Suffix Array
Maintenance Guide

Tal E. Cohen

Table of contents

1. Overview

In this project our aim was to develop an optimized approach for storing and analyzing data concerning sub-sequences within char sequences of given text files. The Suffix Array we implemented in this project – as opposed to the Suffix Trie data structure we've begun with, based on previous researches – allows handling almost unlimited file sizes and outperforms the Trie approach significantly.
Both approaches were tested on generic text files and specific DNA files alike. The developed software allows classification of genetic diseases by comparing a subject's DNA data to reference groups of people representing various conditions of genetic disease immunity.

2. Acronyms and glossary

	Meaning
Infix	A volatile sub-sequence of a Suffix starting from the beginning up to any position in the Suffix
Prefix	A non-volatile sub-sequence of a Suffix starting from the beginning up to any position in the Suffix
SA	Suffix Array
.saf	SA file extension of each of the SA's splits, saved as compressed references to the input
.sai	SA input extension of the compressed saved input
Split	Sub-tree of a SA, independent of other sub-trees
Suffix	A sub-sequence of an input line starting at any position in the line up the end of the line

3. Versions

Document versions

#	Description	Date
1.2	CS B.Sc. final project submission	June 30th 2017

Software versions

#	Description	Date
1.8	SuffixArrayBuilder – builds and saves splits of Suffix Array	June 25th 2017
1.8	SuffixArrayAnalyzer – generates Infixes out of splits and ranks them	June 25th 2017

4. Introduction

The system is divided into two sub-systems: SuffixArray and classification (which is documented in a separate guide).
SuffixArray is composed of two parts as follows:

1. SuffixArrayBuilder

   Decomposes the input data into splits (using a set of unique prefixes) and builds a SA data structure for each split. Both the input data and the splits are saved in a compressed form for further analysis.

2. SuffixArrayAnalyzer

   Loads the saved input and retrieves the splits one by one while accumulating statistic data and rating each Infix, to finally output the requested data (a "top" list and "training" files).

The input is raw text in which all characters are significant (an additional utility is available for extracting the relevant data from standard DNA files). The capacity of the current software is up to 1.5 million lines of up to 1,000 characters.

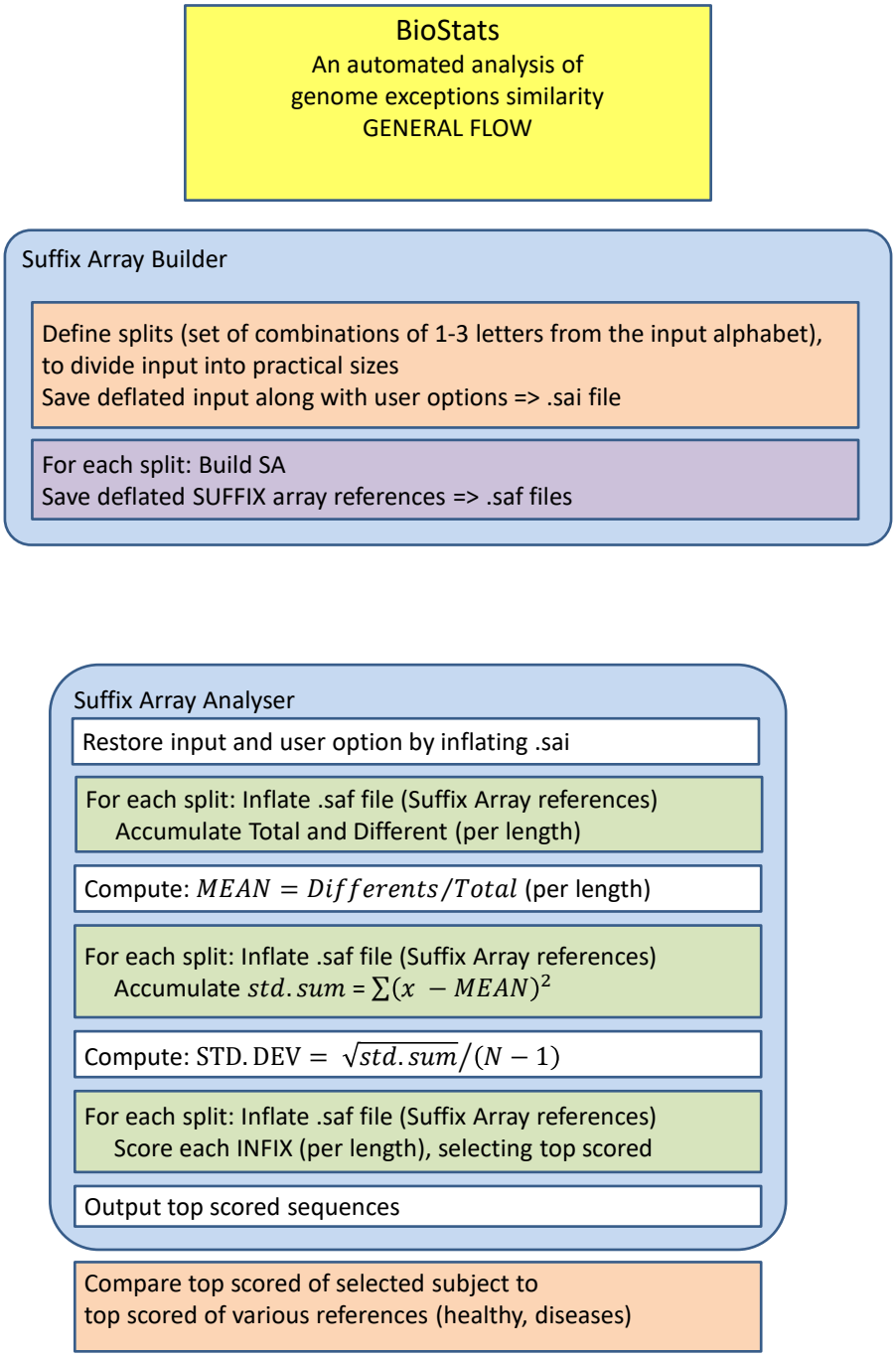
The data is processed by the SuffixArrayBuilder that saves:

1. The input data as a .sai file (compressed raw input).
2. The SA references (a set of a reference to the input line and Suffix count) as .saf files (also compressed).

The output of the SuffixArrayAnalyser includes:

1. A List of the top x Infixes, using user-selected criteria (Z-score, count, etc.).
2. Optional lists (of user-selected length) of:
   a. Infixes with a count over a user-defined threshold.
   b. Infixes with a Z-score over a user-defined threshold.

5. Program Structure



1. Overview

   The program uses a library of generic functions (TecCommon.jar) including:
   - Command line parser
   - Large files reader (reading by chunks)
   - Progress and debug printing
   - Time measurements and formatting
   - Generic string converting and formatting
   - Filename and folder decomposition and formatting
   - Platform dependent functionalities

   The program uses a library of SuffixArray functions (SuffixArray.jar) including:
   - Prefix and Suffix classes
   - SA file inflating and deflating
   - .sai/.saf wrappers
   - Split set generator (based on input statistics)
   - Single split SuffixArray generator

2. The Suffix Array

   I. Highlights

   The main principle of the SA is to collect all the unique suffices within the data into a sorted list. Each suffix is either added (if it does not exist already) or accounted for as the building process goes. After that it is possible to run over the sorted list of suffices referring to each infix (prefix of a suffix) and accumulate global statistics to finally calculate the Z-score of each one.

   II. Implementation

   Runtime optimization

   The SA is built using a hash-table of prefixes whose items point each to a sorted table containing suffices that share that same prefix. This double-layered implementation is done in order to speed up the insertion process of suffices by using small tables instead of a single sorted list.

   Memory usage optimization

   In addition, the data is split into disjoint suffix groups (referred to as "splits" in the code) to allow working on a smaller section of data at a

time. The program takes advantage of the split method in both the building and analyzing stages.

Process steps

Builder:

- Parse, analyze and save the command-line parameters
- Read and store the input file(s) and determine the working set of characters (the "alphabet"). The input data is also saved as a .sai file to be used by the SuffixArrayAnalyzer later on
- Gather statistics in order to divide the stored input into splits, where each split is a subset of the alphabet character combinations (one to three characters in length)
- For each split the program scans the stored data and processes the relevant suffices, inserting them into their rightful place inside the Suffix Array structure (hash-table of sorted tables). The actual data being stored is a reference to the input line and a counter. The sorted list of these references is saved as a .saf file to be used by the SuffixArrayAnalyzer later on

Analyzer:

- Parse, analyze and save the command-line parameters
- Load the .sai file created by the SuffixArrayBuilder
- Identify the splits (from the SA input folder)
- Suffices are loaded from each split in turn in order to extract their infixes allowing to calculate:
  - the average of each infix length
  - the standard deviation of each infix length
  - the Z-score for each infix individually. The best scores are saved in a list (of user-selected size) and finally outputted

3. Classes

I. SuffixArrayBuilder

The main class of this project is the Control class that implements:

- User-selection retrieval using CTec_CmdLine
- Read the input lines using CTec_FileReader and store them into a .sai file using SuffixArray.
- Obtain the split division using SuffixArray and save the input using SAFileInput

- Builds a partial Suffix Array for each specific split and organizes the structure in order to save it as a .saf file
- Optional output of statistical data of the run (in the form of a log file)

II. SuffixArrayAnalyzer

The main class of this project is the Control class that implements:

- User-selection retrieval using CTec_CmdLine
- Load the input lines stored in the .sai file by using SAFileInput
- Compute the average by loading each split (.saf file) in turn and accumulating the data for each length
- Compute the standard deviation by loading each split (.saf file) in turn, using the average calculated in the previous stage and accumulating the data for each length
- Compute the Z-score by loading each split (.saf file) in turn, using the standard deviation calculated in the previous stage and storing the top results
- Printing the top results

III. Suffix Array

SASuffix

This class holds a reference (line number, position in line and length) and a counter of a suffix instance in the input.

SAPrefix

This class extends SASuffix by adding a list of suffices that begin with the same prefix.

SuffixArray

This class is the main focus of the SuffixArray package. It serves both the SuffixArrayBuilder and the SuffixArrayAnalyzer programs by providing them with the necessary functionalities of the Suffix Array structure such as:

- Storing the input lines
- Analyzing the input in order to divide it into splits
- Inserting new suffices into the structure
- Generating .sai and .saf files
- Loading back .sai and saf files
- Computing statistics and scoring the infixes

- Storing the top results to be outputted

The class also contains the inner-class Infix (see below).

SAFileCommon

This class is intended specifically for supporting proprietary SuffixArray file formats (.sai and .saf) with:

- Write and read file headers
- Inflate and deflate data blocks

SAFileInput

This class derives from SAFileCommon and has the following added functionalities:

- Write and read alphabet and max-infix-length to be used by the Analyzer
- Write and read the input lines by blocks

SAFileSplit

This class derives from SAFileCommon and has the following added functionality:

- Stores and loads Suffix instances of a specific split

IV. Utilities

CTec_Utils

This class contains general functions like:

- Open/write/close log files
- Data and time formatting
- Filename handling
- General string formatting
- OS specific interface

CTec_FileReader

This class reads large text files by chunks into a buffer and parses the buffer to return each line in turn.

CTec_CmdLine

This class parses the command-line, extracting free parameters and switch options. These parameters are validated against a program-defined set.

6. Data files

1. Input
2. .sai

   The SA saved input file (.sai) is composed of:
   - A general header identifying the file type, the version of the program it was created by, the record size and the number of records
   - A specific header containing the alphabet and the user parameters
   - Blocks of compressed input lines
3. .saf

   The SA saved split file (.saf) is composed of:
   - A general header identifying the file type, the version of the program it was created by, the record size and the number of records (dummy)
   - Blocks of compressed SA references

4. Output

7. Non-trivial implementations

The efficiency of the SuffixArrayBuilder and SuffixArrayAnalyzer programs is achieved by 3 main methods that are derived from an algorithmic analysis of the properties of the Suffix Array structure.

I. *The Split division (Builder + Analyzer)*

The incentive of the split division is to allow the processing of larger files in spite of memory limitations.

In the technical literature a Suffix Array is defined as a sorted list of suffices. As a result, groups of suffices starting by the same prefix (in this case a sequence of 1-3 characters) are contiguous and therefor can be treated as independent sub-lists of the original list. We take advantage of this property by dividing the input into sorted sets of prefixes; each set is referred to as a "split" and can be processed separately.

The division takes into account the number of occurrences of each staring prefix so that the splits are close is size.

II. *The Suffix Array structure implementation (Builder)*

The Suffix Array implementation uses a hash-table of prefixes each pointing to a sorted list of suffices.
The incentive of this double-layered implementation is to reduce the runtime, most of it spent on insertions. The insertion time, being roughly proportional to the table size, is greatly reduced by the use of smaller tables.

III. *The Infix scanning (Analyzer)*
The scan is done by going through the Suffix Array (the sorted-list of suffices of the current split), one suffix at a time.
The Z-score, by definition, is computed separately for every infix length. We use an array of objects, each dedicated to a specific length. Every object holds the previous infix (string and data) and the accumulated data up to the current suffix. This array allows gathering that data on all infixes lengths in a single pass, through all splits and in an orderly manner.
We take advantage of the fact that any Infix is a prefix of the current suffix in order to compare the previous and current infixes. The comparison is done starting with the shortest length (1 character) up to the max length (user-defined).
The moment the scanning reaches inequality we can dismiss further comparisons for this suffix and take into account the remaining infixes (increasing the number of different strings per length, computing Z-scores).

8. Future extensions

- Add – optimization of the exisiting SuffixArrayBuilder option that allows to add standard input to an exsiting .sai file and then generate a new SA structure
- Merge – a SuffixArrayBuilder option allowing to merge .sai files
- Statistics acceleration – a SuffixArrayAnalyzer runtime enhancement using only statistic sample of the input data to compute average and standard deviation (expected gain: 50% of runtime)
- Parallel processing – a SuffixArrayBuilder runtime enhancement allowing the processing of more than one split at a time
- Handle input files – a SuffixArrayBuilder option allowing direct input of standard DNA files using only the relevant fields for further processing

- GUI – will facilitate the selection process of the input folders by navigating via Windows Explorer or Linux file manager

9. Appendix

Log examples and Screenshots are available in the User Guide as well as in the additional files that come with this guide.