PharmaSug 2018 - Paper 20

Aligning DNA Sequences with SAS FCMP and SAS Viya

Emily (Yan) Gao, SAS Institute Inc.;
Doris Feng, McMaster University;
Jinchao Di, SAS Institute Inc.

ABSTRACT

DNA sequence alignment is an essential aspect of bioinformatics although its complex computation is an ongoing issue. We show users how SAS FCMP and SAS Viya work together to provide an efficient solution. SAS FCMP makes dynamic programming — a powerful method used to compute the best alignment — a practical option in SAS; SAS Viya runs in parallel across many machines to increase the computational speed. In this paper, we demonstrate that SAS is able to find the best similarity score, alignment and backtrace for ten thousand pairwise sequences in just a few minutes.

BASICS OF SEQUENCE ALIGNMENT

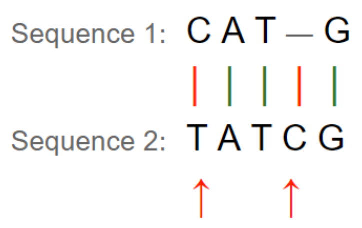
WHAT IS SEQUENCE ALIGNMENT?

Sequence alignment is a technique used to align strands of DNA, RNA or protein within samples or against a reference to identify regions which are the same or similar. These regions may be originated from the functional, structural, or evolutionary relationships that exist among samples, organisms or species — indicating significance. It can pinpoint genes which contribute to some genetic conditions, like autistic spectrum disorder or discover homologous genes between different organisms. It can also identify proteins which may play similar functional roles in the same biological process or signalling pathways. (Baral, 2003). Sequence alignment lays the foundation for systematic sequence analysis, which plays a pivotal role in the bioinformatics pipeline.

HOW ARE SEQUENCES ALIGNED?

A program aligns the strands as closely as possible by finding the dissimilarity between corresponding residues to determine the number of modifications necessary to match them. The alterations may occur as substitution, insertion and deletion which subtracts a penalty while matches add a reward to the total score. The score decreases with each mismatch and increases with each match; a score of -14 represents more modifications than a score of 55. (Armstrong, 2008)

EX. HOW WILL 'CATG' AND 'TATCG' BE ALIGNED?



In order to match sequence 1 to sequence 2, substitute 'C' for 'T' and insert a 'C'.

WHAT SEQUENCE ALIGNMENT DO WE FOCUS ON?

There are different types of alignment problems, and of those we will focus on global and pairwise alignments. Global alignments match every residue in one sequence to another, so it works the best when it's applied to strands which have a similar corresponding residue and length. Pairwise alignments match two strands at a time. In essence, our code focus on alignment problems where every residue is matched between two sequences. (Mount, 2006;Tumanyan, Roytberg, Polyvanovsky, 2011) In our example, we align ten thousand pairs of DNA sequences simultaneously. Each sequence is 500 residues long.

If users wish to process longer sequences such as the human genome, our code provides a great foundation. Our APPENDIX section provides a starting guide.

WHY IS SEQUENCE ALIGNMENT DIFFICULT?

The best sequence alignment is difficult to find due to the considerable number of possibilities. Consider the formula below.

For two sequences of length n, this formula shows the number of possible alignments.

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

Table 1: The number of possible alignments for pairwise sequences of length 10 and 100 are shown below. As you see, two sequences with just 10 residues have 184,756 possible alignments.

n	enumeration	Dynamic Programming
10	184,756	100
100	9.00E+58	10,000

DYNAMIC PROGRAMMING

To simplify the problem, we employ dynamic programming, which performs a much better job at aligning the sequences. It's a method used to resolve complex problems by breaking it into simpler subproblems and solving these recursively. Partial solutions are saved in a big table, so it can be quickly accessed for succeeding calculations while avoiding repetitive work. Through this process of building on each preceding result, we eventually solve the original, challenging problem efficiently. Many difficult issues can be resolved using this method. (Kellis, 2005)

SEQUENCE ALIGNMENT AND SAS

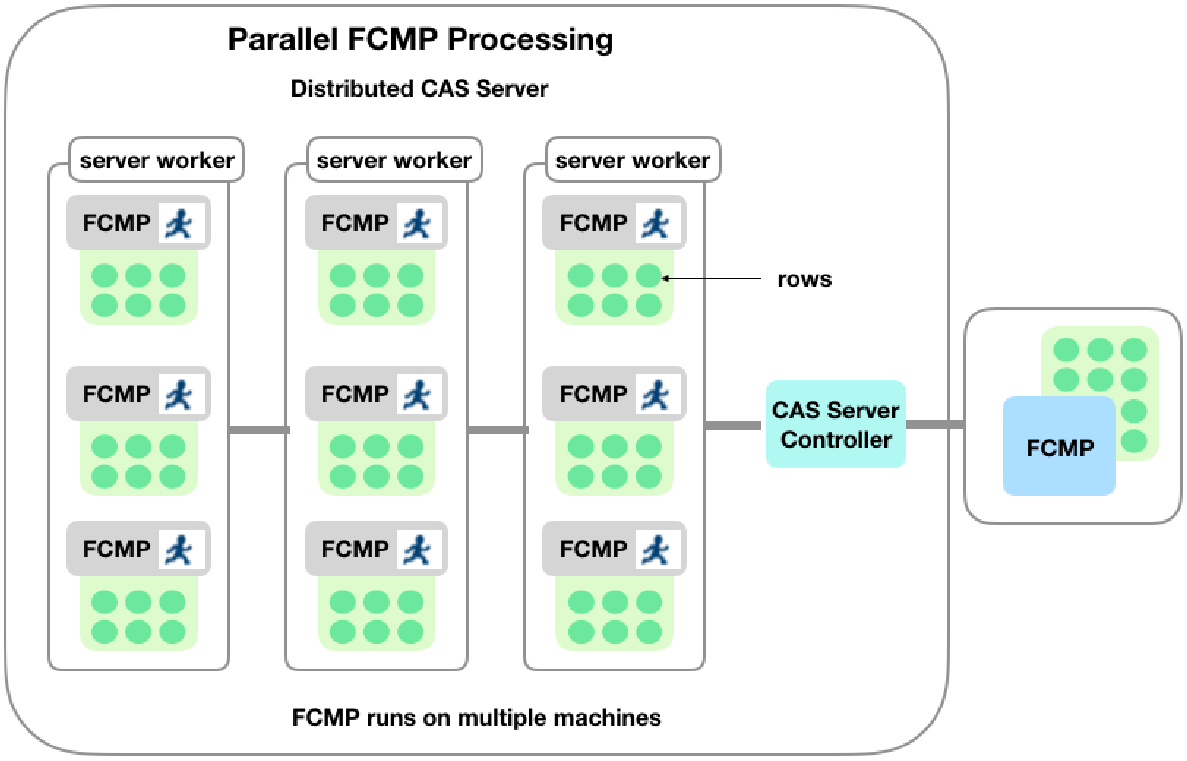
Two fundamental components in SAS will streamline your alignment calculations: SAS FCMP and SAS Viya. With these, the best similarity score and alignment for a ten thousand pairwise alignments can be found within minutes.

SAS FCMP BACKGROUND

SAS comes with some predefined general functions and call routines, but it may still be too generic for each user's needs. FCMP rectifies this by providing the option to create custom functions and subroutines and also allows users to define it once and simply call it later on. This makes dynamic programming a practical option to use by creating the custom function and reducing the repetitive work. Without FCMP, the code must be completely written every time we want to use dynamic programming, making it cumbersome and difficult to use. FCMP allows us to bring the computational powers of dynamic programming to SAS.

SAS VIYA BACKGROUND

Since sequence alignments are computation-intensive tasks, running the data in a single machine would be impractically slow. When it is run in CAS, computations run in parallel across different machines which significantly improves the speed. This computational power makes it possible to process a large quantity of sequences at once. In our example, ten thousand pairwise sequences are processed in minutes. (Sober, 2018)



DEFINING SEQUENCE ALIGNMENT COMPUTATIONALLY

RECURSIVE RELATION

- S is the similarity score matrix
- sub is the substitution matrix

$$S(m, n) = \max \begin{cases} S(m-1, n-1) + sub(x_i, y_i) \rightarrow \text{substitution or match (refer to substitution matrix)} \\ S(m-1, n) - \text{linear gap penalty} \rightarrow \text{deletion} \\ S(m, n-1) - \text{linear gap penalty} \rightarrow \text{insertion} \end{cases} \quad \textbf{(1)}$$

SCORE ASSIGNMENT

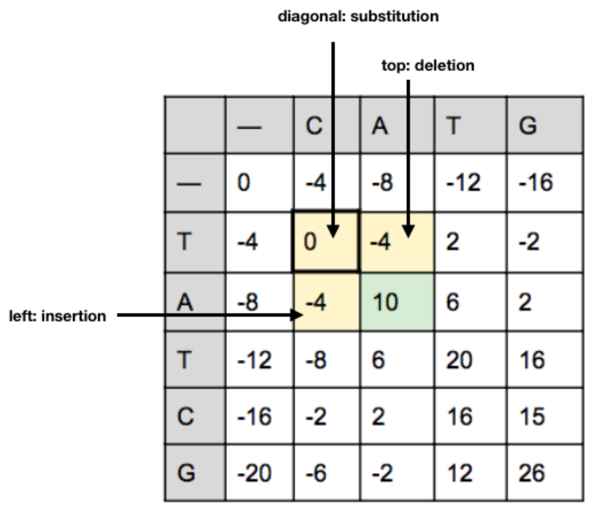
We will use the following score assignment in our example.

- Insertions or deletions carry a score penalty of -4
- Matches carry a score reward of +10
- This substitution matrix outlines the score penalty between different bases

	A	C	G	T
A	10	-5	0	-5
C	-5	10	-5	0
G	0	-5	10	-5
T	-5	0	-5	10

Ex. If you're trying to match the bases 'G' and 'C', the pink boxes show the score penalty that will be added to the total score.

INTERMEDIATE STEPS


diagonal: substitution
top: deletion
left: insertion

	—	C	A	T	G
—	0	-4	-8	-12	-16
T	-4	**0**	-4	2	-2
A	-8	-4	10	6	2
T	-12	-8	6	20	16
C	-16	-2	2	16	15
G	-20	-6	-2	12	26

Table 2: It will note the values of the previous step (yellow boxes) before it calculates the score to the current step (green box) for all three options. The present stage will build off whichever option produces the maximum score, and in this case it's substitution (bolded element). Sometimes, more than one option will provide the maximum score.

The score in the green box represents the best alignment thus far; in this case it's the best alignment between 'CA' and 'TA'.

	—	C	A	T	G
—	0	-4	-8	-12	-16
T	-4	0	-4	2	-2
A	-8	-4	10	6	2
T	-12	-8	6	20	16
C	-16	-2	2	16	15
G	-20	-6	-2	12	26

Table 3: The green element in this shows the best alignment score between the two sequences, while the yellow element show the backtrace to this solution.

The backtrace shows which path yields the maximum score to an element.

The backtrace matrix builds off the results found in the similarity score matrix.

USING SAS TO CALCULATE THE SEQUENCE ALIGNMENT

THE THREE WRAPPER FUNCTIONS

In the full code, we have three wrapper functions: getSeqSimScore, getBackTrace and sequenceAlignment. Each of these wrappers will output a different part of the result depending on your analysis needs. For our summary below, we will only introduce the getSeqSimScore wrapper to improve the length and clarity of the paper. The two other wrappers have a similar seven step process, so the explanations can be applied to them. The full code including the other two wrappers are located at the end of the paper.

getSeqSimScore

This will find the score for the best alignment between two sequences.

sequenceAlignment

This will find the score for the best alignment just like getSeqSimScore, and it will also show you the locations where modifications should be made within each sequence.

getBackTrace

This will find the similarity score just like getSeqSimScore, and it will also compute the backtrace matrix (See Table. 2) for users interested in viewing the intermediate step.

CODE OVERVIEW

DEFINE THE WRAPPER FUNCTION

```
function getSeqSimScore(sequence1 $, sequence2 $);
  array score[1,1]/nosymbols;
  seq1_len = length(sequence1);
  seq2_len = length(sequence2);

  seq1_len_plus_1 = seq1_len+1;
  seq2_len_plus_1 = seq2_len+1;
  call dynamic_array(score, seq1_len_plus_1, seq2_len_plus_1);
```

PREPARE THE INITIAL STATES

```
  do i=1 to seq1_len_plus_1;
    do j=1 to seq2_len_plus_1;
      score[i, j]=9999;
    end;
  end;
```

CALL THE RECURSIVE FUNCTION

```
  final_score = seqSimRecursive(sequence1, sequence2, score, seq1_len, seq2_len);
```

RETURN THE FINAL SCORE

```
  return (final_score);
endsub;
```

SET UP THE COMPUTATIONAL STEP

```
function seqSimRecursive(sequence1 $, sequence2 $, score[*,*], m, n);
  outargs score;
```

DEFINE THE RETURN CONDITION

```
  if score[m+1,n+1] < 9999 then
    return (score[m+1,n+1]);

  linear_gap_penalty = -4;
```

COMPUTE THE SIMILARITY SCORE

```
  if m=0 and n=0 then score[m+1,n+1]=0;
  else if m=0 then score[m+1,n+1]=n*linear_gap_penalty;
  else if n=0 then score[m+1,n+1]=m*linear_gap_penalty;
  else do;
    sub_score = getSubstitutionScore(substr(sequence1, m, 1), substr(sequence2, n, 1));
    diag = seqSimRecursive(sequence1, sequence2, score, m - 1, n - 1) + sub_score;
    up = seqSimRecursive(sequence1, sequence2, score, m - 1, n) + linear_gap_penalty;
    left = seqSimRecursive(sequence1, sequence2, score, m, n - 1) + linear_gap_penalty;
    score[m+1,n+1] = max(diag,up,left);
  end;

  return (score[m+1,n+1]);
endsub;
```

SUMMARY OF THE IMPORTANT SECTIONS

1) DEFINE THE WRAPPER FUNCTION

Summary: Introduces a wrapper function. The wrapper calls a second subroutine to do the logic recursively, which eliminates the details and simplifies it for end users.

```
function getSeqSimScore(sequence1 $, sequence2 $);
```

sequence1 & sequence2 → Parameters which represent the two sequences used.

2) PREPARE THE INITIAL STATES

Summary: This computational stage initializes the similarity score matrix.

```
do i=1 to seq1_len_plus_1;
   do j=1 to seq2_len_plus_1;
```

We are using a two-layer loop to accommodate the two-dimensional arrays.

```
   score[i, j]=9999;
   end;
end;
```

We are giving all the elements on the similarity score matrix a generic score of 9999 at first. Once our sequences are processed, it will start to generate a similarity score for each element, and it will be smaller than 9999. In our code, the length of a sequence after modifications is up to 1000. The placeholder value of 9999 will work for every case except when every single residue in a 1000 long sequence are exactly the same. In order to align this or longer sequences, simply increase the placeholder value so that the results will be included.

Score[i, j] → Two-dimensional array used to save the score of all sequences.

3) CALL THE RECURSIVE FUNCTION

```
final_score = seqSimRecursive(sequence1, sequence2, score, seq1_len, seq2_len);
```

This function is used to solve the problems recursively. The variable 'score' saves the similarity scores of previous steps so it can be easily accessed to build the answers for succeeding ones.

4) RETURN THE FINAL SCORE

Summary: Returns the final similarity score between two input sequences.

```
return (final_score);
```

5) SET UP THE COMPUTATIONAL STEP

Summary: Defines the recursive computational function.

```
function seqSimRecursive(sequence1 $, sequence2 $, score[*,*], m, n);
  outargs score;
```

Outargs → Tells the subroutine which argument from the argument list to be updated.

m → The index of sequence1.

n → The index of sequence2.

6) DEFINE THE RETURN CONDITION

Summary: This return condition is set so that if the value of score[m+1,n+1] is smaller than 9999, it will be returned. If score[m+1,n+1] still has the placeholder value of 9999, it will not be returned. It means the solutions do not exist yet so it will go look for it in section seven. This check avoids calculating the similarity score for each element more than once, which improves the performance.

```
if score[m+1,n+1] < 9999 then
    return (score[m+1,n+1]);
```

7) COMPUTE THE SIMILARITY SCORE

```
if m=0 and n=0 then score[m+1,n+1]=0;
  else if m=0 then score[m+1,n+1]=n*linear_gap_penalty;
  else if n=0 then score[m+1,n+1]=m*linear_gap_penalty;
```

Compute the similarity score when m=0 or n=0 or both equal zero.

```
 else do;
     sub_score = getSubstitutionScore(substr(sequence1, m, 1), substr(sequence2, n, 1));
     diag = seqSimRecursive(sequence1, sequence2, score, m - 1, n - 1) + sub_score;
     up = seqSimRecursive(sequence1, sequence2, score, m - 1, n) + linear_gap_penalty;
     left = seqSimRecursive(sequence1, sequence2, score, m, n - 1) + linear_gap_penalty;
     score[m+1,n+1] = max(diag,up,left);
  end;
```

This will take the maximum score from a match, substitution, deletion and/or insertion by using the formula from (1). It also specifies how the running similarity scores is affected by each change: does it increase or decrease, and by how much?

FULL CODE

```
/* Start CAS Server */
cas casauto host="host.example.com" port=5570;
libname sascas1 cas;

/* Define the sequence alignment function */
proc cas;
action sessionProp.setSessOpt / cmplib="CASUSER.seqFunc";
run;
loadactionset "fcmpact";
action addRoutines /
routineCode = {
"
function getSeqSimScore(sequence1 $, sequence2 $);
  array score[1,1]/nosymbols;
  seq1_len = length(sequence1);
  seq2_len = length(sequence2);

  seq1_len_plus_1 = seq1_len+1;
  seq2_len_plus_1 = seq2_len+1;
  call dynamic_array(score, seq1_len_plus_1, seq2_len_plus_1);
  do i=1 to seq1_len_plus_1;
     do j=1 to seq2_len_plus_1;
       score[i, j]=9999;
     end;
  end;

  final_score = seqSimRecursive(sequence1,  sequence2, score, seq1_len, seq2_len);

  return (final_score);
endsub;


function getBackTrace(sequence1 $, sequence2 $);
  array score[1,1]/nosymbols;
  array backtrace[1,1]/nosymbols;
  seq1_len = length(sequence1);
  seq2_len = length(sequence2);

  seq1_len_plus_1 = seq1_len+1;
  seq2_len_plus_1 = seq2_len+1;
  call dynamic_array(score, seq1_len_plus_1, seq2_len_plus_1);
  call dynamic_array(backtrace, seq1_len_plus_1, seq2_len_plus_1);

  do i=1 to seq1_len_plus_1;
     do j=1 to seq2_len_plus_1;
       score[i, j]=9999;
     end;
  end;

  backtrace[1, 1]=0;
  do i=2 to seq1_len_plus_1;
     backtrace[i, 1]=2;
  end;

  do j=2 to seq2_len_plus_1;
     backtrace[1, j]=3;
  end;

  final_score = backTraceRecursive(sequence1, sequence2, backtrace, score, seq1_len, seq2_len);

  return (final_score);
endsub;


function sequenceAlignment(sequence1 $, sequence2 $);
  outargs sequence1, sequence2;

  array score[1,1]/nosymbols;
  array backtrace[1,1]/nosymbols;

  seq1_len = length(sequence1);
  seq2_len = length(sequence2);
  seq1_len_plus_1 = seq1_len+1;
```

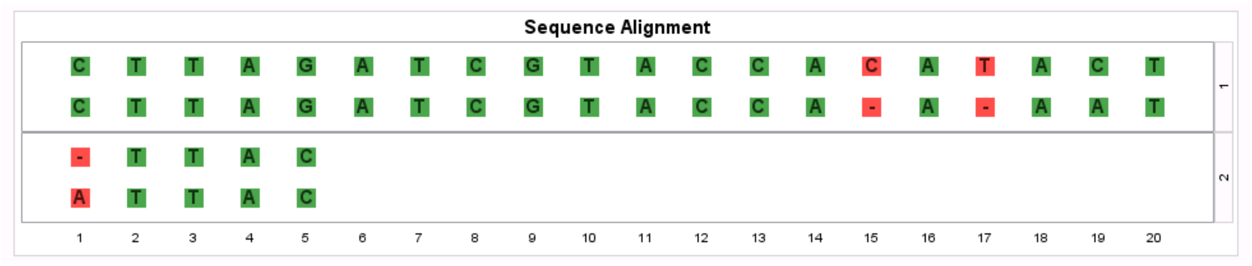
```
   seq2_len_plus_1 = seq2_len+1;
   call dynamic_array(score, seq1_len_plus_1, seq2_len_plus_1);
   call dynamic_array(backtrace, seq1_len_plus_1, seq2_len_plus_1);

   do i=1 to seq1_len_plus_1;
      do j=1 to seq2_len_plus_1;
         score[i, j]=9999;
      end;
   end;

   backtrace[1, 1]=0;
   do i=2 to seq1_len_plus_1;
      backtrace[i, 1]=2;
   end;

   do j=2 to seq2_len_plus_1;
      backtrace[1, j]=3 ;
   end;

   final_score = backTraceRecursive(sequence1, sequence2, backtrace, score, seq1_len, seq2_len);

   bt_m=seq1_len_plus_1;
   bt_n=seq2_len_plus_1;
   loop_max = max(seq1_len_plus_1, seq2_len_plus_1);
   length seq1_out $1000;
   length seq2_out $1000;
   seq1_out = '';
   seq2_out = '';
   do i=loop_max to 1 by -1;
      if backtrace[bt_m, bt_n] eq 1 then do;
         seq1_out = cats(seq1_out, substr(sequence1, bt_m-1, 1));
         seq2_out = cats(seq2_out, substr(sequence2, bt_n-1, 1));
         bt_m=bt_m-1;
         bt_n=bt_n-1;
      end;
      else if backtrace[bt_m, bt_n] eq 2 then do;
         seq1_out = cats(seq1_out, substr(sequence1, bt_m-1, 1));
         seq2_out = cats(seq2_out, '-');
         bt_m=bt_m-1;
      end;
      else if backtrace[bt_m, bt_n] eq 3 then do;
         seq1_out = cats(seq1_out, '-');
         seq2_out = cats(seq2_out, substr(sequence2, bt_n-1, 1));
         bt_n=bt_n-1;
      end;
   end;

   sequence1 = reverse(strip(seq1_out));
   sequence2 = reverse(strip(seq2_out));

   return (final_score);
endsub;
```

```
function charToNumber(char $);
   if char eq 'A' then i=1;
   else if char eq 'C' then i=2;
   else if char eq 'G' then i=3;
   else if char eq 'T' then i=4;

   return (i);
endsub;


function getSubstitutionScore(char1 $, char2 $);
   array sub_array[4,4];
   sub_array[1,1]=10;
   sub_array[1,2]=-5;
   sub_array[1,3]=0;
   sub_array[1,4]=-5;
   sub_array[2,1]=-5;
   sub_array[2,2]=10;
   sub_array[2,3]=-5;
   sub_array[2,4]=0;
   sub_array[3,1]=0;
   sub_array[3,2]=-5;
   sub_array[3,3]=10;
   sub_array[3,4]=-5;
   sub_array[4,1]=-5;
   sub_array[4,2]=0;
   sub_array[4,3]=-5;
   sub_array[4,4]=10;
   char1_index = charToNumber(char1);
   char2_index = charToNumber(char2);
   sub_score = sub_array[char1_index, char2_index];

   return (sub_score);
endsub;


function seqSimRecursive(sequence1 $, sequence2 $, score[*,*], m, n);
   outargs score;

   if score[m+1,n+1] < 9999 then
      return (score[m+1,n+1]);

   linear_gap_penalty = -4;

   if m=0 and n=0 then score[m+1,n+1]=0;
   else if m=0 then score[m+1,n+1]=n*linear_gap_penalty;
   else if n=0 then score[m+1,n+1]=m*linear_gap_penalty;
   else do;
      sub_score = getSubstitutionScore(substr(sequence1, m, 1), substr(sequence2, n, 1));
      diag = seqSimRecursive(sequence1, sequence2, score, m - 1, n - 1) + sub_score;
      up = seqSimRecursive(sequence1, sequence2, score, m - 1, n) + linear_gap_penalty;
      left = seqSimRecursive(sequence1, sequence2, score, m, n - 1) + linear_gap_penalty;
      score[m+1,n+1] = max(diag,up,left);
   end;

   return (score[m+1,n+1]);
endsub;


function backTraceRecursive(sequence1 $, sequence2 $, backtrace[*,*], score[*,*], m, n);
   outargs backtrace, score;

   if score[m+1,n+1] < 9999 then
      return (score[m+1,n+1]);

   linear_gap_penalty = -4;

   if m=0 and n=0 then score[m+1,n+1]=0;
   else if m=0 then score[m+1,n+1]=n*linear_gap_penalty;
   else if n=0 then score[m+1,n+1]=m*linear_gap_penalty;
   else do;
      sub_score = getSubstitutionScore(substr(sequence1, m, 1), substr(sequence2, n, 1));
      diag = backTraceRecursive(sequence1, sequence2, backtrace, score, m - 1, n - 1) + sub_score;
      up = backTraceRecursive(sequence1, sequence2, backtrace, score, m - 1, n) + linear_gap_penalty;
      left = backTraceRecursive(sequence1, sequence2, backtrace, score, m, n - 1) + linear_gap_penalty;
      ans = max(diag,up,left);
      score[m+1,n+1] = ans;

      if ans eq diag then do;
         backtrace[m+1,n+1]=1;
      end;
      else if ans eq up then do;
         backtrace[m+1,n+1]=2;
      end;
      else if ans eq left then do;
         backtrace[m+1,n+1]=3;
      end;
   end;

   return (score[m+1,n+1]);
endsub;
"
}
package  = "pkg"
saveTable = true
funcTable = {caslib="CASUSER" name="seqFunc" replace=true};
run;
quit;

/* Create a test data with 10K rows of sequence pairs */
/* and each pair has two sequences with 500 residues */
data sascas1.DNASequences;
   infile cards missover;
   input str1 : $500. str2 : $500.;
   length sequence1 $ 1000;
   length sequence2 $ 1000;
   sequence1=repeat(strip(str1),19);
   sequence2=repeat(strip(str2),19);
   do i=1 to 10000; output; end;
   drop str1 str2;
cards;
CTTAGATCGTACCAAAATATTAC  CTTAGATCGTACCACATACTTTAC
;
run;


/* Test functions */
proc cas;
action sessionProp.setSessOpt / cmplib="CASUSER.seqFunc";
run;
loadactionset "fcmpact";
runProgram
   inputData={name="DNASequences"}
   outputData={name="DNAsequenceAlignment" replace=True}
   routineCode={"sa_score=sequenceAlignment(sequence1, sequence2);"};
run;
quit;


/* Print alignments of the first pair */
proc print data=sascas1.DNAsequenceAlignment(where=(i=1));
run;quit;


/* Visualize alignment results */
%let n=100;
data graphdata;
   set sascas1.DNAsequenceAlignment(where=(i=1));
   i=0;
   group=1;
   do j=1 to length(sequence1);
      i+1;
      char1=substr(sequence1,j,1);
      char2=substr(sequence2,j,1);
      if char1='-' or char2='-' then char_color=1;
      else char_color=2;
      seq=1;
      char=char1;
      output;
      seq=2;
      char=char2;
      output;
      if mod(j,&n)=0 then do;
         i=0;
         group=group+1;
      end;
   end;
   call symputx('maxline',group);
```

```
run;

title h=15pt 'Sequence Alignment';
ods graphics on / height=&maxline.in width=15in;
proc sgpanel data=graphdata noautolegend;
  panelby group / layout=rowlattice novarname onepanel;
  scatter x=i y=seq /
    datalabel=char datalabelattrs=(color=black size=15pt weight=bold) datalabelpos=center
    markerAttrs=(symbol=squareFilled size=25)
    colorResponse=char_color colorModel=(red green) transparency=0.3;
    colaxis values=(1 to &n)
         display=(nolabel noticks)
         valueattrs=(size=10pt)
         offsetmin=0.05 offsetmax=0.05;
    rowaxis display=none;
Run;quit;
```

VISUAL ALIGNMENT RESULTS

The section of code under /* Visualize alignment results */ will produce the following results to show which area of the code will match or require modification.



CONCLUSION

Sequence alignment is an essential aspect of bioinformatics, and we show users an efficient way of obtaining the answers they would like by employing FCMP and Viya within SAS. The combination of computing power and best alignment will prove to be an invaluable tool for streamlining projects.

REFERENCES

Armstrong, D. (n.d.). *Sequence Alignment*. Reading. http://www.inf.ed.ac.uk/teaching/courses/bio2/lectures09/lecture5.pdf

Baral, C. (n.d.). *Sequence Alignment 1*. Lecture.

Kellis, M. (n.d.). *Sequence Alignment and Dynamic Programming*. Reading. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-096-algorithms-for-computational-biology-spring-2005/lecture-notes/lecture5_newest.pdf

Mount, D. W. (2006). *Bioinformatics: Sequence and genome analysis*. Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press.

SAS 9.4 Product Documentation. sas. Available at http://support.sas.com/documentation/94/

Sober, S. (n.d.). My Experiences in Adopting SAS® Cloud Analytic Services into Base SAS® Processes. Retrieved from https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1710-2018.pdf

Tumanyan, V., Roytberg, M., & Polyanovsky, V. (n.d.). Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3223492/.

CONTACT INFORMATION

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

Name: Emily (Yan) Gao
Phone: +86-10-83193355
Email: yan.gao@sas.com

Name: Doris Feng
Phone: + (613) 402-6988
Email: fengd2@mcmaster.ca

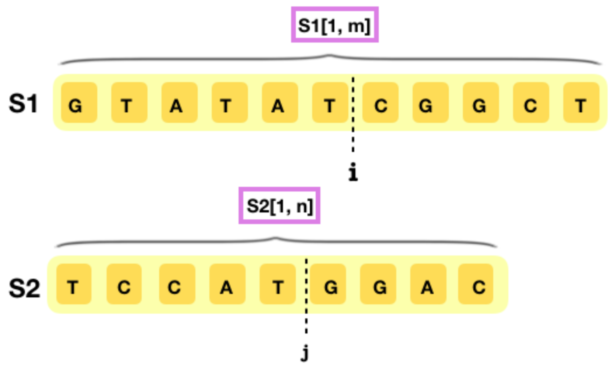
Name: Jinchao Di
Phone: +86-10-83193401
Email: jinchao.di@sas.com

APPENDIX

ADDITIVE PROPERTY

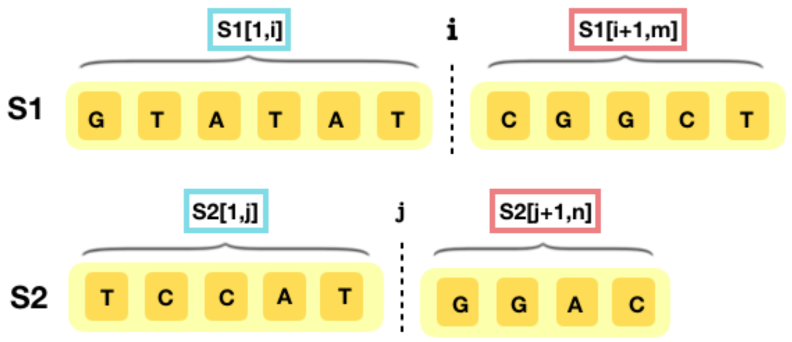
For sequence 1 (S1) of length m & sequence 2 (S2) of length n.

Figure 1



The best similarity score and best alignment for the entire sequence (Figure 1) is equal to the sum of adding up the best score for each section of the sequence (Figure 2).

Figure 2



In essence (Kellis, 2005):

best similarity score/alignment of $\boxed{\text{S1[1, m]}}$ and $\boxed{\text{S2[1, n]}}$ = best similarity score/alignment of $\boxed{\text{S1[1, i]}}$ and $\boxed{\text{S2[1, j]}}$ + best similarity score/alignment of $\boxed{\text{S1[i+1, m]}}$ and $\boxed{\text{S2[j+1, n]}}$

APPLYING THE ADDITIVE PROPERTY
In order to apply the additive property, each sequence must be snipped at a strategic location. The strategic location may not be at the same location in each sequence. For example, the first sequence may need to be snipped after five residues while the second need to be snipped after only three. In order to find this location, users can use Hirschberg's algorithm or FastDCA.