers of replicated treatments for each row and column). In addition, random placement of equal numbers of different treatments within a grid can be done in such a way so that identical treatments are separated both horizontally and vertically from one another by other treatments (spacing of treatments).

In a second paper [11], this algorithm was improved to include horizontal, vertical and diagonal separation. Other algorithms can place treatments in grids using unequal numbers of replicates of several treatment types (including no treatment), with or without a spacing function. Algorithms for generation of Latin squares with randomized non-repeated subscripts and Graeco–Latin squares were also presented. These algorithms comprised methods to randomize integers in experimental designs where treatments, positions, and time periods must be selected.

In this paper additional algorithms are presented for selection of treatments or samples in spatial and temporal contexts. One objective is to devise algorithms that sample time or perimeter distance with a random or variable degree of uniformity. Another objective is to construct a method to sample areal plots with variable uniformity. Finally, programmed algorithms are described that can randomly select any of all possible Latin squares (not just a sub-set as earlier [10]) as well as three-dimensional Latin cubes and Graeco–Latin cubes.

METHODS

The seven programs are coded in the QuickBASIC language (version 4.5, Microsoft Corp.) and the output is based on the pseudo-random number generator command, RND, which repeats its sequence every 16,777,214 steps [11]. The BASIC language code is also compatible with BASICA (older version of BASIC) and QBASIC (packaged with DOS operating systems). A color monitor is recommended to take advantage of the COLOR commands. Large Latin cubes and Graeco–Latin cubes, programs (6) and (7), require two- and three-dimensional arrays that for 24 treatments require slightly less than 64K bytes of memory. QuickBASIC will allow arrays larger than 64K only by starting the program with the **QB/AH** command (for *Arrays Huge*). This would allow up to about 38 treatments before running out of the 640K conventional memory, however, only 24 treatments will print on the screen. Also construction of squares or cubes with more than 24 treatments becomes prohibitively slow. Program output can be printed directly by pressing the [Print Screen] key with a printer connected, or by adding **LPRINT** commands, or by opening a sequential file in BASIC and using the **PRINT #1** command. Sequential files can be imported into a word processor as a text file.

RESULTS AND DISCUSSION

Program 1: *Match sequential samples evenly to treatments*

The program (Fig. 1A) takes a sequence of numbers representing a series of samples and divides them into contiguous groups each containing the same number of samples as

A.

```
10 CLS : COLOR 15: PRINT "MATCH SEQUENTIAL SAMPLES EVENLY TO TREATMENTS"
20 COLOR 11: INPUT "ENTER NUMBER OF SAMPLES IN SEQUENCE"; A
30 INPUT "ENTER NUMBER OF TREATMENTS TO MATCH SAMPLES WITH"; B
40 INPUT "ENTER NUMBER FOR RANDOM SEED"; RN: RN = RND(-RN)
50 IF A / B / 2 <> INT(A / B) / 2 THEN PRINT "NOT DIVISIBLE": GOTO 20
60 DIM S(B, A / B): COLOR 15
70 FOR W = 0 TO A / B - 1: FOR Z = 1 TO B
80 R = INT(RND * B) + B * W + 1
90 FOR Q = 1 TO Z - 1: IF S(Q, W + 1) = R THEN NG = 1
100 NEXT Q: IF NG = 1 THEN NG = 0: GOTO 80
110 S(Z, W + 1) = R: NEXT Z: NEXT W
120 FOR W = 1 TO B: PRINT "TREATMENT"; W; ":";
130 FOR Q = 1 TO A / B: PRINT S(W, Q);: NEXT: PRINT : NEXT
```

B.

```
MATCH SEQUENTIAL SAMPLES EVENLY TO TREATMENTS
ENTER NUMBER OF SAMPLES IN SEQUENCE? 20
ENTER NUMBER OF TREATMENTS TO MATCH SAMPLES WITH? 4
ENTER NUMBER FOR RANDOM SEED? 1

TREATMENT 1 : 1  6  11  15  19
TREATMENT 2 : 3  8  12  16  18
TREATMENT 3 : 2  7  9  13  17
TREATMENT 4 : 4  5  10  14  20
```

Fig. 1. (A) Listing of BASIC program used to randomly assign sequential samples evenly to treatments. (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

the desired number of different treatments. Then the samples of each group are matched at random to each of the different treatments. For example, to see which of four bark beetle species produce the most progeny in logs cut from a Norway spruce tree, one would want to balance as much as possible the bark areas and thickness among the species. Reducing this source of variation could be done by cutting 20 logs from along the trunk (numbered 1 to 20) and dividing them into five contiguous groups of four logs each and uniquely assigning at random each log within a group to one of the four species treatments so that all treatments are represented within each group (i.e. each species colonizes similar areas and regions of the tree).

One enters the number of samples (20), the number of different treatments (4), and a number which serves as the random seed. The random seed number will determine the starting point in the sequence of random numbers and thus it should not be the same if the experiment is repeated. Otherwise, any number for the random seed is as likely to give results as random as any other. For each group of samples (five groups) the program picks at random a number representing a treatment and matches this to the samples in sequence within the group. However, the program first checks to see that none of the preceding samples within the group have the same treatment, if so, then another random number is picked until a unique treatment is found. Output of the program for four treatments and 20 sample positions (1–20) is shown in Fig. 1B. As can be seen, the four bark beetle species would be fairly evenly distributed along the trunk in various logs of similar areas.

Program 2: *Select items from categories for treatments*

This program divides sample items K from one or more populations J into N treatment groups (Fig. 2A). For example, suppose that four species of bark beetle ($J=4$) with $K_J=35$, 42, 18 and 29 individuals, respectively, are to be evenly divided at random among four treatment groups ($N=4$) so that individuals within a treatment can be injected with a different chemical dose in a known sequence (within groups or for all groups, although sequential information can be ignored).

One enters the number of different kinds of items, or species in this case (4). Then the number of items (or individuals) for each kind of item (35, 42, 18 and 29) is entered. The number of treatments (4) and a random seed is entered. A two-dimensional array holds the sequences of numbers for each kind of item and the number of each item. The program picks at random numbers from this array and places them non-redundantly into the treatment groups. Output of the program for four treatment groups (1–4) with item types (A–D) and the sequential numbers (which may or may not be used) is shown in Fig. 2B.

Program 3: *Time or perimeter sampling with uniformity*

This program (Fig. 3A) will list a series of starting and stopping times (or distances) chosen at random for observations of a specified length distributed over any period of time (or perimeter distance). The number of samples and their duration or length are specified by the user. The uniformity of the sampling distribution can be increased by specifying a minimum time or distance allowed between samples. A check is made by the program to see if the above parameters will allow the desired spacing of the samples.

To sample the behavior of an insect over a period of time at random but with varying degrees of uniformity, one enters the number of time units (seconds or minutes) during the period (120), the duration of a sampling (5), the minimum number of time units between any samples (5), and the total number of samples (8), as well as a random seed. The user also is asked whether time or perimeter distance will be considered. In the case of time, the starting time (e.g. 12) is specified to aid understanding. As can be seen in Fig. 3B, the observation times are uniformly spaced throughout the duration of the observation period and are sorted in ascending order for convenience. The same procedure can be used to determine the places along a perimeter where sampling can be done of specified length and minimum distance between samples. The advantage of uniform random sampling compared to periodic sampling (the usual case in biological experiments) is that there is less possibility of the sampling coinciding with behavioral periodicities and thereby biasing results. Also, behavioral periodicities can more likely be detected with a uniform random sampling than with a periodic sampling, in which the latter might coincide only with peaks of activity.

Program 4: *Spaced point sampling within rectangular area*

This program (Fig. 4A) selects sample points of x, y coordinates at random within a rectangular area of any size. The points can be increasingly uniformly distributed by not allowing the selection of coordinates that are closer than a minimum distance to any previously placed points. One enters the lengths of the X-axis (100) and Y-axis (100), the number of points to select coordinates for (6), and the minimum spacing between sample points (20). The maximum hexagonal spacing that a certain number of points N in an area ($AREA$) can be spaced is given by $1.0746/\sqrt{N/AREA}$ [12], and is 43.9 units for the parameters above. Usually, the minimum spacing distance used must be about 70% or

A.

```
10 CLS : COLOR 15: PRINT "SELECT ITEMS FROM CATEGORIES FOR TREATMENTS"
20 COLOR 11: DEFINT A-Z: INPUT "ENTER NUMBER OF DIFFERENT CATEGORIES"; J
30 DIM I(J, 200): DIM N(J): DIM NT(J)
40 FOR K = 1 TO J
50 PRINT "ENTER NUMBER OF ITEMS "; CHR$(64 + K); : INPUT N(K)
60 NT(K) = N(K): NEXT K
70 INPUT "ENTER NUMBER OF TREATMENT GROUPS TO PLACE ITEMS"; G
80 INPUT "ENTER NUMBER FOR RANDOM SEED"; RN: RN = RND(-RN)
90 FOR H = 1 TO J: FOR K = 1 TO N(H): I(H, K) = K: NEXT: NEXT
100 FOR W = 1 TO G: COLOR 15: PRINT : PRINT "TREATMENT"; W; ":";
110 FOR ITEM = 1 TO J: COLOR ITEM MOD 7 + 1: FOR H = 1 TO NT(ITEM) \ G
120 X = INT(RND * N(ITEM) + 1): PRINT CHR$(64 + ITEM);
130 PRINT MID$(STR$(I(ITEM, X)), 2); " "; : N(ITEM) = N(ITEM) - 1
140 FOR K = X TO N(ITEM): I(ITEM, K) = I(ITEM, K + 1): NEXT K: NEXT H
150 NEXT ITEM: NEXT W
```

B.

```
SELECT ITEMS FROM CATEGORIES FOR TREATMENTS
ENTER NUMBER OF DIFFERENT CATEGORIES? 4
ENTER NUMBER OF ITEMS A? 35
ENTER NUMBER OF ITEMS B? 42
ENTER NUMBER OF ITEMS C? 18
ENTER NUMBER OF ITEMS D? 29
ENTER NUMBER OF TREATMENT GROUPS TO PLACE ITEMS? 4
ENTER NUMBER FOR RANDOM SEED? 1

TREATMENT 1 :A2 A4 A8 A9 A22 A20 A1 A17 B29 B7 B21 B4 B35 B19 B39 B14 B34 B26 C
18 C8 C13 C7 D2 D20 D24 D27 D25 D5 D29
TREATMENT 2 :A5 A24 A30 A13 A26 A29 A35 A28 B8 B3 B42 B27 B9 B1 B2 B17 B31 B24 C
4 C10 C1 C12 D8 D11 D26 D10 D17 D7 D15
TREATMENT 3 :A10 A33 A21 A7 A12 A25 A23 A3 B40 B6 B15 B12 B20 B11 B10 B36 B38 B
13 C15 C14 C3 C5 D12 D1 D16 D9 D21 D22 D23
TREATMENT 4 :A15 A32 A34 A6 A14 A31 A18 A16 B22 B23 B37 B16 B25 B33 B30 B32 B5 B
18 C11 C9 C2 C16 D13 D28 D14 D6 D4 D3 D18
```

Fig. 2. (A) Listing of BASIC program used to randomly assign samples from each of several categories evenly to several treatments. (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

A.

```
10 CLS : COLOR 15: PRINT "TIME OR PERIMETER SAMPLING WITH UNIFORMITY"
20 COLOR 11: INPUT "ENTER TOTAL UNITS IN TIME PERIOD OR PERIMETER"; TU
30 INPUT "ENTER SAMPLE LENGTH IN UNITS"; SP
40 INPUT "ENTER MINIMUM NUMBER OF UNITS BETWEEN SAMPLES"; ML
50 INPUT "ENTER NUMBER OF SAMPLES"; N
60 IF N * SP + (N - 1) * ML > TU * .7 THEN GOTO 50
70 INPUT "SAMPLING: ENTER 0 FOR TIME, 1 FOR PERIMETER"; TS
80 IF TS = 0 THEN INPUT "ENTER BEGINNING HOUR OR MINUTE"; BH
90 INPUT "ENTER NUMBER FOR RANDOM SEED"; S: S = RND(-S): DIM TM(N)
100 COLOR 15: IF TS = 0 THEN PRINT "TIMES: "; ELSE PRINT "DISTANCES: ";
110 FOR W = 1 TO N
120 R = INT(RND * (TU - SP))
130 FOR Q = 1 TO W - 1
140 IF R > TM(Q) - SP - ML AND R < TM(Q) + SP + ML THEN NG = 1
150 NEXT: IF NG = 1 THEN NG = 0: GOTO 120
160 TM(W) = R: NEXT: FOR W = 1 TO N: SM = W: FOR J = W + 1 TO N
170 IF TM(J) < TM(SM) THEN SM = J
180 NEXT: IF SM > W THEN SWAP TM(W), TM(SM)
190 NEXT: FOR W = 1 TO N: IF TS = 0 THEN GOTO 210
200 PRINT STR$(TM(W)); "-"; MID$(STR$(TM(W) + SP), 2); " "; : GOTO 280
210 PRINT CHR$(32); STR$((BH * 60 + TM(W)) \ 60); ":";
220 J$ = MID$(STR$((BH * 60 + TM(W)) MOD 60), 2): GOSUB 260
230 PRINT J$; "-"; MID$(STR$((BH * 60 + TM(W) + SP) \ 60), 2); ":";
240 J$ = MID$(STR$((BH * 60 + TM(W) + SP) MOD 60), 2): GOSUB 260
250 PRINT J$; : GOTO 280
260 IF LEN(J$) = 0 THEN J$ = "00"
270 IF LEN(J$) = 1 THEN J$ = "0" + J$: RETURN ELSE RETURN
280 NEXT
```

B.

```
TIME OR PERIMETER SAMPLING WITH UNIFORMITY
ENTER TOTAL UNITS IN TIME PERIOD OR PERIMETER? 180
ENTER SAMPLE LENGTH IN UNITS? 5
ENTER MINIMUM NUMBER OF UNITS BETWEEN SAMPLES? 10
ENTER NUMBER OF SAMPLES? 15
SAMPLING: ENTER 0 FOR TIME, 1 FOR PERIMETER? 0
ENTER BEGINNING HOUR OR MINUTE? 12
ENTER NUMBER FOR RANDOM SEED? 1

TIMES: 12:04-12:09 12:18-12:23 12:36-12:41 12:46-12:51 13:05-13:10 13:17-
13:22 13:32-13:37 13:44-13:49
```

Fig. 3. (A) Listing of BASIC program used to uniformly sample at random over a time period a set number of samples of specified duration and minimum time between samples beginning at a specific time. Distance of a perimeter can replace time. (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

less of the maximum spacing distance theoretically possible in order for the program to finish spacing points [13]. In addition to the *x*, *y* coordinates, the polar coordinates are given by a distance from the origin (lower left corner of the area) and an angle (from the *X*-axis). The distance from the origin to each point is calculated with the Pythagorean theorem and the angle is given from BASIC by **ATN(Y/X))*180/3.1415926#** (see Fig. 4B). The spacing of sample points may be desired if sampling disturbs the surrounding area such that it is undesirable to sample nearby as could occur with a truly random selection. Also, it is less likely that a clumped sampling distribution could occur by chance and bias the conclusions. These ideas are embodied in stratified sampling methods [14].

Program 5: Any possible Latin square

A Latin square of five treatment numbers on a side has five rows and five columns of numbers (1 to 5) in which each number occurs only once in each row and column and all treatment numbers are represented in each row and column, yielding 25 cells with numbers. Algorithms for rapid construction of Latin squares of any size are given in

A.

```
10 CLS : COLOR 15: PRINT "SPACED POINT SAMPLING WITHIN RECTANGULAR AREA"
20 COLOR 11: INPUT "ENTER X-AXIS LENGTH"; XL
30 INPUT "ENTER Y-AXIS LENGTH"; YL: A = XL * YL
40 INPUT "ENTER NUMBER OF SAMPLE POINTS"; N
50 INPUT "ENTER MINIMUM SPACING BETWEEN SAMPLE POINTS"; MAD
60 MAX = 1.0746 / SQR(N / A) * .7
70 IF MAD > MAX THEN PRINT "SPACING TOO LARGE": GOTO 40
80 INPUT "ENTER NUMBER FOR RANDOM SEED"; RN: RN = RND(-RN)
90 INPUT "ENTER A FILENAME FOR POINTS, OR <ENTER>"; A$
100 IF LEN(A$) <> 0 THEN OPEN A$ FOR OUTPUT AS #1
110 DIM X(N), Y(N), R(N), ANG(N): COLOR 15
120 FOR W = 1 TO N
130 X(W) = RND * XL: Y(W) = RND * YL
140 FOR Q = 1 TO W - 1
150 IF X(W) < X(Q) - MAD OR X(W) > X(Q) + MAD THEN GOTO 200
160 IF Y(W) < Y(Q) - MAD OR Y(W) > Y(Q) + MAD THEN GOTO 200
170 X = X(Q) - X(W): Y = Y(Q) - Y(W)
180 IF SQR(X * X + Y * Y) > MAD THEN GOTO 200
190 Z = 1: REM EXIT FOR 'EXIT FOR in QuickBASIC
200 NEXT Q: IF Z = 1 THEN Z = 0: GOTO 130
210 R(W) = SQR(X(W) * X(W) + Y(W) * Y(W))
220 ANG(W) = ATN(Y(W) / X(W)) * 180 / 3.1415926#
230 PRINT W, "X="; X(W), "Y="; Y(W), "R="; R(W), "ANG="; ANG(W)
240 IF LEN(A$) <> 0 THEN GOTO 250 ELSE GOTO 260
250 PRINT #1, W, "X="; X(W), "Y="; Y(W), "R="; R(W), "ANG="; ANG(W)
260 NEXT W: CLOSE
```

B.

```
SPACED POINT SAMPLING WITHIN RECTANGULAR AREA
ENTER X-AXIS LENGTH? 100
ENTER Y-AXIS LENGTH? 100
ENTER NUMBER OF SAMPLE POINTS? 6
ENTER MINIMUM SPACING BETWEEN SAMPLE POINTS? 20
ENTER NUMBER FOR RANDOM SEED? 1
ENTER A FILENAME FOR POINTS, OR <ENTER>? PTS

1      X= 3.584582  Y= 8.635235  R= 9.349679  ANG= 67.45612
2      X= 57.32005  Y= 51.06652  R= 76.76834  ANG= 41.69789
3      X= 3.219455  Y= 40.33364  R= 40.46193  ANG= 85.43629
4      X= 67.02209  Y= 16.61372  R= 69.05054  ANG= 13.92209
5      X= 47.61788  Y= 9.018242  R= 48.46433  ANG= 10.7241
6      X= 80.32864  Y= 43.61303  R= 91.40452  ANG= 28.49898
```

Fig. 4. (A) Listing of BASIC program used to randomly select x, y coordinates of points for sample centers within any rectangular area. The points can be spaced apart a minimum distance from 0 (completely random) to about 70% of the maximum hexagonal spacing possible (see text for details). The polar coordinates are also given. (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

A.

```
10 CLS : DEFINT A-Z: COLOR 15: PRINT "ANY POSSIBLE LATIN SQUARE"
20 COLOR 11: INPUT "ENTER NUMBER OF TREATMENTS"; N
30 INPUT "ENTER NUMBER FOR RANDOM SEED"; RN: RN = RND(-RN)
40 COLOR 15: DIM C(N): DIM A(N, N)
50 FOR R = 1 TO N:
60 FOR C = 1 TO N: C(C) = C: NEXT C: J = N
70 FOR C = 1 TO N: I = 0
80 X = INT(RND * J + 1)
90 REM CHECK COLUMN IF OK (ROW IS INHERENTLY OK)
100 FOR H = 1 TO R - 1
110 IF I > 50 THEN 60: REM ROW NO GOOD
120 IF A(H, C) = C(X) THEN I = I + 1: GOTO 80: REM COLUMN NO GOOD
130 NEXT H
140 A(R, C) = C(X): J = J - 1
150 FOR K = X TO J: C(K) = C(K + 1): NEXT K: NEXT C
160 FOR CL = 1 TO N: PRINT USING "###"; A(R, CL);: NEXT: PRINT : NEXT
```

B.

```
ANY POSSIBLE LATIN SQUARE
ENTER NUMBER OF TREATMENTS? 8
ENTER NUMBER FOR RANDOM SEED? 1

 1 2 3 4 7 6 5 8
 6 4 1 8 3 7 2 5
 5 8 6 3 1 2 7 4
 7 5 8 1 2 4 3 6
 3 6 5 2 4 1 8 7
 4 3 7 6 5 8 1 2
 8 7 2 5 6 3 4 1
 2 1 4 7 8 5 6 3
```

Fig. 5. (A) Listing of BASIC program used to randomly select any possible Latin square (each treatment represented once in each column and row). (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

Program 2 of Byers [10]. This method uses a randomized column and row array (each equal to the number of treatments) whose intersection sums give the value for each cell. However, if the sum is larger than the number of different treatments then this number is subtracted from the sum [10, 15, 16]. Although fast, this method is not capable of generating all possible Latin squares since only $N!(N-1)!$ squares can be obtained, or 2880 for a 5×5 square [9]. In fact, for 5×5 Latin squares there are 161,280 different possible permutations [4, 8].

The program (Fig. 5A) used in a 486-type computer can make Latin squares of up to about 20 cells on a side in a few seconds, and in theory any of the millions of possible squares can be obtained. The algorithm picks at random a treatment number for a cell in a row from a temporary row array such that this number is no longer available for selection, but first cells in the same column from previously generated rows are checked for redundancy, if such is found then the number is kept in the temporary array and another number is tried until no redundancies are found, whereupon the program moves to the next cell in the row, or a new row. It is possible that no number remaining in the temporary array can satisfy the conditions so after 50 tries at random, the entire row is erased and another random sequence is tried until satisfying the conditions of a Latin square.

One enters the number of treatments (8) and a random seed. The progress of the calculations are shown row by row, the first rows are of course generated rapidly, while for larger squares of 15 or more cells per side, the last rows take increasingly longer times to find solutions. Output of the program is shown in Fig. 5B. One could imagine a test in which the unequal effects of area and time on attraction of bark beetles to eight chemical baits are evened by testing each bait once for a day in each of eight areas (columns) for a period of 8 days (rows).

Program 6: *Latin cube*

A Latin cube is similar to a Latin square, but a cube of cells exists in which each treatment occurs only once in each horizontal row (x) and column (z) and as well as in each vertical column (y), and all treatments (equal to the number of cells along any edge) occur in each of the three dimensions (x, y, z) connecting any cells. Computers probably can not generate a Latin cube purely at random unless they are incredibly lucky since thousands of constraints are imposed. However, my program (Fig. 6A) can generate many different Latin cubes at random starting from a well-randomized Latin square. After the first Latin square is made as in program 5 above, the remaining levels forming the cube (consisting of Latin squares) are made by adding 1 to each cell number in the Latin square immediately beneath. However, if the sum is larger than the number of treatments (N), then N is subtracted from this sum. In this way Latin squares are made in sequence from the bottom one and at the same time a Latin cube of stacked Latin squares is made. So far, this Latin cube does not appear random, but by randomizing the initial order of the levels of Latin squares, the conditions of a Latin cube are maintained (since any vertical order of treatment numbers is allowed).

There are 120 different ways to order five levels of Latin squares in a $5 \times 5 \times 5$ Latin cube (permutations of N things taken N at a time $= N!$). Thus, for a 5×5 Latin square there are 161,280 permutations [9] which multiplied by 120 ways to order 5 squares yields over 19 million possible Latin cubes with the algorithm here. The number of ways to order 20 (20×20) Latin squares in a cube is 2.43×10^{18}, and there are innumerable possibilities for a Latin square of 20 treatments. For these larger Latin cubes, the numbers of possibilities are clearly phenomenal.

One enters the number of treatments (6) and a random seed. Output of the program is shown in Fig. 5B with each level representing Latin squares in the cube. One could imagine a test in which a species of bark beetle would be allowed to colonize logs of six species of tree (rows) at six different temperatures (columns) at six different densities per log (treatment numbers). Each of the factors: tree species, colonization density, and temperature would occur once for all possible combinations to reduce sources of variation and determine the reproductive rates under all these conditions.

Program 7: *Graeco–Latin cube*

Before one can envision a Graeco–Latin cube, a definition of a Graeco–Latin square is needed. This type of square is one in which every Latin letter and every Greek letter occurs once in each row and once in each column. In addition, each Latin letter and Greek letter (represented here by numbers) occur only once together in the square. Thus, for a Graeco–Latin square of 5×5 there are 25 combinations of Latin letters and numbers which can be placed in 25 positions. This means that there are 25! (about 10^{25}) permutations not all of which are Graeco–Latin squares. Graeco–Latin squares of a side of 4 or odd numbers can be made with program 6 in Byers [11], while it appears impossible to make such squares with even numbered sides larger than 4 [9, 11].

The Graeco–Latin algorithm (lines 10–230) [11] is used to make the bottom square, which then forms the basis for making upper-story squares in a way similar to the Latin cube in program 6 above. However, the levels (each a Graeco–Latin square) are constructed and their order randomized in the same step (lines 300–330) to avoid storage in 3-D string arrays that may not be possible due to memory constraints. Actually, while each horizontal row and column, and each vertical column in the Graeco–Latin cube have each letter and number, and each horizontal square has all unique letter–number

A.

```
10 CLS : COLOR 15: DEFINT A-Z: PRINT "LATIN CUBE"
20 COLOR 11: INPUT "ENTER NUMBER OF TREATMENTS"; N: IF N > 24 THEN 20
30 INPUT "ENTER NUMBER FOR RANDOM SEED"; RN: RN = RND(-RN)
40 COLOR 31: DIM C(N): DIM Q(N, N, N): PRINT "WORKING...": COLOR 15
50 FOR R = 1 TO N
60 FOR C = 1 TO N: C(C) = C: NEXT C: J = N
70 FOR C = 1 TO N: I = 0
80 X = INT(RND * J + 1): REM CHECK IF COLUMN OK (ROW IS INHERENTLY OK)
90 FOR H = 1 TO R - 1: REM CHECK COLUMN (ROW IS OK)
100 IF I > 50 THEN 60: REM ROW NO GOOD
110 IF Q(1, H, C) = C(X) THEN I = I + 1: GOTO 80: REM COLUMN NO GOOD
120 NEXT H: Q(1, R, C) = C(X): J = J - 1
130 FOR K = X TO J: C(K) = C(K + 1): NEXT K: NEXT C
140 NEXT R: REM MAKE OTHER LATIN SQUARE LEVELS OF CUBE
150 FOR V = 2 TO N: FOR R = 1 TO N: FOR C = 1 TO N
160 Q(V, R, C) = Q(1, R, C) + V - 1
170 IF Q(V, R, C) > N THEN Q(V, R, C) = Q(V, R, C) - N
180 NEXT C: NEXT R: NEXT V: REM NOW RANDOMIZE LEVELS
190 FOR T = 1 TO N
200 X = INT(RND * N + 1): Y = INT(RND * N + 1): IF X = Y THEN 200
210 FOR R = 1 TO N: FOR C = 1 TO N
220 SWAP Q(X, R, C), Q(Y, R, C): NEXT: NEXT: NEXT: CLS
230 Z = INT(70 / (N * 3)): IF Z = 0 THEN Z = 1
240 FOR V = 1 TO N: COLOR V MOD 7 + 9
250 FOR R = 1 TO N: FOR C = 1 TO N
260 LOCATE R + E, G * N * 3 + (C - 1) * 3 + 1 + G * 2
270 PRINT USING "###"; Q(V, R, C);: NEXT C: NEXT R
280 LOCATE R + E, G * N * 3 + 2 + G * 2
290 PRINT "LEVEL"; V; : INPUT ; Q$: IF V >= N THEN 310 ELSE G = G + 1
300 IF G = Z THEN G = 0: E = E + N + 1: IF E > 24 - N THEN E = 0: CLS
310 NEXT V
```

B.

```
LATIN CUBE
ENTER NUMBER OF TREATMENTS? 6
ENTER NUMBER FOR RANDOM SEED? 1

 6 1 2 3 5 4   3 4 5 6 2 1   1 2 3 4 6 5
 2 4 6 1 3 5   5 1 3 4 6 2   3 5 1 2 4 6
 1 5 3 4 6 2   4 2 6 1 3 5   2 6 4 5 1 3
 4 6 5 2 1 3   1 3 2 5 4 6   5 1 6 3 2 4
 5 3 4 6 2 1   2 6 1 3 5 4   6 4 5 1 3 2
 3 2 1 5 4 6   6 5 4 2 1 3   4 3 2 6 5 1
 LEVEL 1 ?     LEVEL 2 ?     LEVEL 3 ?
 5 6 1 2 4 3   4 5 6 1 3 2   2 3 4 5 1 6
 1 3 5 6 2 4   6 2 4 5 1 3   4 6 2 3 5 1
 6 4 2 3 5 1   5 3 1 2 4 6   3 1 5 6 2 4
 3 5 4 1 6 2   2 4 3 6 5 1   6 2 1 4 3 5
 4 2 3 5 1 6   3 1 2 4 6 5   1 5 6 2 4 3
 2 1 6 4 3 5   1 6 5 3 2 4   5 4 3 1 6 2
 LEVEL 4 ?     LEVEL 5 ?     LEVEL 6 ?
```

Fig. 6. (A) Listing of BASIC program used to randomly select Latin cubes (each treatment represented once in each horizontal column, horizontal row, and vertical column). (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

combinations, as does each vertical-row square, the vertical-column squares do not have all the unique combinations in each square. Thus, a Graeco–Latin cube is not as "symmetrical" as a Latin cube.

A second kind of Graeco–Latin cube (type II) can be made by inserting the following line into the program in Fig. 7A: **295 X=C+(V−1): IF X>N THEN X=X−1** as indicated and replacing **C** in lines 300–330 with **X**. The cube constructed after these program changes is similar to type I except that each vertical column does not necessarily have each letter or number and vertical-row squares are not Graeco–Latin, however, each vertical-row or vertical-column square has all the letter-number combinations even though they are not Graeco–Latin squares.

One uses the program by entering the number of desired cells per side (5) and a random seed (output shown in Fig. 7B). The number of different possible Graeco–Latin cubes (type I or II) can be determined by the number of Graeco–Latin squares possible. For example, for 5 cells per side there are 101 possible squares [11] and 120 ways to layer the squares (discussed in program 6) to form a cube, or a total of 12,120 different $5 \times 5 \times 5$ Graeco–Latin cubes. For 7 cells/side there are 442 squares [11] and 5040 ways to layer 7 squares, making a possible total of 2,227,680 different cubes of each type. Again, the number of possibilities for larger cubes becomes enormous.

It can be supposed that here one wanted to test the attraction of bark beetles to volatiles from logs cut from five (A–E) host tree species in 2-way combinations with five (1–5) release rates of ethanol (making 25 baits) in five positions (columns) at five forest regions (rows) on each of 5 days (levels). Many bark beetle species are attracted to host volatiles (mainly monoterpenes) and ethanol (indicating microbial decay and a susceptible host tree) [17, 18]. As seen in Fig. 7B, each region on a particular day has each of the (A–E) host species logs combined with each of ethanol release rates (1–5), and all 25 baits are tested each day. Over the 5 day period, each region tests all 25 baits only once, furthermore, each position (25 physical locations) never tests the same (A–E) host species logs or (1–5) ethanol release rates on successive days. This might be important to ensure that natural variation in flight activities between areas does not bias the catches on baits less evenly distributed, as might occur in a randomized block design.

Several texts discuss the advantages and drawbacks of using Latin and Graeco–Latin squares as well as the statistical analysis of variance [1, 3, 4, 6, 8, 9]. Latin squares are significantly more efficient or powerful than randomized block designs when using the same number of samples [3, 6, 9]. The experimental designs presented here that employ a degree of uniform spacing among treatments are more appropriate for use than traditional statistical analyses only when there are logical or pragmatic reasons, as in the examples above. Thus it seems advisable to consult a statistician when considering these specialized designs. A compiled program version is available from the author (send a formatted disk and return mailer or $5 for shipping).

SUMMARY

Seven algorithms implemented in the BASIC computer language are presented for use in randomizing treatments and/or sampling during the design and implementation of experiments. Two programs place ordered items of various numbers into groups for treatments in a uniform or balanced way. The third program designates sample time periods of specific durations over a given period with a minimum time between samples possible. This program can also determine sample distances along a perimeter. The fourth program specifies sample points (x, y coordinates) with or without a degree of separation between sample centers. A Latin square algorithm can generate all possible permutations for Latin squares of less than about 24 treatments per side. Latin cubes are built from this algorithm to generate a Latin square from which all upper story squares are made after which these levels are randomized. Graeco–Latin cubes are generated with a previous algorithm in combination with the randomization of layered squares, although not all possible cubes of this type can be made. However, a sufficiently large number are generated so that no experimental bias should result.

A.

```
10 CLS : COLOR 15: PRINT "GRAECO-LATIN CUBE": DEFINT A-Z: COLOR 11
20 INPUT "ENTER 4 OR AN ODD NUMBER OF TREATMENTS"; N: R = N
30 IF N < 3 OR (N <> 4 AND INT(N / 2) = N / 2) OR N > 19 THEN 20
40 PRINT "ENTER A SPECIFIC NUMBER FOR A SPECIFIC RANDOM";
50 INPUT " SEQUENCE"; RN: RN = RND(-RN)
60 REM FIND ONE SOLUTION TO A GRAECO-LATIN SQUARE OF SIDE N
70 DIM A(N, N): DIM B(N, N): IF N = 4 THEN 120
80 FOR R = 1 TO N: FOR C = 1 TO N: P = P + 1: A(R, C) = P
90 B(R, N + 1 - C) = P: IF C = N THEN P = P - 1
100 IF P = N THEN P = 0
110 NEXT C: NEXT R: GOTO 160: REM DATA FOR SIDE OF 4
120 DATA 1,1,2,2,3,3,4,4,2,4,1,3,4,2,3,1
130 DATA 3,2,4,1,1,4,2,3,4,3,3,4,2,1,1,2
140 FOR R = 1 TO N: FOR C = 1 TO N: READ A(R, C), B(R, C)
150 NEXT C: NEXT R
160 FOR W = 1 TO N: REM RANDOMIZE ROWS & COLUMNS N TIMES
170 RN = INT(RND * N) + 1: K = INT(RND * N) + 1
180 FOR C = 1 TO N: SWAP A(RN, C), A(K, C)
190 SWAP B(RN, C), B(K, C): NEXT C: REM ROWS SWAPPED
200 RN = INT(RND * N) + 1: K = INT(RND * N) + 1
210 FOR R = 1 TO N: SWAP A(R, RN), A(R, K)
220 SWAP B(R, RN), B(R, K): NEXT R: REM COLUMNS SWAPPED
230 NEXT W: REM END OF RANDOMIZATION OF ROWS AND COLUMNS
240 COLOR 15: DIM Z(N), C(N): J = N: REM RANDOMIZE Z(I) ARRAY
250 FOR K = 1 TO N: C(K) = K: NEXT
260 FOR I = 1 TO N: X = INT(RND * J + 1): Z(I) = C(X): J = J - 1
270 FOR K = X TO J: C(K) = C(K + 1): NEXT K: NEXT I
280 CLS : Z = INT(86 / (N * 4)): IF Z = 0 THEN Z = 1
290 FOR V = 1 TO N: COLOR V MOD 7 + 9: FOR R = 1 TO N: FOR C = 1 TO N
300 IF A(R, C) + Z(V) > N THEN T = A(R, C) + Z(V) - N
310 IF A(R, C) + Z(V) <= N THEN T = A(R, C) + Z(V)
320 IF B(R, C) + Z(V) > N THEN T1 = B(R, C) + Z(V) - N
330 IF B(R, C) + Z(V) <= N THEN T1 = B(R, C) + Z(V)
340 Q$ = CHR$(T + 64) + MID$(STR$(T1), 2)
350 Q$ = Q$ + STRING$(4 - LEN(Q$), 32)
360 LOCATE R + E, G * N * 3 + (C - 1) * 3 + 1 + G * 2 + INT(N / 10) * C
370 PRINT Q$; : NEXT C: NEXT R: LOCATE R + E, G * N * 3 + 2 + G * 2
380 PRINT "LEVEL"; V; : INPUT ; Q$: IF V >= N THEN 400 ELSE G = G + 1
390 IF G = Z THEN G = 0: E = E + N + 1: IF E > 24 - N THEN E = 0: CLS
400 NEXT V
```

B.

```
GRAECO-LATIN CUBE
ENTER 4 OR AN ODD NUMBER OF TREATMENTS? 5
ENTER A SPECIFIC NUMBER FOR A SPECIFIC RANDOM SEQUENCE? 1
B1 A2 C5 E3 D4  E4 D5 A3 C1 B2  D3 C4 E2 B5 A1  C2 B3 D1 A4 E5
D3 C4 E2 B5 A1  B1 A2 C5 E3 D4  A5 E1 B4 D2 C3  E4 D5 A3 C1 B2
A5 E1 B4 D2 C3  D3 C4 E2 B5 A1  C2 B3 D1 A4 E5  B1 A2 C5 E3 D4
C2 B3 D1 A4 E5  A5 E1 B4 D2 C3  E4 D5 A3 C1 B2  D3 C4 E2 B5 A1
E4 D5 A3 C1 B2  C2 B3 D1 A4 E5  B1 A2 C5 E3 D4  A5 E1 B4 D2 C3
 LEVEL 1 ?       LEVEL 2 ?       LEVEL 3 ?       LEVEL 4 ?
A5 E1 B4 D2 C3
C2 B3 D1 A4 E5
E4 D5 A3 C1 B2
B1 A2 C5 E3 D4
D3 C4 E2 B5 A1
 LEVEL 5 ?
```

Fig. 7. (A) Listing of BASIC program used to randomly select Graeco–Latin cubes (each letter and number represented once in each horizontal column, horizontal row, and vertical column; and all combinations present in each horizontal square and each vertical-row square, but not in each vertical-column square). (B) Printed output of program on computer screen of an IBM PC or compatible computer (output may differ due to BASIC version used).

Acknowledgement—This work was supported by a research grant from the Swedish Agricultural and Forest Research Council, SJFR.

REFERENCES

1. V. L. Anderson and R. A. McLean, *Design of Experiments*. Marcel Dekker, New York (1974).
2. W. G. Cochran, *Sampling Techniques*. Wiley, New York (1963).
3. W. G. Cochran and G. M. Cox, *Experimental Designs*. Wiley, New York (1957).
4. B. E. Cooper, *Statistics for Experimentalists*. Pergamon Press, London (1969).
5. D. R. Cox, *Planning of Experiments*. John Wiley, New York (1958).
6. M. N. Das and N. C. Giri, *Design and Analysis of Experiments*. Wiley, New Delhi (1979).
7. W. E. Deming, *Some Theory of Sampling*. Dover, New York (1950).
8. W. T. Federer, *Experimental Design*. MacMillan, New York (1955).
9. O. Kempthorne, *The Design and Analysis of Experiments*. Wiley, New York (1952).
10. J. A. Byers, Basic algorithms for random sampling and treatment randomization, *Comput. Biol. Med.* **21**, 69 (1991).
11. J. A. Byers, Randomization algorithms in BASIC for experimental design, *Comput. Biol. Med.* **23**, 167 (1993).
12. P. J. Clark and F. C. Evans, Distance to nearest neighbor as a measure of spatial relationships in populations, *Ecology* **35**, 445 (1954).
13. J. A. Byers, Dirichlet tessellation of bark beetle spatial attack points, *J. Anim. Ecol.* **61**, 759 (1992).
14. B. D. Ripley, *Spatial Statistics*. Wiley, New York (1981).
15. W. L. Stevens, The completely orthogonalised square, *Ann. Eug.* **9**, 82 (1938).
16. R. C. Bose, On the application of Galois fields to the problem of the construction of Hyper-Graeco-Latin squares, *Sankhya* **3**, 323 (1938).
17. J. A. Byers, Attraction of bark beetles, *Tomicus piniperda*, *Hylurgops palliatus*, and *Trypodendron domesticum* and other insects to short-chain alcohols and monoterpenes, *J. Chem. Ecol.* **18**, 2385 (1992).
18. J. A. Byers, Host tree chemistry affecting colonization in bark beetles. In *Chemical Ecology of Insects 2*, R. T. Cardé and W. J. Bell (Eds), pp. 154–213. Chapman & Hall, New York (1995).

About the Author—JOHN A. BYERS received degrees in Entomology from Colorado State University (B.Sc. 1971 and M.Sc. 1973) and the University of California at Berkeley (Ph.D. 1978). He was awarded a Docent title (1989) at the Department of Ecology, Lund University, Sweden. He is currently a Högskolelektor (Associate Professor) at the Swedish University of Agricultural Sciences, Alnarp, Sweden. His research interests include insect behavior, chemical ecology, and computer simulation models.