Paper 33-25

A WYSIWYG Application to Summarize Categorical Variables Using the SAS® System

Dorothy E. Pugh, University of North Carolina at Chapel Hill, Chapel Hill, NC
Jeffrey M. Abolafia, University of North Carolina at Chapel Hill, Chapel Hill, NC

ABSTRACT

An ideal SAS frequency table program would generate tables presenting counts (and percents based on those counts) derived from any sets of observations that a user specified, and would put them in the locations the user chose. It would also contain a WYSIWYG interface that made efficient use of code. This paper describes a new high-level frequency table generation language (FlexTab) designed to meet these special needs. This language allows the user to control on a per-cell basis which observations to count to generate an N statistic (on the basis of specified variable values), and which two sets of observations to use to generate a fraction, converted to a percentage via PROC FORMAT. It uses PROC MEANS with a CLASS statement to create a data set with all the necessary frequency counts and uses the RETAIN statement with newly created denominator variables to put numerator and denominator values for that fraction in the same observation. The FlexTab compiler generates both this data manipulation code and its associated PROC TABULATE code.

INTRODUCTION

PROC TABULATE and PROC REPORT have certain shortcomings: they put many restrictions on the observations you can use to calculate denominators directly from an input data set. PROC TABULATE requires that the denominators be the same as the Ns for the CLASS variables. PROC REPORT gives you more flexibility in defining with its COMPUTE blocks, but you lose the flexibility with totals and subtotals that PROC TABULATE gives you, unless you choose to use the LINE statement, which is a de facto PUT statement. Neither PROC REPORT nor PROC TABULATE are really WYSIWYG: the layout of the final table doesn't stand out unless it's very simple. Finally, both PROC's are black boxes: neither creates a SAS data set of computed summary statistics that you can actually print out as a diagnostic measure.

With FlexTab, a high-level WYSIWYG (what you see is what you get) frequency table generation language discussed in this paper, you can have your cake and eat it too, in part because it generates PROC TABULATE code. Using it, you can control the observations used in Ns (including both numerators and denominators of fractions) and the cells they are presented in. The compiler (which translates the FlexTab code into SAS code) generates a PROC MEANS step using a CLASS statement for every CLASS variable you specify. The output data set of this PROC MEANS contains counts for every combination of unique values for every combination of CLASS variables. The compiler identifies the observations with the same combinations of class variable values that the FlexTab code specifies and, for each specified column *n*, creates a COL*n* variable and sets it to the value of _FREQ_. Denominator values are RETAINed variables set either to the value of a previously defined COLn variable or newly created. Fractions (which can be converted into percents via a PROC FORMAT step) are the ratios of previously defined COLn variable values and those of previously defined denominator variables. The resulting data set is the input data set to PROC TABULATE code also generated by the compiler. This SAS source code file in turn generates a summary table.

HOW TO USE FLEXTAB: EXAMPLE

Here is a step-by-step illustration of how to use FlexTab to create a summary table. This table shows the difference in reviewer quality by clinical center in a hypothetical clinical trial, thereby identifying centers with possible reviewer training problems. A value of AGREE = 1 means another QC reviewer agreed with the review decision.

Step 1: Example FlexTab code and a line-by-line explanation of what it means

```
class center baseline agree staffid;
2>baseline(1)*agree(.)*staffid(.) = count;
3>baseline(1)*agree(1)*staffid(.) = count;
4>3/baseline(1)*agree(.)*staffid(.) = count;
5>baseline(0)*agree(.)*staffid(.) = count;
6>baseline(0)*agree(1)*staffid(.) = count;
7>baseline(0)*agree(1)*staffid(.)/5 = count;
8>staffid in center = count;
9>staffid(agree = sum(=>6)) in center = count;
10>9/8 = count;
```

First line: List all CLASS (categorical) variables to be used, including those the values of which you will use to control the observations used in calculations. Put the "row" (Column 1) variable ("center") first.

Second line: The "2>" indicates that the output will go in column 2. We start numbering with column 2 because there is one "row" variable, which occupies column 1. This code selects all the observations for which the variable BASELINE = 1 and AGREE and STAFFID have any value, counts them, and puts them in column 2. CENTER is not used in this expression: however, the line "2>center(. 1-8)* baseline(1)*agree(.)*staffid(.) = count;" would be its equivalent. These lines would also be equivalent:

```
2>center(. 1-4)*baseline(1)*agree(.)*staffid(.) =
count,
center(5-8)*baseline(1)*agree(.)*staffid(.) = count;
```

Lines 3,5 and 6 follow the same basic logic.

Fourth line: This line calculates a fraction. The numerator is the same as the N generated by the third line, and is referenced by the "3" accordingly. The denominator, after the "/", is another N calculated the same way as those in lines 2 and 3.

Seventh line: This time, the denominator is the same as the N generated by the fifth line, and is referenced by the "5."
Eighth line: Use the "in" operator to count the number of STAFFID's per CENTER, i.e., the number of unique values of STAFFID within CENTER.
Ninth line: This line 1) takes the sum of AGREE=1 observations for each STAFFID, and 2) counts the number of STAFFID's per CENTER (as defined above) who have at least 6 observations with AGREE=1.
Tenth line: To get the column 10 value, we divide the column 9 value by the column 8 value (as computed above) by simply referring to their column numbers.

Step 2: Generation of data manipulation and table generation code. Flextab has two input parameters, the filespec for the input file, a flat file containing the code given in Step 1, and the filespec for the output file, which contains the code below for this example. Note that you still need to fill in the blanks and to replace every "?" with valid SAS code. The code used to create data set IN for this example is shown in Appendix A.

```
proc format;
  picture countfmt
    low-high = ' 0009  '
    other    = ' 0  '
  ;
  picture pctfmt
    low - <0 = ' N/A ' (noedit)
    other    = '  009.9 '  (mult=1000)
    ;
run;

%let rowfmt=;
%let rowsize=;
%let misstxt=;
%let formchar= _ _ _ _ ;
%let box=;     /* Header of first column */
%let colhd2 = ;
%let colhd3 = ;
%let colhd4 = ;
%let colhd5 = ;
%let colhd6 = ;
%let colhd7 = ;
%let colhd8 = ;
%let colhd9 = ;
%let colhd10 = ;

%let colfmt2 =;
%let colfmt3 =;
%let colfmt4 =;
%let colfmt5 =;
%let colfmt6 =;
%let colfmt7 =;
%let colfmt8 =;
%let colfmt9 =;
%let colfmt10 =;

*************************************
*     Build Data IN
*************************************;

data in;
  ???
run;

proc sort data=in;
  by center staffid;
run;

data in;
  set in;
  by center staffid;
  if first.staffid  and staffid  gt    .z then count8 = 1;
run;

proc sort data=in;
  by center staffid;
run;

data in;
  set in;
  by center staffid;
  retain presum  0;
  if first.staffid  then presum  = 0;
  presum  = sum (presum ,agree );
  if last.staffid  and presum =>6  then count9 = 1;
  people = 1;
run;

proc sort data=in;
  by center baseline agree staffid;
run;

proc means data=in noprint;
  class center baseline agree staffid;
  var count8 count9 people;
  output out=sums sum= count8 count9 people;
run;

proc sort data=sums;
  by center baseline agree staffid;
run;

data sums;
  set sums;
  by center baseline agree staffid;
  retain den4_1;
  retain denom5 denom8;
  if baseline  eq 1  and agree  eq    .  and staffid  eq
     . then col2  = people;
  if baseline  eq 1  and agree  eq 1  and staffid  eq    .
     then col3  = people;
  if baseline  eq 1  and agree  eq    .  and staffid  eq
     . then den4_1  = people;
  if baseline  eq 0  and agree  eq    .  and staffid  eq
     . then col5  = people;
  if baseline  eq 0  and agree  eq 1  and staffid  eq    .
     then col6  = people;
  if col5  ne    . then denom5  = col5 ;
  if baseline eq    . and agree eq    . and staffid eq    .
     then col8  = count8;
  if baseline eq    . and agree eq    . and staffid eq    .
     then col9  = count9;
  if col8  ne    . then denom8  = col8 ;
```

```
  col4  = col3 /den4_1 ;
  if baseline  eq 0  and agree  eq 1  and staffid  eq   .
     and denom5 ne    . then col7  = people/denom5
     ;
  col10  = col9 /   denom8 ;
  if col2 ne . or col3 ne . or col4 ne . or col5 ne . or
     col6 ne . or col7 ne . or col8 ne . or col9 ne . or
     col10 ne .;
run;

PROC TABULATE DATA = sums
          NOSEPS
          MISSING
          FORMCHAR= " &formchar "
          FORMAT=&rowfmt ;
  VAR col2 - col10 ;
  CLASS  center;
  TABLE center = ' ' ?
    (
    col2 = "&colhd2"*mean=' '*f=&colfmt2
    col3 = "&colhd3"*mean=' '*f=&colfmt3
    col4 = "&colhd4"*mean=' '*f=&colfmt4
    col5 = "&colhd5"*mean=' '*f=&colfmt5
    col6 = "&colhd6"*mean=' '*f=&colfmt6
    col7 = "&colhd7"*mean=' '*f=&colfmt7
    col8 = "&colhd8"*mean=' '*f=&colfmt8
    col9 = "&colhd9"*mean=' '*f=&colfmt9
    col10 = "&colhd10"*mean=' '*f=&colfmt10
    )
    /PRINTMISS
     MISSTEXT =  "&misstxt"
     RTS = "&rowsize"
     BOX = " &box " ;
run;
```

Step 3: Fill in the blanks and "?'s" on the above file and run it. Note that you should replace the "?" in the TABLE statement with a "*" or a ",". Final summary table output for this example is presented in Appendix B. A printout of the final data set input to PROC TABULATE code is presented in Appendix C.

COMPILER DESIGN

A compiler is a program that reads input code written in one computer language, parses it (i.e., analyzes its structure and components), and generates equivalent code in another, usually lower-level, language. This section will explain how this compiler was designed and display an intermediate data set, a data dictionary that contains a data set containing the results of the parsing.

The interpretation consists of 1) parsing the code, i.e., recognizing and storing grammatical units (i.e., by applying rules of that input language's grammar, and 2) converting each grammatical unit into code for an equivalent grammatical unit in the output language. In this case, COMP.SAS reads the input code as an input flat file, parses it, puts the language units into a SAS data set, and, on the basis of its content, writes equivalent SAS code to an output flat file.

The grammar of this language is analogous to that of English. English sentences are broken up into syntactical units, such as "subjects" and "predicates." In turn, each syntactical unit is broken up into "parts of speech" which include nouns, verbs, adjectives, adverbs, etc. This is not a hierarchical relationship, however, since, for instance, a noun can be either part of a "subject" or a "predicate." However, different "parts of speech" are allowable in a "subject" and others in a "predicate."

Imagine, then, a new grammar in which the syntactical units have numbers. Within them, "Variable" and "Stat" are roughly equivalent to nouns or noun clauses, and are modified by "quantifiers," which may be constants, variables or expressions. Operators, somewhat like verbs in English, are "*" (intersection), "/" (division), "," (end of a phrase), ";" (end of a sentence). A sentence fully describes the contents of a summary table column, while a phrase describes part of a column, sometimes as little as a table cell. Another operator, "in," represents the observation count for one variable within each unique value of another.

A state transition diagram describing the language grammar serves a compiler programming aid in the way that a flow chart serves as a data processing program aid. It is displayed in Figures 1 and 2. Figure 2 is an expansion of the "Quantifier" state in Figure 1 describing the special construction in Statement 9 of the FlexTab code above.

Figure 1. State Transition Diagram

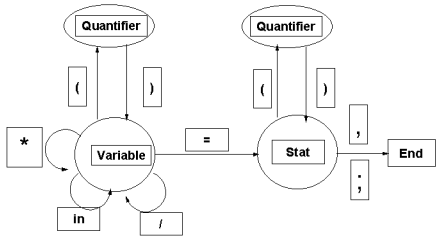
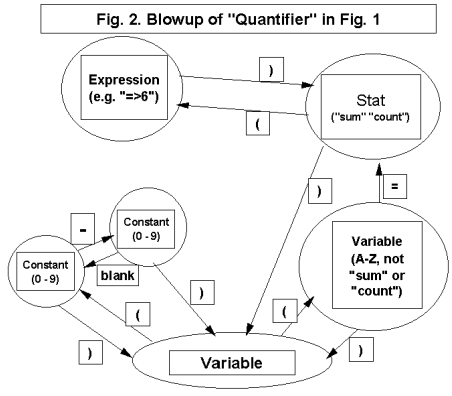


Fig. 2. Blowup of "Quantifier" in Fig. 1



ANOTHER EXAMPLE

One feature the previous example does not illustrate is the use of one category variable as a row sequence number designator, which can in turn have a complicated format illustrating hierarchical relationships. This format illustrates what the first column of the table would look like:

```
VALUE fmt
     1 = 'Hypertension              Yes'
     2 = '    No'
     3 = '   Unknown'
     4 = '    Missing'
     5 = 'Diabetes mellitus         Yes'
     6 = '   No'
     7 = '   Unknown'
     8 = '   Missing'
     9 = 'Insulin treated diabetes  Yes'
    10 = '   No'
    11 = '  Unknown'
    12 = '   Missing'
    13 = 'Oral hypoglycemic Rx      Yes'
    14 = '  No'
    15 = '  Unknown'
    16 = '  Missing'
    17 = 'Smoking History           Yes'
    18 = '   No'
    19 = '   Unknown'
    20 = '   Missing'
    21 = 'Hypercholesterolemia      Yes'
    22 = '   No'
    23 = '   Unknown'
    24 = '   Missing'
    25 = 'Estrogen Prior to MI      Yes'
    26 = '   No'
    27 = '  Unknown'
    28 = '   Missing'
    29 = 'Estrogen use, Current     Yes'
    30 = '   No'
    31 = '  Unknown'
    32 = '   Missing'
     ;
```

Here is the accompanying FlexTab code:

```
class varno group sex;
3>varno(1-24)*sex(' ') = count,
   varno(25-32)*sex('F') = count;
4>varno(1-8)*sex(' ')/varno(0)*sex(' ') = count,
   varno(9-12)*sex(' ')/varno(5)*sex(' ') = count,
   varno(13-24)*sex(' ')/varno(0)*sex(' ') = count,
   varno(25-28)*sex('F')/varno(0)*sex('F') = count,
   varno(29-32)*sex('F')/varno(25)*sex('F') = count;
```

After you generate the SAS code with this FlexTab code, you will still need to calculate the values of VARNO by assigning either 1 or 0 to a series of logical variables, e.g., R0-R32. Set R0 to 1, then assign the others according to whether they meet the binary conditions described in the above format. Then transpose them into a single variable called VARNO, which will have 33 observations of binary values. At the end of the last DATA IN data step, DROP all observations for which VARNO is either missing or 0.

This is a table with many different N's and denominators. For some entries, we need to report data that applies only to diabetic patients (VARNO=5), women (SEX='F'), or women who have taken estrogen prior to MI (SEX='F'*VARNO(25)).

Note that the second line starts with "3>". This indicates that there are two categorical variables this time (VARNO and GROUP) that will appear in the CLASS statement of the PROC TABULATE code that this FlexTab code will generate. Two columns are specified (3 and 4), but the final number of columns depends on what you replace the "?" with in the TABLE statement. If you replace it with "*", you will have four columns, one each for VARNO and GROUP, and two reflecting, respectively, the counts and fractions as specified. However, if you replace the "?" with ",", the results will not be quite as WYSIWYG: You will have one COLUMN for VARNO values, then six others:, the unique values of GROUP by the two columns above. Since GROUP values were not restricted by the above FlexTab statements, they default to the two unique values GROUP assumes, plus a third overall category.

This is the resulting SAS code:

```
proc format;
  picture countfmt
    low-high = ' 0009  '
    other    = ' 0  '
  ;
  picture pctfmt
    low - <0 = ' N/A ' (noedit)
    other    = '  009.9 '  (mult=1000)
    ;
run;

%let rowfmt=;
%let rowsize=;
%let misstxt=;
%let formchar= _ _ _ _ ;
%let box=;    /* Header of first column */
%let colhd3 = ;
%let colhd4 = ;

%let colfmt3 =;
%let colfmt4 =;

**************************************
*    Build Data IN
**************************************;
data in;
  ???
run;

proc sort data=in;
  by varno group sex;
run;

proc means data=in noprint;
  class varno group sex;
  var people;
  output out=sums sum= people;
run;

proc sort data=sums;
  by varno group sex;
run;

data sums;
  set sums;
```

```
by varno group sex;
retain den4_1 den4_2 den4_3 den4_4 den4_5;
if 1 <=varno <=24  and sex  eq ' ' then col3  = people;
if 25 <=varno <=32  and sex  eq 'F' then col3  = people;
if varno  eq 0  and sex  eq ' ' then den4_1  = people;
if varno  eq 5  and sex  eq ' ' then den4_2  = people;
if varno  eq 0  and sex  eq ' ' then den4_3  = people;
if varno  eq 0  and sex  eq 'F' then den4_4  = people;
if varno  eq 25  and sex  eq 'F' then den4_5  = people;

if 1 <=varno <=8 and sex eq ' ' and den4_1 ne . then
    col4    = people/den4_1;
if 9 <=varno <=12 and sex eq ' ' and den4_2 ne . then
    col4    = people/den4_2;
if 13 <=varno <=24 and sex eq ' ' and den4_3 ne . then
    col4    = people/den4_3;
if 25 <=varno <=28 and sex eq ' ' and den4_4 ne . then
    col4    = people/den4_4;
if 29 <=varno <=32 and sex eq ' ' and den4_5 ne . then
    col4    = people/den4_5;
```

CONCLUSION

We have shown you how to use FlexTab to do most of the work toward generating an important type of summary table with complexities that once represented a quality assurance nightmare. The section explaining the compiler should serve as a guide to help you modify your own copy of the code from us, which we will distribute on request. This new language can, at the very least, serve as a useful communication tool between you and the most harried statistician or medical monitor. At the other extreme of possibilities, it can help you 1) to identify bugs in the most complex algorithms generating categorical variables, 2) to spot extra, i.e., unwanted, observations in your data set, 3) to display troublesome trends in complex abstracted or otherwise categorical data, including signs of danger in biomedical data. We especially hope that this prototype program will someday lead to the development of a power tool reducing the current burden on the clinical monitoring aspect of clinical trials programming enough to keep clinical trials from being the bottleneck in the prohibitively expensive drug development process.

ACKNOWLEDGMENTS

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks of their respective companies.

APPENDIX A
Example of user-written code creating "raw" input data

```
****************************************
*     Build Data  IN
****************************************;

data in;
  do center = 1 to 8;
    do pat = 1 to 5;
      patid = 10*center+pat;
    if col3 ne . or col4 ne .;
run;

PROC TABULATE DATA = sums
          NOSEPS
          MISSING
          FORMCHAR= " &formchar "
          FORMAT=&rowfmt ;
  VAR col3 - col4 ;
  CLASS  varno group;
  TABLE varno = ' ' ? group = ' ' ?
    (
    col3 = "&colhd3"*mean=' '*f=&colfmt3
    col4 = "&colhd4"*mean=' '*f=&colfmt4
    )
    /PRINTMISS
     MISSTEXT =  "&misstxt"
     RTS = "&rowsize"
     BOX = " &box " ;
run;
```

REFERENCES

Abolafia, Jeffrey M. and Stephen M. Noga (1997), "The Tabulate Procedure: One Step Beyond the Final Chapter," *Proceedings of the Sixth Annual Southeast SAS Users Group Conference*,6,292-300.

SAS Institute Inc. (1990), *The SAS Guide to TABULATE Processing,* Second Edition, Cary, NC: SAS Institute, Inc.

SAS Institute Inc., *The SAS Guide to the REPORT Procedure: Usage and Reference, Version VI,* First Edition, Cary, NC: SAS Institute, Inc.

Author Contact Information

Dorothy Pugh
29 Sandstone Ridge Dr.
Durham, NC 27713
(919) 493-1237
KarlG87710@aol.com

Jeffrey Abolafia
University of North Carolina at Chapel Hill
Department of Biostatistics CB #803
Chapel Hill, NC 27514
(919) 966-5304
jeff_abolafia@mail.cscc.unc.edu

```
      do vis = 1 to 5;

        baseline = vis gt 2;
        agree = ranuni(357) gt    .5;
        output;
      end;
    end;
  end;
run;

data in;
  set in;
  staffid = 100*center + int(2*ranuni(2));
RUN;
```

APPENDIX B
Summary Table Output

QC Overread Stats for Staff

Clinical Center	Number Overread Baseline	Number Agr. Baseline	Percent Agr. Baseline	Number Overread Followup	Number Agr. Followup	Percent Agr. Followup	Number of Interviewers	Number with >=6 Overread Agr.	Percent with >=6 Overread Agr.
Overall	120	72	60.0	80	41	51.2	16	13	81.2
A Univ.	15	12	80.0	10	2	20.0	2	2	100.0
B Univ.	15	11	73.3	10	6	60.0	2	2	100.0
C Univ.	15	11	73.3	10	4	40.0	2	2	100.0
D Univ.	15	9	60.0	10	3	30.0	2	1	50.0
E Univ.	15	8	53.3	10	7	70.0	2	2	100.0
F Univ.	15	5	33.3	10	4	40.0	2	1	50.0
G Univ.	15	6	40.0	10	8	80.0	2	1	50.0
H Univ.	15	10	66.6	10	7	70.0	2	2	100.0

APPENDIX C
Observations from input data set to PROC TABULATE

Obs	clinic	baseline	staff	_TYPE_	_FREQ_	count8	count9	pop1	den4	denom58	den12	col2	col3	col5	col6	col8	col9	col4	col7	col10
1	.	.	.	0	200	16	13	200	.	.	16	.	.	.	.	16	13	.	.	0.8125
2	.	0	.	4	80	13	.	80	.	80	16	.	.	80	.	.	.	.	.	.
3	.	0	1	6	41	9	.	41	.	80	16	.	.	.	41	.	.	.	0.5125	.
4	.	1	.	4	120	3	13	120	120	80	16	120	.	.	.	.	.	.	.	.
5	.	1	1	6	72	2	11	72	120	80	16	.	72	.	.	.	.	0.60000	.	.
6	1	.	.	8	25	2	2	25	120	80	2	.	.	.	.	2	2	.	.	1.0000
7	1	0	.	12	10	2	.	10	120	10	2	.	.	10	.	.	.	.	.	.
8	1	0	1	14	2	.	.	2	120	10	2	.	.	.	2	.	.	.	0.2000	.
9	1	1	.	12	15	.	2	15	15	10	2	15	.	.	.	.	.	.	.	.
10	1	1	1	14	12	.	1	12	15	10	2	.	12	.	.	.	.	0.80000	.	.
11	2	.	.	8	25	2	2	25	15	10	2	.	.	.	.	2	2	.	.	1.0000
12	2	0	.	12	10	2	.	10	15	10	2	.	.	10	.	.	.	.	.	.
13	2	0	1	14	6	2	.	6	15	10	2	.	.	.	6	.	.	.	0.6000	.
14	2	1	.	12	15	.	2	15	15	10	2	15	.	.	.	.	.	.	.	.
15	2	1	1	14	11	.	2	11	15	10	2	.	11	.	.	.	.	0.73333	.	.
16	3	.	.	8	25	2	2	25	15	10	2	.	.	.	.	2	2	.	.	1.0000
17	3	0	.	12	10	1	.	10	15	10	2	.	.	10	.	.	.	.	.	.
18	3	0	1	14	4	1	.	4	15	10	2	.	.	.	4	.	.	.	0.4000	.
19	3	1	.	12	15	1	2	15	15	10	2	15	.	.	.	.	.	.	.	.
20	3	1	1	14	11	1	2	11	15	10	2	.	11	.	.	.	.	0.73333	.	.
21	4	.	.	8	25	2	1	25	15	10	2	.	.	.	.	2	1	.	.	0.5000
22	4	0	.	12	10	1	.	10	15	10	2	.	.	10	.	.	.	.	.	.
23	4	0	1	14	3	1	.	3	15	10	2	.	.	.	3	.	.	.	0.3000	.
24	4	1	.	12	15	1	1	15	15	10	2	15	.	.	.	.	.	.	.	.
25	4	1	1	14	9	.	1	9	15	10	2	.	9	.	.	.	.	0.60000	.	.
26	5	.	.	8	25	2	2	25	15	10	2	.	.	.	.	2	2	.	.	1.0000
27	5	0	.	12	10	1	.	10	15	10	2	.	.	10	.	.	.	.	.	.
28	5	0	1	14	7	1	.	7	15	10	2	.	.	.	7	.	.	.	0.7000	.
29	5	1	.	12	15	1	2	15	15	10	2	15	.	.	.	.	.	.	.	.
30	5	1	1	14	8	1	2	8	15	10	2	.	8	.	.	.	.	0.53333	.	.
31	6	.	.	8	25	2	1	25	15	10	2	.	.	.	.	2	1	.	.	0.5000
32	6	0	.	12	10	2	.	10	15	10	2	.	.	10	.	.	.	.	.	.
33	6	0	1	14	4	1	.	4	15	10	2	.	.	.	4	.	.	.	0.4000	.
34	6	1	.	12	15	.	1	15	15	10	2	15	.	.	.	.	.	.	.	.
35	6	1	1	14	5	.	.	5	15	10	2	.	5	.	.	.	.	0.33333	.	.
36	7	.	.	8	25	2	1	25	15	10	2	.	.	.	.	2	1	.	.	0.5000
37	7	0	.	12	10	2	.	10	15	10	2	.	.	10	.	.	.	.	.	.
38	7	0	1	14	8	1	.	8	15	10	2	.	.	.	8	.	.	.	0.8000	.
39	7	1	.	12	15	.	1	15	15	10	2	15	.	.	.	.	.	.	.	.
40	7	1	1	14	6	.	1	6	15	10	2	.	6	.	.	.	.	0.40000	.	.
41	8	.	.	8	25	2	2	25	15	10	2	.	.	.	.	2	2	.	.	1.0000
42	8	0	.	12	10	2	.	10	15	10	2	.	.	10	.	.	.	.	.	.
43	8	0	1	14	7	2	.	7	15	10	2	.	.	.	7	.	.	.	0.7000	.
44	8	1	.	12	15	.	2	15	15	10	2	15	.	.	.	.	.	.	.	.
45	8	1	1	14	10	.	2	10	15	10	2	.	10	.	.	.	.	0.66667	.	.