PharmaSUG 2012 - Paper HO07

Using the SAS System as a bioinformatics tool: a macro that calls the standalone Basic Local Alignment Search Tool (BLAST) setup

Kevin Viel, Ph.D.
Histonis, Incorporated, Atlanta, GA

ABSTRACT

Aligning DNA and RNA sequences is an integral task in genomics research. For brief and practical purposes, DNA and RNA can be thought of as character strings comprised of A, C, G, and T (DNA) or U (RNA). Within the constraints of the SAS System, namely 32,767 bytes for a character variable, patterns can be matched or "aligned" using, for example, regex expressions (PRXMatch). This size limit typically suffices even for the state of the art sequencing projects, in which read lengths may be well under 15,000 base-pairs. For larger sequences, a SAS programmer may have to consider calling a perl program or use a tool like Basic Local Alignment Search Tool (BLAST®), a popular tool for alignments. An example of an alignment to a sequence larger than the SAS limit might be the need to determine the start position of a primer within a gene, for instance 186 Kb *F8*. The National Center for Biotechnology Information (NCBI) of National Institutes of Health (NIH) provides the blast+ software package. The **goals** of this paper are to describe a SAS macro that calls BLASTn using the X statement and demonstrate an example alignment using the macro.

INTRODUCTION

Genotyping, the determination of the sequence of **DNA** (deoxyribonucleic acid) or **RNA** (ribonucleic acid), is necessary to identify variants that may be associated with human health outcomes. For instance, alleles of a variant locus may be associated with risk of a disease or with the levels (concentration or activity) of a protein of interest. Previously, I presented a programmatic approach using the SAS System®[1] to align or match short sequences within the amplicons of a Sanger Sequencing project that used the flanks of the locus of interest as the match criterion, allowing for at most one base mismatch in one flank or the other, and employed regular expressions (**regexes**, via the PRX functions and calls, such as PRXMATCH)[2]. A second approach, which is more conventional, is to locally alignment the test and the reference sequences. One of the most popular local alignment algorithms is the Basic Local Alignment Search Tool (**BLAST®**)[3, 4], which is available as a web-tool or as a freely downloadable stand-alone suite via the National Center for Biotechnology Information (**NCBI**) of the National Institutes of Health (**NIH**) (http://blast.ncbi.nlm.nih.gov/Blast.cgi). The **goals** of this paper are to describe a SAS macro that calls the BLASTn program of this stand-alone suite using the X statement and demonstrate an example alignment, and 3) illustrate how the macro can be used to create primer and amplicon names.

DNA and RNA. For the purposes of this paper, we will consider DNA as the sequence of four nucleotides (**bases**): adenine (**A**), cytosine (**C**), guanine (**G**) and thymine (**T**). In RNA, uracil (**U**) is present instead of thymine. The bases may be modified, for example cytosine may appear as 5-methylcytosine or even as 5-hydroxymethylcytosine, but since these forms are not readily detected in "conventional" sequencing technologies, such as Sanger Sequencing, I mention them only as a point of interest. DNA is a double-stranded helix in which hydrogen bonding between A and T and between C and G occurs, hence we described the length of DNA in terms of base-pairs (**bp**). The strands of DNA (and double-stranded RNA) are reverse complimentary and, by convention, we list the sequence in the 5'-3' direction, which refers to the positions of the phosphodiester bonds on the ribose rings. Cells transcribe DNA into RNA (**transcription** or **expression**) in their nuclei and may translate the resulting RNA into proteins (**translation**). Regions of DNA that code for protein are called **genes**. Many functions of RNA beyond the messenger RNA (**mRNA**) involved in translation have been documented and research into some, such as small interfering RNA (**siRNA**), is burgeoning.

A DNA variant may be a single nucleotide substitutions (**SNS**), an insertion or deletion (collectively, **INDEL**s), or an inversion (see **Table 1**). SNSs are the most common and may occur every 300 to 1,000 bp. In the example in Table 1, the inversion is probably contrived to demonstrate the concept, but such small inversions may exist. An important, real example of an inversion is the Intron 22 Inversion (**I22I**) of *F8*, a recurrent variant that accounts for approximately 40% of all severe Hemophilia A patients. Importantly, I22I involves approximately 10 kilobases (Kb) and the mechanism likely involves 300-400 Kb of DNA. The lengths of sequences for which BLAST might be used, at least in this paper, are much shorter.

Table 1. Types of variants found in DNA.

Type	DNA Sequence
Wild-type (WT)	ATCTGACCGTGATGGCGGCCATTGGCTTGGGCTTCCTCACC ||||||||||||||||||||||||||||||||||||||||| ATCTGACCGTGATGGCGGCCATTGGCTTGGGCTTCCTCACC
SNS	ATCTGACCGTGATGGCGGCCATTGGCTTGGGCTTCCTCACC |||||||||||||||||||| |||||||||||||||||||| ATCTGACCGTGATGGCGGCC**C**TTGGCTTGGGCTTCCTCACC
Insertion	ATCTGACCGTGATGGCGGCCA--TTGGCTTGGGCTTCCTCACC ||||||||||||||||||||| |||||||||||||||||||| ATCTGACCGTGATGGCGGCCA**CG**TTGGCTTGGGCTTCCTCACC
Deletion	ATCTGACCGTGATGGCGGCCATTGGCTTGGGCTTCCTCACC ||||||||||||||||||||||||||||||||||||||||| ATCTGACCGTGATGGCGG**---**TTGGCTTGGGCTTCCTCACC
Inversion	ATCTGACCGTGATGGCGGCCATTGGCTTGGGCTTCCTCACC ||||||||||||||||| ||||||||||||||||||| ATCTGACCGTGATGGCG**TACCG**TGGCTTGGGCTTCCTCACC

SNS's may occur in the **codons** (a sequence of three nucleotides that codes amino acids) of genes; when alteration of the codon results in an amino acid substitution, the SNS may be referred to as a non-synonymous single nucleotide polymorphism (**ns-SNP**) or **missense** mutation, depending on whether the variants is found in "normal" subjects of a population or whether the locus is associated with a deleterious outcome, respectively. The third possibility is that the SNS transforms the codon to one of the three stop codon sequences, in which case we refer to the variant as a **nonsense** mutation. The term polymorphism conventionally refers to a variant that has a minor allele frequency (**MAF**) of 5% or more, but some may use the term for variants with MAF of 1%. For instance, the D(1241)E variant of the coagulation Factor VIII is a typically considered a ns-SNP, but W(0255)C is thought be a missense mutation[5]. D(1241)E was found (consistently) in populations of non-bleeders (non-hemophilia A subjects)[6], whereas W(0255)C was found in patients with hemophilia A. However, one report[7] associated the E_{1241} allele with a loss of FVIII mRNA, highlighting both the difficulties of the genotyping the entire DNA sequence of a 186 Kb gene required to rule out other causes and of concretely elucidating the effect the variant may have on the mRNA or protein produced. In reality, even with mechanistic studies, determining the "definitive" effect(s) of a variant may require large samples with whole genome data: DNA, RNA, and epigenetic (control of expression); a potential limiting factor in the wide-spread adoption of genomics in medicine at this time.

One must always consider the fidelity of the variant discovery investigation (**VDI**) due to technological and resource issues, such as technician-hours available for manual review; it may be possible that another putative disease-causing allele existed, possibly not even in the candidate gene, but was unidentified. This important point emphasizes the advantage of using the SAS System to call stand-alone BLAST or the use of regular expressions (regexes) to match the flanks that I presented in a previous paper[2]: *they enable the users to create a database record for every covered target base, even if it ultimately was not present in the test sequence, in which case the entry should record it as missing*. Missing or poor-quality coverage may result in a "default" call that matches the reference sequence: the question should not only be whether the VDI uncovered evidence that a variant existed ("true positives"), but also whether the VDI data supported the fact that sequence indeed matched the reference sequence ("true negatives"). The potential that in (large scale) projects a failure to document missing or poor-quality coverage results in miscalled bases is something that is easily avoided and audited using programmatic approaches involving BLAST or matching by flanks. The creation of records in a relational genomics database using the results of both the BLAST macro detailed in this paper or the results generated using the approach of my previous paper (regex/flank matching) will be covered in a separate paper or the reader may contact the author for details. The issue is not quite trivial; in a project with 1,000 patients and 4X coverage, an amplicon of 500 bp will generate 2,000,000 records into the projects database and each record will have several fields. Further, projects typically involve numerous amplicons, amounting to a daunting or even infeasible task to perform "by hand".

To genotype, one has to determine the identity of the nucleotides comprising the test sequence of interest. Often one is interested in a candidate locus and has a reference sequence to which one can *map* or *align*, although one might be building an unknown sequence, for instance in *de novo* assemblies. The focus of this paper is the former, however this macro can be used to established the consensus sequence in projects without a reference sequence since this macro can align multiple attempts to generate the test sequence (for instance a forward and reverse pair). The amplicons of a Sanger Sequencing project provide such a reference sequence since the investigator must flank the region of interest by the primers for the polymerase chain reaction (PCR), thereby limiting the size of the amplicon to approximately 1,200 bp. Typically, PCR projects involve genomic regions with completely established sequences, so the investigator knows the sequence of the amplicon when he or she identifies the proper primers. This sequence can be the subject sequence against the investigator aligns the test (query) sequence.

From a successful BLAST alignment, one can identify each of the types of variants listed in Table 1. **Figure 1** presents the results of a BLAST run for which the subject sequence was AAAGAGGTAATAAA CCGAGTAATAGAGCTA, which cheekily translates to “KEVIN R VIEL”. The SAS code in the second panel creates the query sequence. Note that the value of variable *Sequence* in the data set QUERY_DS is actually TAGCTCTATTACGTTTATTACCTCTTT. The section **PANEL 2 Code** below describes this SAS code in detail.

Figure 1. The results of example BLASTn run.

Panel 1

```
BLASTN 2.2.25+
Query= Query
Length=27
Subject= KEVIN R VIEL
Length=30

Score = 36.2 bits (19),  Expect = 8e-009
Identities = 27/30 (90%), Gaps = 3/30 (10%)
Strand=Plus/Minus

Query  1   TAGCTCTATTAC--G-TTTATTACCTCTTT  27
           ||||||||||||  | ||||||||||||||
Sbjct  30  TAGCTCTATTACTCGGTTTATTACCTCTTT  1

Lambda     K      H
1.33    0.621     1.12

Gapped
Lambda     K      H
1.28    0.460    0.850

Effective search space used: 598

Matrix: blastn matrix 1 -2
Gap Penalties: Existence: 0, Extension: 2.5
```

Panel 2

```
01 Data Query_DS ( Keep = Query Sequence ) ;
02   Retain Query "Query" ;
03   File "C:\Subject.fas" ;
04   FP = "AAAGAGGTAATAAAC" ;
05   M  = "CGA" ;
06   TP = "GTAATAGAGCTA" ;
07   Length AA $ 20 ;
08
09   Do I = 1 To 5 ;
10     AA = CatS( AA
11              , Put( SubStr( FP , ( I - 1 ) * 3 + 1 , 3 ) , $C2AA. )
12              ) ;
13   End ;
14
15   AA = CatX( " " , AA , Put( M                    , $C2AA. )) ;
16   AA = CatX( " " , AA , Put( SubStr( TP , 1 , 3 ) , $C2AA. )) ;
17
18   Do I = 2 To 4 ;
19     AA = CatS( AA
20              , Put( SubStr( TP , ( I - 1 ) * 3 + 1 , 3 ) , $C2AA. )
21              ) ;
22   End ;
23
24   Put "> " AA ;
25   Put FP +(-1) M +(-1) TP ;
26
27   Sequence = Upcase(Strip(Reverse(Translate(Upcase(CatS( FP , TP ))
28                                             , "a" , "T"
29                                             , "c" , "G"
30                                             , "g" , "C"
31                                             , "t" , "A"
32                                             )
33                                    )
34                            )
35                      ) ;
36 Run ;
37
38 %BLASTn ( Query_DS = Query_DS
39         , Subject  = C:\Subject.fas
40         , Out_DS   = BLASTn
41         , Option   = %Str(,) " -evalue 1000 -word_size 7"
42         , FDelete  = 0
43         )
```

OVERVIEW OF THE MACRO

The BLASTn macro has three basic parts. The first is a data step the reads a SAS data set and write a text file for each observation and writes subsequent SAS code upon its execution. By virtue of a EXECUTE call routine, the data step makes a system call of the BLASTn program, which writes its results to a text file. The second part of the macro reads this text file and creates a SAS dataset of the parsed contents. The third part of the macro conditionally cleans up the directory by deleting the input and result files.

By examining a possible run of BLASTn as a command line submission, one may gain a better understanding of the macro. In Panel 2 of Figure 1, the command line would be:

```
blastn –subject C:\Subject.fas –query C:\Query.fas –out C:\Query.blastn evalue 1000 -word_size 7
```

EXPLANATION OF SAS CODE

PANEL 2 Code

The DATA statement (Line 01) begins the data step that creates a temporary SAS data set called QUERY_DS. The data set KEEP= option restricts the variables that the data step outputs to *Query* and *Sequence*. The RETAIN statement (Line 02) creates the variable *Query* and assigns it the value “Query”. By virtue of this assignment, *Query* is a character variable of length 5. The FILE statement (Line 03) opens a new text file C:\Subject.fas. The extension “.fas” denotes a FASTA file, which is a text file containing a nucleotide or peptide sequence in single character format. The format is typical 60 or 80 bytes per line and the first line may be a descriptor if it starts with a greater than (>) sign. The input files for BLAST must adhere to this format. The next three lines (Lines 03-05), create the character variables *FP*, *M*, and *TP* and assign their values. The LENGTH statement (Line 07) creates the variable *AA* (short for Amino Acid). A sequence of three codons codes for an amino acid. Five codons comprise *FP*. The Do-Loop (Lines 09-13) extracts them and uses the PUT() function (Line 11) to apply the format $C2AA (Codon to Amino Acid, **Figure 3**). The contiguous codons start at positions 1, 4, 7, 10, and 13 (1-3: AAA, 4-6: GAG,…). The second argument to the SUBSTR() function (Line 11) resolves to these proper starting positions for the given value of the increment *I* while the third, the constant 3, ensures that the result of the function is strictly three bytes in length. The CATS() function concatenates the STRIP() results of the previous value of AA and the results of the PUT() function. At the termination of this Do-Loop, *AA* has the value "KEVIN". The next step is to "translate" the value of *M* to its amino acid and concatenate it to the value of *AA*. However, since the value of AA will be used as a descriptor of the FASTA file, a space before and after the middle initial will improve legibility. The CATX() function (Line 15) concatenates the value of its arguments after the STRIP() function has been applied to each, except the first argument, which is the separator, a single space in this case. After Line 15, the value of *AA* is now "KEVIN R". At this point, the translation of the codons of *TP* and the concatenation of the resulting AAs could proceed by a Do-Loop similar to the first. However, a space between the middle initial and last name is necessary for improved legibility. Therefore, the first codon is extracted and the Do-Loop cycles through codons two through four. Line 16 is analogous to Line 15 but the first argument to the PUT() is the SUBSTR() function. After Line 16, the value of *AA* is now "KEVIN R V". The Do-Loop (Line 18-22) extracts codons 2-4 from *TP* and concatenates the translated values to *AA*. The PUT statement (Line 24) writes the literal string "> " and the value of AA to the first line of C:\Subject.fas. (At this point having seen the translation too much, I wish I could have chosen the phrase "BE SURE TO DRINK YOUR OVALTINE", but both "RALPHIE" and "FRAGILE" would have been possible.) The next PUT statement (Line 25) writes the values of *FP*, *M*, and *TP* to the second line of C:\Subject.fas, but it eliminates the trailing space that occurs by default by explicitly controlling the pointer, which is instructed to move forward -1, in other words, immediately following the value. One could have accomplished this a variety of ways, including creating another variable and using the CATX() function. Finally, to demonstrate the alignment of a reverse compliment sequence, the sequence of interest is manipulated. Remember that the sequence has a deletion by design, the middle initial is purposely omitted. To obtain the reverse compliment of a DNA sequence, each nucleotide must be replaced by it complimentary pair (A->T, C->G, G->C, T->A). To avoid multiple replacements, the TRANSLATE() function (Line 27) replaces the target base with its replacement lower case compliment (A->t). The results of the TRANSLATE() function, which is the compliment, are then the argument to the REVERSE() function. Since trailing blanks become leading blanks in the REVERSE() function, the results of it are then the argument of the STRIP() function. Finally, these results are the argument to the UPCASE function. The macro call (Lines 38-43) then takes the FASTA file and the data in the SAS data set to produce the results in **Panel 1**. The following section describes the macro and, hence, this macro call.

The BLASTn macro

The MACRO statement that indicates the start of the macro code and names the macro, BLASTN, uses **keyword parameters** to give the user flexibility (Lines 1-6). When the user calls the macro, he or she must provide values for the parameters, except for the parameters OPTION and FDELETE, which have default values. The default value for OPTION is null or "", but this requires the use of the %STR() function. Given that the macro employs a system call, the use of the SAS System Options XSYNC, NOXWAIT, and XMIN (Line 08) controls the Windows command prompt. In essence, these options keep the process "in the background".

The data step (Lines 10-31) reads the observations in the SAS data set and 1) writes text files and then call the BLASTn program. The use of the keyword _NULL_ in the DATA statement (Line 10) means that the data step does not create a data output data set, but is rather writes external files and other SAS code that follows its execution. The LENGTH statement (Line 11) creates three variables *Query*, *BLASTn*, and *Seq* that are character variables with lengths 150, 1000, and 60 bytes, respectively. The dollar sign indicates that they will be character variables (the absence of which would mean that they would be numeric variables, the only other choice in SAS) and the number of bytes (length) follows the dollar sign. The SET statement (Line 12) reads the SAS data set to which the SAS Macro variable &Query_DS. resolves. This value is a macro parameter that the user must specify (Line 1). The next line is an assignment statement that creates the variable FV (Line 13), which SAS creates as a character variable since its value is the result of the CATS() function. This assignment names the text file, so the user must provide appropriate values of *Query* when he or she creates the data set. The value of FV is the FILEVAR= option to the FILE statement (Line 14); when the value of FV changes, SAS dynamically close the currently opened output file and changes to a new physical file.

The FILE statement (Line 14) instructs the data step to write to the external files, but by virtue of the FILEVAR= option, the filename Q is simply a placeholder. *FV* specifies the text file. The PUT statement in the next line (Line 15) writes the literal "> " and the value of *Query* to the text file. Since the format of FASTA implies a line length of up to 120 bytes and conventionally this is 60 or 80 bytes, the Do-Loop (Lines

16-19) calculates the number of times the *Sequence* must wrap. The use of the CEIL() guarantees the needed number of lines; this function returns the next largest integer functions: for instance, if the sequence length was 61 bp, the number of lines would be 2 (61/60 = 1.0167, the next largest integer is 2). The starting index for the SUBSTR() function would be 1, 61, 121 corresponding to the strings 1-60, 61-120,121-180, *et cetera*. With each iteration of this Do-Loop, the substring is written to the text file (Line 18).

The value of *BLASTn* is assigned in Lines 20-29. Importantly, this value is the argument to the CALL EXECUTE() call routine, which effectively writes SAS code upon execution of the data step in which it is called. The SAS code thus written is executed after the execution of the completion of the execution of the data step in which was called has completed. The value of *BLASTn* is the result returned from the CAT() function with strings, SAS functions, or SAS variables as arguments, with the exceptions of the macro variable &OPTION. Notably, no comma proceeds this variable (Line 27). Instead, the user must provide the comma in the value he or she writes for the parameter. If so, the he or she must "hide" the comma from the macro parser. One way would be to use the %STR() function (Line 5). Once the FASTA file is written and the value of BLASTn is assigned, the data step uses the CALL EXECUTE() routine (Line 30) and the data step either implicitly loops or the data steps terminates (Line 31).

By virtue of the XSNYC option (Line 8), SAS does not continue until the Windows operation system returns control to it. The system call via the X statement runs the BLASTn program, which writes to the output files specified in line 26. Note that the user must ensure that the values of *Query* in &Query_DS. are unique. Upon receiving control, SAS then executes the data step that reads the BLASTn output (text) files (Line 34-153).

This data step creates a temporary SAS data set named blastn. Between calls of this macro, the user must save this data step or it will be overwritten. Use of the APPEND procedure is one method to save the results. The data set option DROP= (Line 34) instructs SAS to not write the listed variables to the output data set. The effected variables are *Line*, explicitly listed, and any variable beginning with "__", by virtue of the colon following "__".

A series of variables found in the results are listed in the LENGTH statement (Line 35). Depending on the project, the user may wish to increase the size of the "sequence" variables (Line 37). If so, the user may wish to convert this constant to a macro parameter. Lines 43-51 create a series of "temporary" variables that hold the return codes (RC) for the PRXParse() functions. These functions compile the perl regular expression and return the numeric pattern identifier that can then be the argument to subsequent PRX functions. Note that in other data step structures, the user may wish to RETAIN these values if the data step loops. This data step implicitly loops once, but rather uses Do-Until Loops to read the entire data set and the subsequent entire raw text files.

The Do-Until Loop (Lines 53-151) iterates until the last observation of the SAS data set specified in the macro variable specified in the parameter Query_DS is read. This is indicated by the END= option to the SET statement (Line 54), which SAS assigns as zero (0) until it reads the last observation of the data set. The Query sequence may align multiple times within the Subject sequence. If so, it is important to track the number of times. It may also not align at all. The variable *Hit* is initialized to zero (0) to count the number of times (Line 55), as indicated by the number of times the term "Score =" is encountered in the output.

The value of *Query* is again used but this time to create the name of the input raw data file. Lines 13 and 56 differ only in their third argument to the CATX() function, "fas" and "blastn", respectively. The INFILE statement (Line 57) then uses the value of *FV* as the argument to the FILEVAR= option and IF is again a place holder. Two additional options to the INFILE statement appear, END= and LENGTH=. The former is an indicator variable of the last record of the external file and the latter stores the line size for the variable record length raw text file.

The Do-Until Loop (Line 58-139) reads the external text file until the last record. Each line of the external text file is read (Line 59) and parsed for the keyword in the BLAST output. Although the BLAST output does not track the number of hits, the number of times that they are encountered is used. An important aspect of genetics is the number of repetitive sequences that might be found (LINEs and SINEs, for instance). One should not be surprised to find exact sequences of length 30 or even 50 that match in both orientations multiple times in a gene. If one has a reference amplicon, one can determine the length of shortest sequence that matches uniquely within it[2]. Using this, one may provide an option to the BLASTn program via the BLASTn macro parameter Option.

Due to the explicit looping of the data step, the values of the variables should be reset to missing manually. The CALL MISSING routines (Lines 64 and 143) perform this task. After all lines of the text files are read and for each observation in the data set, the data step ends (Line 153).

The user must decide with to keep the text files created by this macro. If the names of the files are unique, then it may be sensible to keep them as they are typical small. If the user chooses to delete them, however, setting the macro parameter FDelete to 1 (Line 5) will allow the macro to write the SAS code in lines 160-169. The effect of these lines is to delete the query FASTA file and its associated output from the BLASTn program.

DISCUSSION

As we learn more about the role of genomics in health outcomes, the SAS programmer may be required to moonlight as a bioinformatician to obtain the necessary data for analyses. Many bioinformatics programs already exist and calling external programs from SAS is common. This paper described a macro that builds a "command line" for the BLASTn program available in the stand alone suite of BLAST programs from the NCBI. BLAST is a very popular alignment algorithm with a wealth of publications to support it. BLAST Documentation is available from this website and the email help desk to is very informative and prompt (blast-help@ncbi.nlm.nih.gov).

This macro also provides the ability to parse the contents of the output and creates a temporary SAS dataset. This data set can be used to enter data into a genomics database or the user can parse the output files using a custom written approach. The macro can handle single or multiple test (query) or reference (subject) sequences with a single use of the macro, given the appropriate structure of the SAS data set.

The use of this macro in investigations that generate DNA sequence files, such as Sanger Sequence projects, is rapid and the result allow for the easy creation of a genomics database that is auditable and includes an entry for every target base in the sequence. This approach is complimentary to the matching of flanks approach I previously reported, but has two advantages. The first is that BLAST is extremely popular and recognized. The results are easy to explain. The second is that this approach handles deletions much more efficiently than attempts to match by flanks, even if one secondarily uses the data from the database instead of attempting to construct the flanks from the reference sequence.

ACKNOWLEGEMENTS

This work was supported in part by the following grants: NIH-5RC2HL101851-02, NIH-5R01HL072533-04, and NIH-5K08HL071130-05.

I am indebted to a great number of people who have provided sequencing projects, criticism, and information. I would like to thank Tom Howard of the Greater Los Angeles Veteran Administration Hospital and Shelley Cole, Margie Britten, and Vanessa Ayala of the Texas Biomedical Research Institute (formerly the Southwest Foundation for Biomedical Research) for collaborative efforts on projects and input concerning files and process. I would like to thank Tao Tao of the NCBI User Services for answering questions and providing guidance. Finally, I am indebted to the countless members of both the SAS-L (http://www.listserv.uga.edu/archives/sas-l.html) and ABRF (www.abrf.org) listservs.

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Name: Kevin Viel
Enterprise: Histonis, Incorporated
City, State ZIP: Atlanta, GA 30342
E-mail: kviel@histonis.org, genepistat@gmail.com
Web: www.histonis.org

Figure 2. The BLASTn macro.

```
%Macro BLASTn ( Query_DS =
              , Subject  =        /* C:\F8.fas                           */
              , Out_DS   =        /* BLASTn                              */
              , Option   = %Str() /* "%Str(,) -evalue 1000 -word_size 7" */
              , FDelete  = 1
              ) ;

  Options XSync NoXWait XMin ;

  Data _null_ ;
    Length Query $ 150 BLASTn $ 1000 Seq $ 60 ;
    Set &Query_DS. ;
    FV = CatS( "C:\" , CatX( "." , Query , "fas" )) ;
    File Q Filevar = FV ;
    Put "> " Query ;
    Do I = 1 To Ceil( Length( Sequence ) / 60 ) ;
      Seq = SubStr( Sequence , 1 + ( I - 1 ) * 60 ) ;
      Put Seq ;
    End ;
    BLASTn = Cat( "X 'blastn "
                , " -subject "
                , Strip( "&Subject." )
                , " -query "
                , Strip( FV )
                , " -out "
                , Strip( TranWrd( FV , ".fas" , ".blastn" ))
                &Option.
                , "' ;"
                ) ;
    Call Execute( Blastn ) ;
  Run ;

  /*****************/
  Data blastn ( Drop = __: Line ) ;
    Length Query $ 150 Query_Length 8 Subject $ 60 Subject_Length 8 Score $ 30 Expect 8
           Identities $ 16 Gaps $ 16 Strand $ 11
           Query_Sequence Match_Sequence Subject_Sequence $ 1500
           Line $ 100
           Lambda K H Gapped_Lambda Gapped_K Gapped_H Effective_search_space_used 8
           Matrix $ 50 Gap_Penalties_Existence Gap_Penalties_Extension 8
           FV $ 300
           ;
    __RC_Query_Line = PRXParse( "/Query\s*(\d+)\s*([A-Za-z\-])+\s*(\d+)/o" ) ;
    __RC_Query      = PRXParse( "/Query=\s*(.*)/o"                         ) ;
    __RC_Length     = PRXParse( "/Length=\s*(.*)/o"                        ) ;
    __RC_Subject    = PRXParse( "/Subject=\s*(.*)/o"                       ) ;
    __RC_Score      = PRXParse( "/Score =\s*(.*),/o"                       ) ;
    __RC_Expect     = PRXParse( "/Expect =\s*(.*)/o"                       ) ;
    __RC_Identities = PRXParse( "/Identities =\s*(.*),/o"                  ) ;
    __RC_Gaps       = PRXParse( "/Gaps =\s*(.*)/o"                         ) ;
    __RC_Strand     = PRXParse( "/Strand=\s*(.*)/o"                        ) ;

    Do Until ( End_Query_DS ) ;
      Set &Query_DS. End = End_Query_DS ;
      Hit = 0 ;
      FV = CatS( "C:\" , CatX( "." , Query , "blastn" )) ;
      Infile IF FileVar = FV End = End_Infile Length = Len ;
      Do Until (End_Infile ) ;
        Input Line $Varying100. Len ;
        If Hit > 0 And PRXMatch ( __RC_Score , Line )
        Then
          Do ;
             Output ;
             Call Missing ( Score , Expect , Identities , Gaps , Strand , Query_Sequence
                          , Match_Sequence , Subject_Sequence , Gapped_Lambda , Gapped_K
                          , Gapped_H , Effective_search_space_used , Matrix
                          , Gap_Penalties_Existence , Gap_Penalties_Extension
                          , Query_Start , Subject_Start
                          ) ;
          End ;

        If PRXMatch ( __RC_Query , Line ) Then Query = PRXPosN( __RC_Query , 1 , Line ) ;
        If PRXMatch ( __RC_Length , Line ) And Query_Length = .
        Then Query_Length = Input( PRXPosN( __RC_Length , 1 , Line ) , 8. ) ;
          Else If Subject_Length = .
          Then Subject_Length = Input( PRXPosN( __RC_Length , 1 , Line ) , 8. ) ;
        If PRXMatch ( __RC_Subject , Line )
        Then Subject = PRXPosN( __RC_Subject , 1 , Line ) ;
        If PRXMatch ( __RC_Score , Line )
        Then
          Do ;
             Score = PRXPosN( __RC_Score , 1 , Line ) ;
             Hit + 1 ;
          End ;

        If PRXMatch ( __RC_Expect , Line )
        Then Expect = Input( PRXPosN( __RC_Expect , 1 , Line ) , 8. ) ;
        If PRXMatch ( __RC_Identities , Line )
        Then Identities = PRXPosN( __RC_Identities , 1 , Line ) ;
        If PRXMatch ( __RC_Gaps , Line ) Then Gaps = PRXPosN( __RC_Gaps , 1 , Line ) ;
        If PRXMatch ( __RC_Strand , Line ) Then Strand = PRXPosN( __RC_Strand , 1 , Line ) ;
        If Line = "***** No hits found *****"
        Then
          Do ;
             Query_Sequence   = "***** No hits found *****" ;
             Match_Sequence   = "***** No hits found *****" ;
             Subject_Sequence = "***** No hits found *****" ;
          End ;
        If Line =: "Query "
        Then
          Do ;
             If Query_Start = . Then Query_Start = Input( Scan( Line , 2 , " " ) , 8. ) ;
             Query_Sequence = CatS( Query_Sequence , Scan( Line , 3 , " " )) ;
             Input Line $Varying100. Len ;
             Match_Sequence = Cats( Match_Sequence , Line ) ;
             If Line ~ =: " " Then Put Line= ;
             Input Line $Varying100. Len ;
             If Subject_Start = .
             Then Subject_Start = Input( Scan( Line , 2 , " " ) , 8. ) ;
             Subject_Sequence = Cats( Subject_Sequence , Scan( Line , 3 , " " )) ;
             If Line ~ =: "Sbjct" Then Put Line= ;
          End ;
          If Line =: "Lambda"
          Then
            Do ;
               Input Line $Varying100. Len ;
               Lambda = Input( Scan( Line , 1 , " " ) , 8. ) ;
               K      = Input( Scan( Line , 2 , " " ) , 8. ) ;
               H      = Input( Scan( Line , 3 , " " ) , 8. ) ;
            End ;
          If Line =: "Gapped"
          Then
            Do ;
               Input Line $Varying100. Len ;
               Input Line $Varying100. Len ;
               Gapped_Lambda = Input( Scan( Line , 1 , " " ) , 8. ) ;
               Gapped_K      = Input( Scan( Line , 2 , " " ) , 8. ) ;
               Gapped_H      = Input( Scan( Line , 3 , " " ) , 8. ) ;
            End ;
          If Line =: "Effective search space used:"
          Then Effective_search_space_used = Input( Scan( Line , 2 , ":" ) , 8. ) ;
          If Line =: "Matrix:" Then Matrix = Strip( Scan( Line , 2 , ":" )) ;
          If Line =: "Gap Penalties"
          Then
            Do ;
               Gap_Penalties_Existence = Input( Scan( Line , 3 , ":," ) , 8. ) ;
               Gap_Penalties_Extension = Input( Scan( Line , -1 , ":" ) , 8. ) ;
            End ;
      End ; /* End_Infile */

      Output ;

      Call Missing ( Query , Query_Length , Subject , Subject_Length , Score , Expect
                   , Identities , Gaps , Strand , Query_Sequence , Match_Sequence
                   , Subject_Sequence , Line , Lambda , K , H , Gapped_Lambda , Gapped_K
                   , Gapped_H , Effective_search_space_used , Matrix
                   , Gap_Penalties_Existence , Gap_Penalties_Extension
                   , Query_Start , Subject_Start
                   ) ;

    End ; /* End_Query_DS */

  Run ;

  /************/
  %If &FDelete. = 1
  %Then
      %Do ;
          Data _null_ ;
            Set &Query_DS. ;
            FV = CatS( "C:\" , CatX( "." , Query , "fas" )) ;
            RC = FileName( "FName" , FV ) ;
            If RC = 0 And FExist( "FName" ) Then RC2 = FDelete( "FName" ) ;
            RC = FileName( "FName" ) ;
            FV = CatS( "C:\" , CatX( "." , Query , "blastn" )) ;
            RC = FileName( "FName" , FV ) ;
            If RC = 0 And FExist( "FName" ) Then RC2 = FDelete( "FName" ) ;
            RC = FileName( "FName" ) ;
          Run ;
      %End ;

%MEnd BLASTn ;
```

Figure 3. The C2AA format.

```
Libname Library "C:\Documents and Settings\KViel\My Documents\SAS Formats" ;

Data C2AA ;
  Retain FmtName "C2AA" Type "c" ;
  Infile DataLines EOF = End ;
  Input Start : $3. Label : $4. @@ ;
  Output ;
  Return ;

  End:
    Do ;
       Start = " "   ;
       Label = "Err" ;
       HLO   = "O"   ;
       Output ;
       Stop ;
    End ;
DataLines ;
GCA A GCC A GCG A GCT A TGC C TGT C GAC D GAT D GAA E GAG E TTC F TTT F GGA G GGC G
GGG G GGT G CAC H CAT H ATA I ATC I ATT I AAA K AAG K CTA L CTC L CTG L CTT L TTA L
TTG L ATG M AAC N AAT N CCA P CCC P CCG P CCT P
TAA stop TAG stop TGA stop
CAA Q CAG Q AGA R AGG R CGA R CGC R CGG R CGT R AGC S AGT S TCA S TCC S TCG S TCT S
ACA T ACC T ACG T ACT T GTA V GTC V GTG V GTT V TGG W TAC Y TAT Y
;
Run ;

Proc Format Library = Library CntlIn = C2AA ;
Run ;
```

REFERENCES

1. *SAS® 9.1.3 Language Reference: Dictionary*. Fifth ed. Copyright © 2002–2006: SAS Institute Inc., Cary, NC, USA. All rights reserved.
2. Viel, K., *Creating reference amplicons and genotyping using the SAS System.* PharmaSUG 2011 Proceedings, 2011.
3. Altschul, S.F., et al., *Basic local alignment search tool.* J Mol Biol, 1990. **215**(3): p. 403-10.
4. Camacho, C., et al., *BLAST+: architecture and applications.* BMC Bioinformatics, 2009. **10**: p. 421.
5. Viel, K.R., et al., *Inhibitors of factor VIII in black patients with hemophilia.* N Engl J Med, 2009. **360**(16): p. 1618-27.
6. Viel, K.R., et al., *A sequence variation scan of the coagulation factor VIII (FVIII) structural gene and associations with plasma FVIII activity levels.* Blood, 2007. **109**(9): p. 3713-3724.
7. El-Maarri, O., et al., *Lack of F8 mRNA: a novel mechanism leading to hemophilia A.* Blood, 2006. **107**(7): p. 2759-65.