Paper 063-2010

Challenges In Processing Genomic Data II - One BIG File

Derek Morgan, Principle Solutions Group, St. Louis, MO

ABSTRACT

How do you take a random sample of 1000 SNPs from a seventy-nine gigabyte flat file with one record per SNP per subject, and turn that into something that can be analyzed, with one record per subject and 1000 variables, each one with the name of the SNP, in a reasonable amount of time? You take a healthy dose of the DATA step and the macro facility, add a pinch of the SURVEYSELECT procedure, auto-generate some SAS code and bake.

THE PROBLEM

We had a seventy-nine GB file of SNP (Single Nucleotide Polymorphism) data. There weren't very many variables, but there were well over 1.29 billion records, arranged as one record per subject per SNP per measurement. The output dataset we needed would be arranged as one record per subject (without duplicates) and the genotype of the SNP would be stored in a variable with the SNP name. We also wanted to select a sample of 1000 SNPs from this file, with two of them pre-selected, and the remaining 998 chosen at random. The first attempt at solving this problem was to extract the desired SNP records into an individual file using *awk* inside of a LINUX shell script, with the intent to process the resulting file with SAS. While this method of extraction worked, unfortunately, it took two hours to extract a single SNP. The time allotted for completion of this task was five to six hours.

THIS SOLUTION

We used a two-pass method through the raw data that focused on speeding up the processing of the data we needed. The first pass was designed to get the universe of SNPs for the random sample without caring about the rest of the data, while the second pass only read the fields we were interested in and only from the SNPs we wanted. Surprisingly, the transposing of the data and merging into its final form did not take very long, relatively speaking. The entire solution took approximately two hours and thirty-five minutes to execute, which is thirty-five minutes longer than the LINUX script took to extract one SNP from the original file.

THE FIRST PASS

This timed at a little over two hours. The first part of this is obtaining the SNP identifier from the 79GB flat file. To save time, we did not process the entire input line, only the first field. It took 28:30 to do this. The biggest chunk of time involved removing the duplicate SNP names from the first pass INPUT statement (❶), which took almost an hour and a half. In later benchmark testing, it was shown that the SORT procedure outperformed the SQL procedure (using SELECT DISTINCT.) Once we had the universe of SNP names, we used the SURVEYSELECT procedure (❷) to get our random sample. The WHERE= option on the dataset name option accounts for the two preselected SNPs, because we don't want them as part of the random sample. They will be added to the sample afterwards.

```
1   %GLOBAL time1 time2 time3;
2   OPTIONS SOURCE2;
3
4   %LET rawlib=/data/PLCO_data;
5   %LET rootlib=/data/test;
6   LIBNAME project "&rootlib";
7
8   %LET time1=%SYSFUNC(DATETIME());
9   DATA gtyps;
10  LENGTH snpid $ 20;
11  INFILE "&rawlib/genotypes.txt" PAD MISSOVER DLM='09'x FIRSTOBS= 2;      ❶
12  INPUT snpid;
13  RUN;
14
15  PROC SORT DATA=gtyps NODUPKEY;
16  BY snpid;
17  RUN;
18
19  PROC SURVEYSELECT NOPRINT DATA=gtyps (WHERE=(TRIM(snpid) NOT IN
20  ('rs1447295','rs6983267'))) METHOD=SRS N=998 OUT=these_snps;          ❷
```

```
21  RUN;
```

THE SECOND PASS

The second pass through the 79 GB file processes the SNP name field for every line, but instead of processing the entire line, it only uses that as a key value to determine what should happen with the rest of the line. Since we already obtained the SNP names from the random sample, we create a gigantic IF…IN…THEN DO; clause containing those SNPs (and our pre-selected SNPs) by using a DATA _NULL_ step (❸) to write it to a file that will be included when the second pass is executed. The code that reads the remainder of the line only executes if the SNP name is in our desired sample, the list of which is in the file "inclause.sas" that we generated in ❸. We put the genotype together from the allele values, get rid of the rest, and output the record. Otherwise, the input line is released. The OUTPUT; statement in line 46 means that although SNPID is in the program data vector (read by line 41), we do not have to delete the skipped record explicitly.) After this step, we have all the data we need. The final step before we rename and transpose the data is to add our preselected SNPs to the list of random SNPs (lines 55-66.)

```
22  DATA _NULL_;
23  LENGTH snpname $ 40;
24  SET these_snps end=eof;
25  RETAIN ct 0;
26  FILE "inclause.sas" NOPRINT;
27  IF _N_ EQ 1 THEN
28     PUT "IF snpid IN ('rs1447295','rs6983267'," @;     ❸
29  snpname = QUOTE(TRIM(snpid));
30  PUT snpname @;
31  IF eof THEN
32     PUT ") THEN DO;";
33  ELSE
34     PUT "," @;
35  RUN;
36
37  DATA project.gamble;
38  LENGTH snpid $ 20 id 5 genotyp $ 8;
39  INFILE "&rawlib/genotypes.txt" PAD MISSOVER DLM='09'x FIRSTOBS= 2 DSD;
40  INPUT snpid $ @;
41  %INCLUDE "inclause.sas";
42     INPUT id specimen allele1 $ allele2 $;
43     genotyp = CATX('/',allele1,allele2);        ❹
44     DROP rawy specimen allele1 allele2;
45     OUTPUT;
46  END;  /* END OF DO; clause FROM INCLUDE FILE inclause.sas */
47  ELSE
48     INPUT;  /* Release input line */        ❺
49  RUN;
50
51  %LET time2 = %SYSFUNC(DATETIME());
52  LIBNAME indv "&rootlib/indv";
53
54  DATA snplst;
55  LENGTH snpid $ 20;
56  INFILE cards;
57  INPUT snpid $;
58  CARDS;
59  rs1447295
60  rs6983267
61  ;;;;;
62  RUN;
63
```

```
64  PROC APPEND BASE=snplst DATA=these_snps FORCE;
65  RUN;
```

THE TRANSPOSITION

PROC TRANSPOSE works faster than a macro method like the one below. However, the macro already existed as part of a toolkit, and could just be dropped into the program. Given the time constraints, ultimate efficiency was something to be pursued later. ❻ indicates the code that obtains the number of records in the dataset. You can find out many things about a SAS dataset by querying its attributes using the ATTRN() or the ATTRC() functions via %SYSFUNC without having to actually process the data. This way, you can do it in open SAS code and store the results in a macro variable.

The first DATA _NULL_ step is used to get the SNP name and put it into macro space. We only read one record from SNPLST by using the POINT= option to access the record directly. When you use the POINT= option, however, you must use a STOP; statement, otherwise, you will cause an infinite loop as the DATA step reads the same record over and over.

In later testing, we used PROC SQL (commented out in lines 89-97) to set the genotype as the value of variable represented by the SNP name, but were surprised to find that it was slower than using the DATA step with the WHERE= dataset option and the SORT procedure. We also could have sped this up a little more by renaming the variable instead of using an assignment statement so that the DATA step in lines 99-104 would not contain any executable statements.

```
66  %LET ds = %SYSFUNC(OPEN(snplst));
67  %LET ct = %SYSFUNC(ATTRN(&ds,NOBS));                    ❻
68  %LET rc = %SYSFUNC(CLOSE(&ds));
69
70  OPTIONS NOMPRINT NOSYMBOLGEN NOMLOGIC;
71
72  %MACRO snp_trnspose(lib=,dsn=);
73  PROC DATASETS LIB=indv KILL; /* Make sure we don't have any old data left over */
74  RUN;
75  QUIT;
76
77  OPTIONS NOTES SYMBOLGEN MPRINT MLOGIC;
78  %DO i=1 %TO &ct;
79      DATA _NULL_; /* Get SNP name */
80      temp = &i;
81      SET snplst POINT=temp;
82      CALL SYMPUT('vname',snpid);
83      STOP;
84      RUN;
85
86  /* DATA step + sort benchmarked faster than PROC SQL
87      PROC SQL;
88      CREATE TABLE indv.&vname AS
89      SELECT distinct(id), genotyp AS &vname
90      FROM &lib..&dsn
91      WHERE snpid EQ "&vname"
92      ORDER BY id;
93      QUIT;
94  */
95
96      DATA indv.&vname;
97      SET &lib..&dsn (WHERE=(snpid EQ "&vname"));
98      LENGTH &vname $ 8;
99      &vname = genotyp; /* Explicit - in hindsight, could have used RENAME */
100     DROP snpid genotyp;
101     RUN;
```

```
102     PROC SORT DATA=indv.&vname NODUPKEY;
103     BY id;
104     RUN;
105 %END;
```

PUTTING THE DATA TOGETHER

This method puts each SNP file in an individual permanent dataset for later use, although this application cleans those datasets up. You could reduce the disk overhead by re-using the same temporary dataset for each SNP, and including the assembly inside of the transposition loop. Here, the final assembly of the data is done in 3 steps: 1) getting a list of all the files in the temporary directory via PROC CONTENTS (line 110); 2) popping a dataset name out (lines 126-132), and 3) performing a MERGE of the final dataset with each SNP dataset, except for the first SNP dataset (lines 134-143.) The first SNP dataset is SET to create the final dataset. In this instance, the individual SNP files are cleaned off after we are done. The remainder of the program is devoted to timing, since the original intent was to work towards finding the most efficient method of doing this, but that project will have to remain unfinished.

```
106 PROC CONTENTS DATA=indv._all_ OUT=snps DIRECTORY NOPRINT;
107 RUN;
108
109 OPTIONS NOTES;
110 PROC SORT DATA=snps NODUPKEY;
111 BY memname;
112 RUN;
113
114 /* Remove any existing final dataset from earlier runs */
115 PROC DATASETS LIB=project;
116 DELETE snp_final;
117 RUN;
118 QUIT;
119
120 %DO i=1 %TO &ct;
121
122     DATA _NULL_;
123     temp = &i;
124     SET snps point=temp;
125     dsn = CATX('.',libname,memname);
126     CALL SYMPUT('dsn',dsn);
127     STOP;
128     RUN;
129
130     DATA project.snp_final;
131     %IF &i EQ 1 %THEN %DO;
132         SET &dsn;
133     %END;
134     %ELSE %DO;
135         MERGE project.snp_final &dsn;
136         BY id;
137     %END;
138     RUN;
139 %END;
140
141 PROC DATASETS LIB=indv KILL;";
142 RUN;
143 QUIT;";
144
145 %MEND snp_trnspose;
146
147 %snp_trnspose(lib=project,dsn=gamble);
```

```
148
149 /* Timing */
150
151 %LET time3=%SYSFUNC(DATETIME());
152
153 DATA timer;
154 t1 = SYMGET('time1');
155 t2 = SYMGET('time2');
156 t3 = SYMGET('time3');
157
158 rnd_sample_time = t2-t1;
159 snp_transpose_time = t3-t2;
160 elapsed_time = t3 - t1;
161
162 FORMAT rnd_sample_time snp_transpose_time elapsed_time TIME8.;
163 RUN;
164
165 PROC PRINT DATA=timer NOOBS;
166 VAR rnd_sample_time snp_transpose_time elapsed_time;
167 RUN;
```

SUMMARY

The programs that collect genomic data aren't currently set up to produce analysis-ready files, but they do produce a great deal of data, and sometimes the files are enormous. Given the large amount of genetic information that can be produced, efficient methods have to be created to process these large files. The SAS System's data manipulation tools are well-suited for this task. All of the data manipulation for this task was accomplished with base SAS, and none of the methods employed for each piece of the task are particularly complicated. While the random sampling could have been performed using base SAS, the SURVEYSELECT procedure achieves the same thing and requires much less programming. All told, this is yet another illustration of breaking a problem into its component parts and solving each sub-problem to yield an effective solution.

ACKNOWLEDGEMENTS

Thank you to Drs. Mike Province and E. Warwick Daw at Washington University Medical School's Division of Statistical Genomics for their permission to present this solution.

CONTACT INFORMATION:

Further inquiries are welcome to:

Derek Morgan
E-mail: derek.p.morgan@gmail.com