


CS51 – Bioinformatics Draft Specification

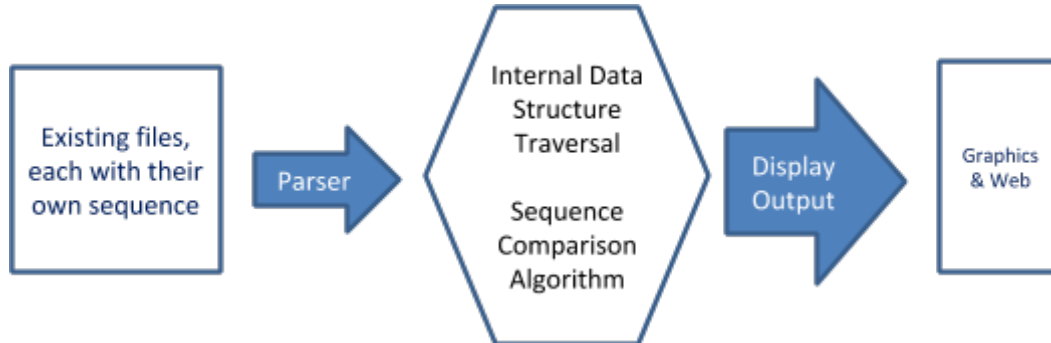
N. Katz & R. Burgess

April 20, 2014

Introduction: This is a proposal for a bioinformatics tool that can be used to compare genetic sequences and search for similar sequences based upon an entered sequence (via an industry standard files for storing sequences). We may focus specifically on sequences found in microbes that point to evolutionary relatedness.

Project will consist of 4 areas of development:

1. Developing a parser that takes existing files, each with their own DNA sequence, and uses them to build an internal data structure.
2. A data structure and a set of functions for traversing it.
3. A comparison algorithm for comparing an entered sequence with another sequence - either one within the data structure, or a second entered sequence.
4. A GUI for displaying an input field and the output results, ally existing in a browser.



Abstractions

Module Type 1: Parser

We are currently working on a parser that will take a sequence file and convert it into a sequence type. The sig will be similar to the following. As you can see, most of the methods by which files are read will be visible but not specified at a granular level, as we may find that amino acid and nucleotide sequence files are quite different.

```
module type PARSER =

exception input_closed
exception output_closed
exception no_more_input

(*abstract input type.*)
type input

(*actual DNA or Amino acid sequence.*)
type sequence

(*meta information, such as chromosome number, species, etc.*)
type meta

(*includes the DNA sequence as well as meta information.*)
type container

(*abstract output type.*)
type 'a output

(*read in a file character by character.*)
val read: input -> char

(*read in a file and add its contents to the container.*)
val read_to_container: input -> container

(*read in the meta portion of a file into the meta part of the type.*)
val read_to_meta: input -> meta

(*read in the sequence portion of a file into the sequence part of the type.*)
val read_to_seq: input -> seq

(*open the input.*)
val open: input -> unit

(*close the input.*)
val close_in: input -> unit

(*add a given container to a list.*)
val add: container -> list

end

(*Some ideas were pulled from http://ocaml-lib.sourceforge.net/doc/IO.html *)
```

Parser Implementation

For DNA and RNA (nucleotide) Sequences:

```
module nucleotide_parser:PARSER
struct
  (*Implementation of the above functions specific to nucleotide sequences, with debug
  functions as well.*)
end
```

For Amino Acid Sequences (if time permits):

```
(*optional - and this is a working title.*)
module polypeptide_parser:PARSER
struct
  (*Implementation of the sig functions specific to amino acid sequences, with debug
  functions as well.*)
end
```

The following type will be used for both DNA and protein. Meta may be expanded upon as we learn more about how to develop the sequence files, but we will be starting with this basic structure.

```
type container = { sequence : string; meta : string ; }

(*These are the beginnings of some helper functions.*)

(* open a channel to a file*)
let input_ch = open_in "filename"

(* read file line to carriage return*)
let lines = input_line "filename"

(* parse the line and store into list of sequences *)
match lines with
| seq -> seq :: sequence_list
```

Helper Functions: The following helper functions will be hidden.

```
print_out_sequence
(* prints out sequence_list to make sure that sequences in the file
  were read into the list correctly *)

debug_in_Channel
(* function to print out the lines as they are read in - for debug only *)

input_from_stdio
(* function to take a sequence from stdio instead of a file *)
```

Module Type 2: Sequence comparison

Internal Data Structure: Sequences will be stored in a linear set that a function will traverse.

Data Structure Module

This module type has two purposes:

- 1) To compare two query sequences that are entered, and
- 2) To compare one entered query sequence with a list of query sequences.

The module type for this will be similar to the following:

Once a query sequence is entered

```
module type SEQ_COMPARE
sig
```

```
type sequence
```

```
type container
```

```
type seq_set
```

```
(*iterate through a list of sequences and return a ranked sequence.*)
```

```
val sequence_map: sequence -> seq_set -> seq_set
```

(*compare two sequences. Return a score that reflects their alignment and how they look aligned. For instance, comparing CTGA and CTGTA may result in the aligned sequences:

```
CTG-A
CTGTA *)
```

These two aligned sequences would be present in the tuple of strings.

The higher the score, the greater probability that these sequences are evolutionarily related.

```
val seq_compare: sequence -> sequence -> (score * (string * string))
end
```

```
(*
 * IMPLEMENTATION
 *)
```

```
module polypeptide_compare : SEQ_COMPARE
struct
(*implementation of the above functions.*)
```

(*If time permits, we may add a function (not mentioned in the sig*) that decides what algorithm to use based on the comparative sequence lengths.*)

(*Pseudo-code: If length of sequence A - length of sequence B is within a specified range, use the Needleman-Wunsch algorithm. Else, use the Smith Waterman algorithm.*)

```
end
```

```
module type base_seq_compare : SEQ_COMPARE
struct
```

```
(*same as above, but with slight differences to accommodate base sequences.*)
end
```

Module Type 3: Comparison Algorithms

Needleman-Wunsch for sequences of similar length; potentially Smith-Waterman for sequences of varying length. I do have a working algorithm, for Needleman-Wunsch. The sig and struct are below.

```
module type SEQ_ALG =
sig
  (* this matrix tells you the score obtained from comparing two given chars. *)
  val base_matrix : int array array

  (* this will give you the index of a given char in the base matrix. *)
  val index_of : char -> int

  (* based on two x-y coordinates, this will return the node in the base matrix. *)
  val getNode : int -> int -> int array array -> int

  (* the main function that aligns two sequences using the NW algorithm. *)
  val align_sequences : string -> string -> (int * (string * string) * (int * int) array
array)

  (*initializes a score matrix. Each node in the matrix starts with a tuple of (0,0).*)
  (*the tuple corresponds to (value,path) wherein value is the score and path tells you the
  neighboring box (upleft = 0, up = 1, or left = -1) from which the score was derived*)
  val init_matrix : string -> string -> (int * int) array array

  (*gives you the maximum of three scores;
  returns 1) the highest score and 2) the box from which it was derived. *)
  val max_trace : int -> int -> int -> (int * int)

  (*adds a (score, path) tuple to a node in the matrix. *)
  val set_score_path : (int*int) -> int -> int -> (int*int) array array -> unit

  (*gets the score for a match between two chars using the scoring matrix. *)
  val getScore : char -> char -> int array array -> int
end
```

Actual Code for Needleman-Wunsch Implementation

This version contains a sig that returns the actual scored matrix for debugging purposes – the final one probably will not.

```
module Needleman_Wunsch_DNA : SEQ_ALG =

struct

(*scoring matrix*)

(*
      a   c   g   t   *)
let a: int array = [| 2; -1; -1; -1 |];;
let c: int array = [| -1; 2; -1; -1 |];;
let g: int array = [| -1; -1; 2; -1 |];;
let t: int array = [| -1; -1; -1; 2 |];;

(* Condensed version:
  A C G T
A 2 -1 -1 -1
C -1 2 -1 -1
G -1 -1 2 -1
T -1 -1 -1 2
*)

let base_matrix = [|a;c;g;t|];;

(*get index of a character in our 1D array*)
let index_of (mychar:char) : int =
  match mychar with
  | 'a' -> 0
  | 'c' -> 1
  | 'g' -> 2
  | 't' -> 3
  | _ -> -1;;

let sample_string = "atcg";;

(*get value in a matrix based on x and y coordinates*)
let getNode (x:int) (y:int) (arr: int array array) : int =
  let row = Array.get arr y in
  Array.get row x;;

(*get value from a tuple in a matrix based on x and y coordinates*)
let getValue (x:int) (y:int) (arr: (int*int) array array) : int =
  let row = Array.get arr y in
  let (value, _) = Array.get row x in value;;

(*get path from a tuple in a matrix based on x and y coordinates*)
let getPath (x:int) (y:int) (arr: (int*int) array array) : int =
  let row = Array.get arr y in
  let (_, path) = Array.get row x in path;;

(*get tuple from a matrix based on x and y coordinates*)
let getTuple (x:int) (y:int) (arr: (int*int) array array) : (int*int) =
  let row = Array.get arr y in
  Array.get row x;;
```

```

(*get score for a match between two chars from scoring matrix*)
let getScore (c1:char) (c2:char) (arr: int array array) : int =
  let x = indexof c1 in
  let y = indexof c2 in
  getNode x y arr;;

(*set a node in the sequence matrix*)
let set_score_path (value:int*int)
  (x:int)
  (y:int)
  (arr:(int*int) array array) : unit =

  let row = Array.get arr y in
  let _ = Array.set row x value in
  ();;

(* get max of three ints with traceback path*)

(*0 = up-left, -1 = left, 1 = up*)
let max_trace (upleft:int) (left:int) (up:int) : int * int =
  let a = max upleft up in
  let top = max left a in
  match (top=upleft,top=left,top=up) with
  | (true,_,_) -> (top, 0)
  | (_,true,_) -> (top, -1)
  | (_,_,true) -> (top, 1);;

let () = assert((max_of 2 3 4) = 4);;

let init_matrix (s1:string) (s2:string) : (int*int) array array =
  let h = String.length s1 in
  let w = String.length s2 in
  Array.make_matrix w h (0,0);;

(*PRIMARY FUNCTION FOR ALIGNING TWO SEQUENCES*)

let align_sequences (s1:string) (s2:string) : (int * (string * string) * 'a array
array) =

  (*gap penalty*)
  let pen = -8 in

  (*highest indices or horiz,vert respectively*)
  let max1 = (String.length s1) - 1 in
  let max2 = (String.length s2) - 1 in

  (*initialize a zero matrix*)
  let mat = init_matrix s1 s2 in

  (*recursive helper function for getting scores*)
  let rec calc_scores (cx: int) (cy:int) : (int*int) =

    (*get a score based on the current chars*)
    let c1 = String.get s1 cx in
    let c2 = String.get s2 cy in

    (*score from base matrix based on current char matching*)
    let base_score = getScore c1 c2 base_matrix in

    (*add some conditions where string is significantly shorter*)

    (*determine score for upleft*)
    let upleft =
      match (cx,cy) with

```

```

    | (0,0) -> 0
    | (0,_) -> pen*cy (*ok to use cy, because it is top left*)
    | (_,0) -> pen*cx
    | (_,_) -> getValue (cx-1) (cy-1) mat in

(*if at top row, penalty * pos in row; else, get score up top*)
let up = if cy = 0 then pen*(cx+1)
        else getValue cx (cy-1) mat in

(*if in left column, penalty * pos in col; else, get score on left*)
let left = if cx = 0 then pen*(cy+1)
          else getValue (cx-1) cy mat in

(*get maximum of the three options*)
let score = max_trace (base_score + upleft) (left + pen) (up + pen) in

(*input the score into the matrix*)
set_score_path score cx cy mat;

(*if x-count less than row length, go to next char in row... *)
if cx < max1 then calc_scores (cx+1) cy else

(*else go to next row.
   if y-count less than # rows, start at next row; else return value*)
if cy < max2 then calc_scores 0 (cy+1) else score in

let (final_score, path) = calc_scores 0 0 in

(* traceback *)

let rec t_helper (align1:string) (align2:string) (rem1:int) (rem2:int) : (string *
string) =

    let char1 = if rem1 > 0 then String.make 1 (String.get s1 rem1) else "%" in
    let char2 = if rem2 > 0 then String.make 1 (String.get s2 rem2) else "$" in

    match (rem1 < 0, rem2 < 0) with
    | (true, true) -> (align1, align2)

    (*correct?*)
    | (false, true) -> t_helper (char1 ^ align1) ("-" ^ align2) (rem1-1) (rem2)
    | (true, false) -> t_helper ("-" ^ align1) (char2 ^ align2) (rem1) (rem2-1)

    (*if both have chars left*)
    | (false, false) ->

        let path = getPath rem1 rem2 mat in

        (match path with
        | 0 -> t_helper (char1 ^ align1) (char2 ^ align2) (rem1-1) (rem2-1) (*get node
from map*)
        | 1 -> t_helper ("-" ^ align1) (char2 ^ align2) (rem1) (rem2-1)
        | _ -> t_helper (char1 ^ align1) ("-" ^ align2) (rem1-1) (rem2)) in

        (final_score, (t_helper "" "" max1 max2), mat)

end

```


Module Type 4: Display Output

We will adapt and write OCaml code to generate search results in the form of a ranked list of sequence names. The names and information displayed will be from the “meta” part of the sequences, though if time permits, we aim to have a show/ hide or URL that takes you to the actual sequence.

```
module type SEQUENCE_DISPLAY
sig
  val std_response_header
  val biostat_home_page
  val query_response_header
  val query_response_footer

  (*Convert a set of sequence names to HTML to splice into ranked sequence list to send
  our clients.
  val html_of_seq_list: container list -> string
  end

module type seq_display: SEQUENCE_DISPLAY
struct
  (* html_of_seq_list implemented here.*/)

  (*The following sample functions are essentially from moogle.ml and will be adapted.*/)

  let std_response_header =
    "HTTP/1.1 200 OK\r\n" ^
    "Server: biostat/0.0\r\n" ^
    "content-type: text/html; charset=utf-8\r\n" ^
    "Content-Language: en-us\r\n" ^
    "Connection: close\r\n\r\n"
  ;;

  (*string for moogle_home_page*)
  let biostat_home_page = "./biostat.html" ;;
  (* read in all the lines from a file and concatenate them into
  * a big string. *)
  let rec input_lines inchan lines =
    try
      input_lines inchan ((input_line inchan)::lines)
    with End_of_file -> List.rev lines
  ;;

  (* The header for search responses to clients. *)
  let query_response_header =
    std_response_header ^
    "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\">" ^
    "<html> <head> <title>Biostat Search Results</title></head>" ^
    "<body><h1>Biostat Search Results:</h1><p><ul>"

    (* The footer for search responses to clients. *)
    let query_response_footer = "</ul><hr></body></html>"
  ;;

  (*additional functions to be included.*/)
end
```

Progress Report

-For now we are seeing what we can accomplish without Biocaml. We are going to create our own sequence files and the parser will be written without Biocaml. If time permits, we will see if it would be useful to integrate it for the purpose of integrating more file types.


-The first iteration of the Needleman-Wunsch Algorithm is complete.

-We are planning to start the web GUI this week and hopefully wire it up to the Needleman-Wunsch algorithm module for performing a comparison between two sequences.

Feature List

Core Features

Searching with one entered sequence: Using the Needleman-Wunsch algorithm (or something similar) the program will take an entered sequence and run through each DNA/RNA sequence in a set of known sequences, each of which occur in their own files. Each of these files is indexed at runtime and stored in a data structure. The program will compare the entered sequence to each known sequence.

What is displayed: The program will only display sequences within a certain threshold. If the threshold is set to sequences that are 80% similar  it will only display those known sequences that are 80% or more similar. For each sequence that meets the required threshold, the program will return the following:

- 1) Meta information - potentially the following:
 - a) name of species to which the sequence belongs
 - b) Chromosome number (if applicable)
 - c) Location on chromosome or larger sequence (if applicable)
- 2) sequence name (if it exists)
- 3) a value indicating the degree of similarity to other sequence

Comparing two sequences: Using the same algorithm as above, the program will take two entered sequences and return a value indicating degree of similarity.

Cool Extensions

Visual Representations of Output

We may Write OCaml code to generate visualizations of bioinformatics data - potentially a very crude version of the [UCSC Genome Browser](#). This will allow the end-user to see potential matches more readily.

Additional Inputs: To control the output, the user may determine the similarity threshold. In addition to returning sequences with a particular threshold, the program might also return a particular number of sequences based on a number the user enters.

Links: For each resulting sequence that gets returned, we may try to display a link to the actual FASTA file or DNA/RNA sequence.

Upon clicking on a given sequence, the program may display two sequences showing regions of similarity or dissimilarity.

The program might display the output sequences in a more visually informative way, such as a phylogenetic tree for microbes wherein the nodes are species names. An excerpt below describes the significance of these representations and their pertinence to evolutionary systematics.

In the laboratories of Woese and others, it was shown that phylogenetic relationships of bacteria, and, indeed, all life-forms, could be determined by comparing a stable part of the genetic code... The part of the DNA now most commonly used for taxonomic purposes for bacteria is the 16S rRNA gene. From <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC523561/>

Identification

The program may also

- Identify the start and stop codons in a gene sequence and return the location(s) of these codons.
- return the number of codons in a DNA sequence.
- return the number of amino acids in an amino acid sequence

Conversion

The program may also

- convert a DNA sequence to its RNA counterpart.
- convert a DNA sequence to its complementary DNA sequence
- convert a DNA or RNA sequence to its corresponding amino-acid sequence
- return the number of bases in a DNA sequence.

Additional Comparison Possibility

Our program might also compare amino acid sequences in proteins, in addition to DNA and RNA sequences.

Technical Specification

Modules Included:

Input_Parser- Ocaml Code that opens a file, potentially at runtime, and converts a set of sequences into our internal data structure for searching algorithms to work on (Sample http://biocaml.org/doc/dev/api/Biocaml_fasta.html)

Helper modules may include a functor that can take sequence and map them onto different data structures, such as lists and red-black trees.

Internal_data - Ocaml code that stores baseline sequences for Comparison - Sample (http://biocaml.org/doc/dev/api/Biocaml_table.html)

The key features in this file would be the structure of the red-black data tree itself. This would probably include a data type that includes how a leaves, branches, and nodes would be represented in code.

Algorithm - Ocaml code that perform sequencing comparisons - Sample (http://biocaml.org/doc/dev/api/Biocaml_pwm.html)

Other resources include:


Sequence Alignment Algorithm
http://www.cmb.usc.edu/papers/msw_papers/msw-103.pdf

Needleman-Wunsch Algorithm
http://en.wikipedia.org/wiki/Needleman-Wunsch_algorithm

This would include an OCaml implementation of a sequence-comparison algorithm such as Needleman-Wunsch. (insert links to algorithm here.)

Graphics Display - Ocaml code that implements a GUI for output analysis - Sample (http://genome.ucsc.edu/cgi-bin/hgTracks?db=hg19&position=chr21%3A33031597-33041570&hgid=369943387_shR6NVp6AxaesiHHI844kfF0ftao)

Graphical Elements would include the following:

- Input fields 
- Search Button
- Radio buttons (for selecting threshold, number of sequences to return, sequence type, etc.)
- Output (either as text or if time permits text with graphics, e.g. phylogenetic trees)

Each of these would have their own styling and may trigger events that the Graphics Display responds to.

We might implement the Graphics and Web piece with the help of the [js_of_ocaml](#) library, which converts OCaml to pure javascript code so that the UI will work in a web browser.

Data Type: Each DNA sequence will be stored in a data type that will contain information such as location in the chromosome, chromosome number, and name of the species. Each sequence will be in its own file - for instance, a FASTA file that contains meta-information (chromosome number, etc.)

Data Structure: Initially, we plan to populate and store our sequence data types in a list since we are still working out how to compare sequences. I do not yet know of an exact way of comparing sequences the way you compare numbers (using $<$ or $>$), strings (ABC order), or webpages (the presence of keywords). So we are still looking into the ideal data structure for storing our sequence data types.

Questions we will investigate: Is there an advantage of using a tree as opposed to just a list if we are going to iterate through each sequence anyway? Is there a way to sort sequences in such a way so that branches or clades with largely dissimilar sequences do not have to be traversed? Maybe if the tree is ordered phylogenetically, wherein the root node is a common ancestor and each level of the tree corresponds to a subsequent generation. We will have to look into this.

Revised Project Schedule

Objectives for Week 2 (April 21-27)

Finish the parser, which should be able to read a file and then store its information in a type.
Design the type for this parser, which will consist of the actual sequence and meta information.

Develop Algorithm Modules

- Revise Needleman-Wunsch Module
- Add Function for comparing proteins

Add a type that incorporates a DNA sequence and meta information.

Add function for comparing one DNA sequence with each member of a list of sequences, and then ranking the sequences in order of similarity.

If time permits:

- Add Smith-Waterman Algorithm for comparing sequences of significantly different lengths.
- Add functions for converting between DNA and RNA sequences.
- Add functions for converting between RNA and amino acid sequences.
- Add an algorithm that look for repeats of a sequence in a given sequence.

Super ambitious goal: Add an algorithm that can construct a phylogenetic tree based on multiple sequence comparisons.

Start the GUI using the Moogles code as a starting point. By the end of the week, I'd like the GUI to compare two sequences that are entered.

Objectives for Week 3 (April 28-May 2)

Build up an internal repository of sequence files.

Connect the parser, the list search functions, the algorithms, and the GUI.

- Have the program build a list of DNA sequences at runtime.
- Make search function work when a sequence is entered into the GUI.
- Determine % similarity
- Return a ranked list of sequences.
- Add the option wherein two entered sequences can be compared via Needleman-Wunsch.

If time permits, add the following cool extensions:

- A visual representation of sequence similarity using OCaml-to-Javascript library.
- additional options to the GUI that allow for conversion of DNA to its complementary sequence, DNA to RNA, and RNA to proteins.
- sequence comparison using both Needleman-Wunsch and Smith-Waterman algorithms.
- Option for searching amino acid sequences.

Repository

We will be using Nevin's Git repository at code.seas.harvard.edu for collaborating.

RESOURCES

Sequence Alignment Algorithm

http://www.cmb.usc.edu/papers/msw_papers/msw-103.pdf

Needleman-Wunsch Algorithm

http://en.wikipedia.org/wiki/Needleman-Wunsch_algorithm

RDP

<http://rdp.cme.msu.edu/>

BLAST Lesson

<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3867762/>

Algorithms for Molecular Biology

<http://www.almob.org/>

16S Ribosomal RNA Algorithms

NAST Algorithm

<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1538769/>

<http://www.ncbi.nlm.nih.gov/pubmed/16845035>

<http://greengenes.secondgenome.com/downloads>

Chimeric Sequences

<http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3044863/>

List of sequence alignment software

http://en.wikipedia.org/wiki/List_of_sequence_alignment_software

<http://www.biostars.org/p/14359/>

Intro bioinfo projects

<http://rosalind.info> (in Python)

BIGCAT

http://www.biqcat.unimaas.nl/wiki/index.php/BigCaT_people

Needleman-Wunsch

<http://stackoverflow.com/questions/21199263/how-to-handle-multiple-optimal-edit-paths-implementing-needleman-wunsch-algorit>

Memorandum

<http://www.ueda.info.waseda.ac.jp/~sakai/ocaml/ocamlmemo.html>