Suffix Arrays for DNA Pattern Matching

Technical Specification

https://github.com/didjeridou/sequences

Guillaume Laliberté

CSCI 51

Prof. Greg Morrisett

Advisor TF Thomas Jiang

April 2015

Technical Specification

This is the technical specification for *Sequences*, a suffix arrays implementation for DNA pattern matching. The project repository is https://github.com/didjeridou/sequences

1. Overview and Goals

While DNA sequence used to be costly to achieve and required significant resources, it is now becoming more accessible everyday. A reason for this is that the quantity of data available is growing exponentially, and DNA databases make available that data to researchers. One challenge remains: processing DNA and comparing the (very long) strings of A,T,C and G can be computationally demanding.

One way to improve the way DNA sequences are compared is to use data structures adapted to this task. The suffix arrays of suffix trees are a very good candidate when is comes down to comparing strings.

With this project, **my goal is to create an Ocaml program that implements said data structure and allows the user to check if a particular sequence can be found in a DNA sequence**. This is very useful to determine if a DNA sequence contains a gene in particular.

2. Feature List

2.1 Core

To answer a question like "does this DNA sequence contain the gene that often cause diabetes?", the following core features will be implemented first:

1. Implementation of the suffix array (the absolute minimal feature)
2. A terminal command to load DNA files and/or string to compare
3. A comparator that uses a suffix array to compare the sequences
4. A simple output of the results
    - i) Is the sequence there (bool)
    - ii) If so, what is its position (int option * int option)
    - iii) Possibility to invert the sequence (because DNA is an helix of two complementary sequences)

2.2 Extra features

With the time left, I hope to add these additional features:

5. Additional metrics for DNA comparison (ATCG composition and % similarity)
   i) ATCG composition (tuple of four integers)
   ii) Similarity percentage (float)
   iii) Most frequent common sequences (string)
6. Implementing the compressed suffix array (CSA) which saves a lot of space
7. Graphical way to represent the DNA similarities
8. A queue to automate comparisons (multiple files)

3. Components and Signatures/Interfaces

The core of the program will be a `DNASequence` module implemented with a suffix array. The `DNASequence` is of type `sequence` of elements of type `base`. The goal is to abstract the array so that the user can manipulate sequences that behaves like a DNA sequence should.

The project structure will be:

```
/Sequences
     DNASequence.ml        DNASequence module (SA & CSA implem.)
     sequences.ml          Main program
     SuffixArray.ml        SuffixArray module
     /data                 Folder containing dna data to test
          test.dna         Example of test DNA files
          cow.dna
          human.dna
          disease.dna
     /_documents           Folder for all other non-code files
          document files
```

The input will allow for direct string inputs in the terminal, or for files containing the DNA data. Commands will look like:

```
(* Search for a pattern  *)
./sequences.native -search "string" -file data/human.dna


(* Inverse the DNA (A<->T, C<->G) and search  *)
./sequences.native -search -inv "string" -file data/alien.dna


(* Display the ATCG composition  *)
./sequences.native -composition -file data/cow.dna


(* CSA option would run the CSA implementation (extra)  *)
./sequences.native -composition -csa -file data/unknown.dna
```

As an extra feature, the following command could open a DNA file and allow the user to run multiple operations. After all, the goal is to create a suffix array once to be able to quickly execute multiple operations quicker:

```
(* Index moo.dna as DNAsequence (suffix implementation) *)
./sequences.native data/moo.dna
```

SuffixArray

The `DNASequence` will be implemented as `SuffixArray (SuffixArray.ml)`:

```
is_empty : array -> bool
create : string -> array
search : string -> array -> (bool * int option * int option)
invert : array -> array
BWTransform : (* optional, allows for compression *)
```

DNASequence

The `DNASequence (DNASequence.ml)` module signature will be slightly different. It will implement abstract types `sequence` and `base`.

```
is_empty : sequence -> bool
create : string -> sequence
```

```
search : sequence -> sequence -> (bool * int option * int option)

invert : sequence -> sequence

string_of_sequence : sequence -> string

base_count : base -> int
```

Main program

The main program will run from `./sequences.native`. It will take as input parameters from the terminal and execute the relevant operations, like indexing a DNA file into a DNASequence (suffix array) and run the operations specified via parameters.

4. Progress Report

Since the last report, a few aspects of the project progressed:

- Chose the final data structure (core: suffix array, extra: compressed suffix array)
- The first signature proposition was erroneous. With more information from a more complete research, the signatures for `SuffixArray` and `DNASequence` were updated.
- Created the repository and project structure (https://github.com/didjeridou/sequences)
- Refined the metrics to analyze. The core feature is now the search for patterns and I hope to be able to complete most or all of the extras metrics-related features (as listed above)
- Started to code the interfaces

5. Next Steps

Here are the next big steps for the project:

- Implement and test SuffixArray module and functions
- Implement and test the DNASequence module and functions
- Test the functionality by finding if a sequence is or is not in another sequence
- Implement the command line utility
- Test the program with the command line arguments
- Add the extra features in the following order
    1) Sequence transformation (A<->T, C<->G)
    2) ATCG composition
    3) CSA implementation of the DNA sequence and performance comparison

4) Percentage of similarity between two sequences or longest common sequence
5) Run the program to parse the DNA and then allow for multiple commands

6. Project Schedule

Date	Goal	Turn in
Apr 13, 2015	Select primary metrics	
Apr 14, 2015	Selection of suffix trees or arrays	
Apr 15, 2015	Proposed modules and functors (mostly completed)	
Apr 16, 2015	Proposed project structure	
Apr 17, 2015	Final Technical Specification	**Technical Specification**
Apr 19, 2015	SuffixArray implementation	
Apr 21, 2015	SuffixArray unit tests, DNASequence implementation	
Apr 23, 2015	Try the program, make sure the core f. works. Check all unit tests, fix bugs and polish code	
Apr 25, 2015	Command-line interface, ATCG metrics implementation	
Apr 27, 2015	Polish code, check unit tests and test the program	**Functionality Checkpoint**
Apr 29, 2015	CSA implementation & comparison or more metrics	
Apr 30, 2015	Review of the whole project. Guidelines, tests, functionality, writeup, and others	
May 1, 2015	Cleaned up code, fully-working project	**Finish Project**