An Investigation on the Efficiency of Multiple Sequence Alignment Tools

Mid-Project Review

Justin Chao - juchao
Chase Meyer - cmeyer3
Bria Lacour - lacour

November 18, 2016

Project Goal

We will be developing a Multiple Sequence Alignment (MSA) program that uses a Markov Chain and a local alignment algorithm to optimize global alignments of protein sequences.

The Needleman-Wunsch algorithm will be used to calculate alignments using the BLOSUM 62 matrix as a scoring reference.

A comparison of run-times and accuracy will then be conducted between our developed MSA tool, BLAST, and MAFFT on a series of test sequences.

Background

A MSA tool can be used by computational biologists to track evolutionary relationships, predict structures, track mutations, etc. MSA tools come in a variety of forms and are tailored to address specific biological problems. These sequences are usually protein, DNA, or RNA, which can affect the methods of analysis in different ways.

The efficiency of these tools can affect scientific analyses, such as identifying significant patterns in protein families or assessing conservation of different genetic characteristics across species. The Needleman-Wunsch algorithm is used for global alignment purposes while the Smith-Waterman algorithm is used for aligning sub-sequences.

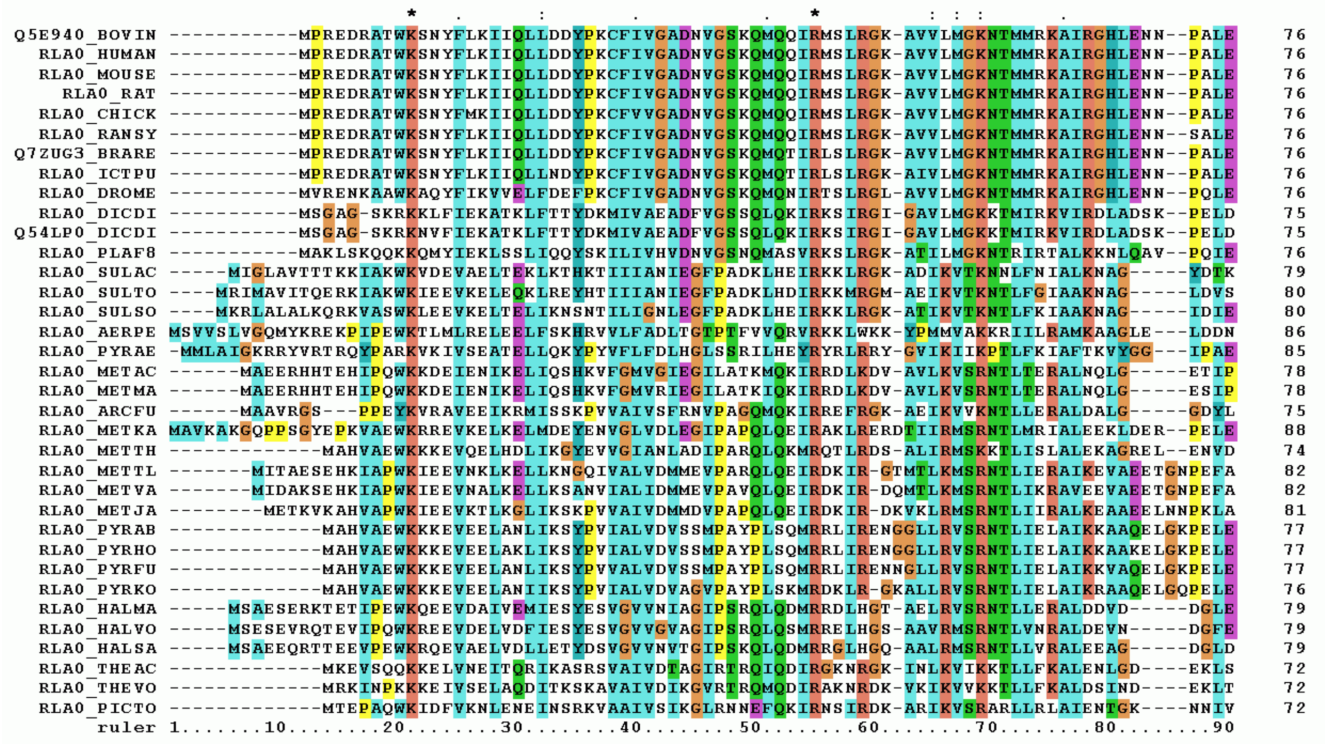

Figure 1: Representation of a protein multiple sequence alignment produced with ClustalW. The sequences are instances of the acidic ribosomal protein P0 homolog (L10E) encoded by the Rplp0 gene from multiple organisms. Only the first 90 positions of the alignment are displayed. The colors represent the amino acid conservation according to the properties and distribution of amino acid frequencies in each column. Note the two completely conserved residues arginine (R) and lysine (K) marked with an asterisk at the top of the alignment. [1]

Progress of Goals

- ☒ Code Needleman-Wunsch Algorithm.
- ☒ Code Smith-Waterman Algorithm.
- ☒ Find sample sequences for testing via NCBI.
- ☐ Parallelize algorithms.
- ☐ Calculate global alignment via Needleman-Wunsch.
- ☐ Calculate local alignments via Smith-Waterman.
- ☐ Use Markov chain to determine best local alignments.
- ☐ Align sequences using MAFFT, MUSCLE, Clustal Omega, and PRANK for comparison.
- ☐ Compare accuracy and run-time efficiency of results.
- ☐ Create visual diagram comparing alignment results of various MSA algorithms

Algorithms

The following are algorithms that will be used in the creation of our MSA tool. [2] [3]

Needleman-Wunsch Algorithm

$$\begin{aligned} M(0,j) &= j \times p \qquad \text{for first row, where p is the gap penalty} \\ M(i,0) &= i \times p \qquad \text{for first column} \end{aligned}$$

$$M(0,j) = max \begin{cases} M(i-1,j) + p & \text{top} \\ M(i,j-1) + p & \text{left} \\ M(i-1,j-1) + s(a_j, b_i) & \text{diagonal} \end{cases}$$

Where $s(a_j, b_i)$ = match/mismatch score for sites j and i in sequences a and b. [4]

Smith-Waterman Algorithm

$$\begin{aligned} M(0,j) &= j \times p \qquad \text{for first row} \\ M(i,0) &= i \times p \qquad \text{for first column} \end{aligned}$$

$$M(0,j) = max \begin{cases} 0 & \\ M(i-1,j) + p & \text{top} \\ M(i,j-1) + p & \text{left} \\ M(i-1,j-1) + s(a_j, b_i) & \text{diagonal} \end{cases}$$

Where $s(a_j, b_i)$ = match/mismatch score for sites j and i in sequences a and b. [4]

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix} \quad T = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ G & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow & \uparrow \\ C & 0 & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow \end{pmatrix}$$

Figure 2: Visual representation of Smith-Waterman algorithm. [5]

Code Examples

Needleman-Wunsch Algorithm

```
# nw_example.c

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "needleman_wunsch.h"

void align(char* seq_a, char* seq_b) {
   // Variables to store alignment result
   nw_aligner_t *nw = needleman_wunsch_new();
   alignment_t *result = alignment_create(256);

   // Decide on scoring
   int match = 1;
   int mismatch = -2;
   int gap_open = -4;
   int gap_extend = -1;

   // Don't penalise gaps at the start
   // ACGATTT
   // ----TTT would score +3 (when match=+1)
   char no_start_gap_penalty = 1;

   // ..or gaps at the end e.g.
   // ACGATTT
   // ACGA--- would score +4 (when match=+1)
   char no_end_gap_penalty = 1;

   char no_gaps_in_a = 0, no_gaps_in_b = 0;
   char no_mismatches = 0;

   // Compare character case-sensitively (usually set to 0 for DNA etc)
   char case_sensitive = 0;

   scoring_t scoring;
   scoring_init(&scoring, match, mismatch, gap_open, gap_extend,
              no_start_gap_penalty, no_end_gap_penalty,
              no_gaps_in_a, no_gaps_in_b, no_mismatches, case_sensitive);

   // Add some special cases
   // x -> y means x in seq1 changing to y in seq2
   scoring_add_mutation(&scoring, 'a', 'c', -2); // a -> c give substitution score -2
   scoring_add_mutation(&scoring, 'c', 'a', -1); // c -> a give substitution score -1

   // We could also prohibit the aligning of characters not given as special cases
   // scoring.use_match_mismatch = 0;

   needleman_wunsch_align(seq_a, seq_b, &scoring, nw, result);

   printf("seqA: %s\n", result->result_a);
   printf("seqB: %s\n", result->result_b);
```

```
   printf("alignment score: %i\n", result->score);

   // Free memory for storing alignment results
   needleman_wunsch_free(nw);
   alignment_free(result);
}

int main(int argc, char* argv[]) {
   if(argc != 3) {
      printf("usage: ./nw_example <seq1> <seq2>\n");
      exit(EXIT_FAILURE);
   }

   align(argv[1], argv[2]);
   exit(EXIT_SUCCESS);
}
```

Smith-Waterman Algorithm

```
# sw_example.c

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "smith_waterman.h"

void align(char* seq_a, char* seq_b)
{
   // Variables to store alignment result
   sw_aligner_t *sw = smith_waterman_new();
   alignment_t *result = alignment_create(256);

   // Decide on scoring
   int match = 1;
   int mismatch = -2;
   int gap_open = -4;
   int gap_extend = -1;

   // Don't penalise gaps at the start
   // ACGATTT
   // ----TTT would score +3 (when match=+1)
   char no_start_gap_penalty = 1;

   // ..or gaps at the end e.g.
   // ACGATTT
   // ACGA--- would score +4 (when match=+1)
   char no_end_gap_penalty = 1;

   char no_gaps_in_a = 0, no_gaps_in_b = 0;
   char no_mismatches = 0;

   // Compare character case-sensitively (usually set to 0 for DNA etc)
   char case_sensitive = 0;
```

```
  scoring_t scoring;
  scoring_init(&scoring, match, mismatch, gap_open, gap_extend,
               no_start_gap_penalty, no_end_gap_penalty,
               no_gaps_in_a, no_gaps_in_b, no_mismatches, case_sensitive);

  // Add some special cases
  // x -> y means x in seq1 changing to y in seq2
  scoring_add_mutation(&scoring, 'a', 'c', -2); // a -> c give substitution score -2
  scoring_add_mutation(&scoring, 'c', 'a', -1); // c -> a give substitution score -1

  // We could also prohibit the aligning of characters not given as special cases
  // scoring.use_match_mismatch = 0;

  smith_waterman_align(seq_a, seq_b, &scoring, sw);

  while(smith_waterman_fetch(sw, result))
  {
    printf("seqA: %s [start:%zu]\n", result->result_a, result->pos_a);
    printf("seqB: %s [start:%zu]\n", result->result_b, result->pos_b);
    printf("alignment score: %i\n\n", result->score);
  }

  // Free memory for storing alignment results
  smith_waterman_free(sw);
  alignment_free(result);
}

int main(int argc, char* argv[])
{
  if(argc != 3)
  {
    printf("usage: ./sw_example <seq1> <seq2>\n");
    exit(EXIT_FAILURE);
  }

  align(argv[1], argv[2]);
  exit(EXIT_SUCCESS);
}
```

---

[6]

Appendix

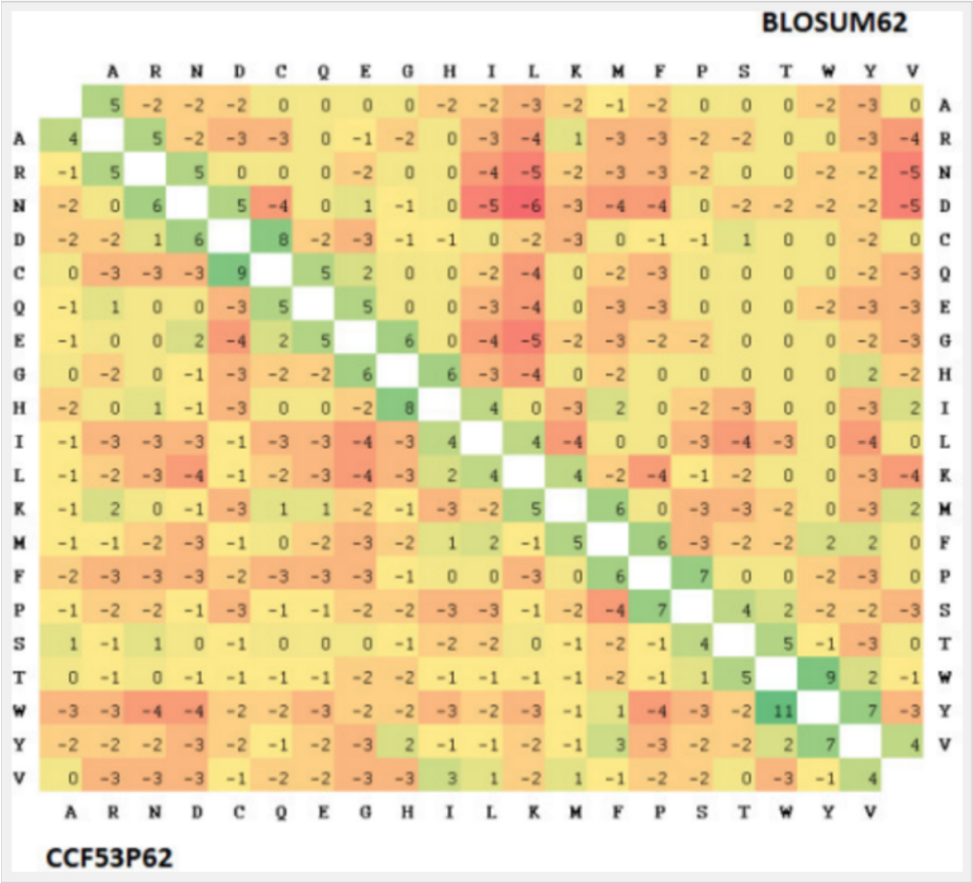

Figure 3: Comparison of BLOSUM62 and CCF53_62 matrices. The BLOSUM62 matrix is given on the upper right, while CCF53 is given on the lower left. Positive substitution scores are colored green, while negative scores are red. Deeper colors represent more positive/negative scores. [7]

MSA tool	Speed	Accuracy
MAFFT	very fast	medium-high
MUSCLE	very fast	medium
Clustal Omega	medium	medium-low
PRANK	very slow	very high

Table 1: Comparison of known MSA tools. Actual sequence alignment times and accuracies can vary depending on sequence size and scoring parameters.

References

[1] Multiple sequence alignment. *Wikipedia*, 2016.

[2] Claus O. Wilke. Aligning sequences. *The Wilke Lab*, 2016.

[3] Russell W. Hanson. Fast fourier transform analysis of dna sequences. *The Division of Mathematics and Natural Sciences, Reed College*, 2003.

[4] Yechiam Yemini. Blosum scoring matrices. *Computational Genomics*, 2007.

[5] Smith–waterman algorithm. *Wikipedia*, 2016.

[6] Isaac Turner. seq-align. *GitHub*, 2016.

[7] Pizzi E Brick K. A novel series of compositionally biased substitution matrices for comparing plasmodium proteins. *U.S. National Library of Medicine*, 2008.