

Suffix Array

User Guide

Tal E. Cohen

Table of contents

1. Overview.....	3
2. Acronyms and glossary.....	3
3. Versions	3
4. Introduction (a.k.a. Suffix Array for dummies).....	4
5. Installation and Preparation	5
6. Operational Instructions.....	5
○ SuffixArrayBuilder.jar	5
○ SuffixArrayAnalyzer.jar.....	6
7. Environments	8
8. Execution examples	8
○ DNAFileGen.jar	8
○ SuffixArrayBuilder.jar	9
○ SuffixArrayAnalyzer.jar.....	10

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 allows handling large files, and is divided into two entities: a builder that takes most of the runtime and an analyzer that offers different types of analysis to be performed on the output of the builder in a relatively short runtime. Both programs were tested on generic text files and specific DNA files alike. The developed software (builder, analyzer and classifier) 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
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

3. Versions

Document versions

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

Software versions

#	Description	Date
1.8	SuffixArrayABuilder – 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 (a.k.a. Suffix Array for dummies)

The complete software package consists of three programs:

- SuffixArrayBuilder

A program that receive as input text files in which each line is a sequence of characters and the analysis is done on sub-sequences of these lines. The output of this program is a representation of the SA structure in the form of a series of binary files (.saf) along with a compressed input file (.sai).

This program is rarely used on the same data, since it builds the data structure of the Suffix Array which only varies according to the MAX_LEN parameter (the maximal length of the infixes that are to be analyzed).

- SuffixArrayAnalyzer

A program that receives as input the work folder of the builder (containing one .sai file and several .saf files) and analyzes each sub-sequence in order to generate a “top” list that meets selected criteria. This program is used on the same data for different combinations of the available parameters (such as: top list and training criteria).

- Classifier

This part is documented in a separate guide.

In the current version the input should be raw text files (see *Installation and Preparation*)

The current software was tested with up to 1.2 million lines of about 600 characters, and the runtime for such data is around 30 minutes on a middle-end computer.

The output of the SuffixArrayAnalyzer includes:

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

5. Installation and Preparation

Preliminary requirements: Java JRE version 1.8 or above.

Link: [Java JRE download page](#)

A/N: Both Suffix Array programs are written in Java as multiplatform self-contained jars, as a result the only prerequisite is the JRE version above-mentioned.

The input text files should be raw text files in which all characters are significant. For standard DNA files a Java utility “DNAFileGen.jar” can be used to extract the relevant field (see *Operational Instructions*).

The capacity of the current version is up to 1.5 million lines of up to 1,000 characters.

The builder part generates a work folder containing the Suffix Array components in a compressed form, which are: the input files (.sai file) and the splits (.saf files).

A/N: Notice that the size of the work folder can reach up 0.5G for large inputs.

6. Operational Instructions

The run is done in two stages: SuffixArrayBuilder.jar followed by SuffixArrayAnalyzer.jar (the final stage is the classification, which is documented in a separate file)

- SuffixArrayBuilder.jar

Command-line template:

```
java -Xmx<memory> -jar ..\SuffixArrayBuilder.jar  
<input folder> <output folder> <switches>
```

[Note: to obtain a printout of the full template and all the switches available with their explanations it is enough to run

```
“java -jar ..\SuffixArrayBuilder.jar -?”]
```

- <memory> [optional] allocates memory for the JRE. It is recommended to allocate at least 2GB (see *Execution examples* and JRE documentation)
- <input folder> is the path to where the input file(s) will be looked for

- <output folder> is the path to where the work folder will be created and the work files will be saved. A unique string (CRC) is created for each set of data files and it appears in the name of the work folder and in the names of the .sai and .saf files that are inside of it.
- <switches>
 - maxlen : [int] determines the max length of the infixes that will be accounted for in the SA structure
 - size : [smlllall] quick selection of an input file by size, selected from existing files in an input folder (supersedes -input)
 - input : [string:file name] select by name a specific file to run on from the input folder (overridden by the switch -size)
 - add : [string:folder path] an option that allows to add standard input to an existing .sai file
 - work : [string:folder name] an option to determine the name of the work folder (inside of the default CRC)
 - debug : [] creates log files that gives information about the processing steps, sizes of the saved files and timings
 - progress : [] for progress feedback in the terminal window

○ SuffixArrayAnalyzer.jar

Command-line template:

```
java -Xmx<memory> -jar
..\SuffixArrayAnalyzer.jar <work folder>
<output folder> <switches>
```

[Note: to obtain a printout of the full template and all the switches available with their explanations it is enough to run

```
“java -jar ..\SuffixArrayAnalyzer.jar -?”]
```

- <memory> [optional] allocates memory for the JRE. It is recommended to allocate at least 2GB (see *Execution examples* and JRE documentation)
- <work folder> is the output folder of a SuffixArrayBuilder run (where all the .sai and .saf files are)
- <output folder> is the path to where the result files will be saved (“top_<CRC>”, “train_count_<CRC>” and “train_score_<CRC>”)
 - <switches>
 - top : [int] define the maximal size of the “top” list that is saved into file “top_<CRC>.txt”

- sort** : [1|2] choose 1 for a “top-score” list (default) or 2 for a “top-count” list
- min.count** : [int] training option that lists all infixes with a count greater than the given threshold into file “training_count_<CRC>.txt”
- min.score** : [int] training option that lists all infixes with a score greater than the given threshold into file “training_score_<CRC>.txt”
- man.lines** : [int] define the maximal number of lines for the training files (default: 10,000)
- norm** : [] normalize the Z-score by input file size
- test** : [string:infix] creates a log file with general debug data and information concerning a specific given infix of interest
- debug** : [] creates log files that gives information about the processing steps, sizes of the saved files and timings
- progress** : [] for progress feedback in the terminal window

7. Environments

Due to the Java implementation SuffixArrayBuilder.jar and SuffixArrayAnalyzer.jar will run under any OS that has a JRE (version 1.8 or above) installed.

The program accepts input files in both Windows and Unix formats. In addition, when building input/output paths the program takes into account the specific format of the file system.

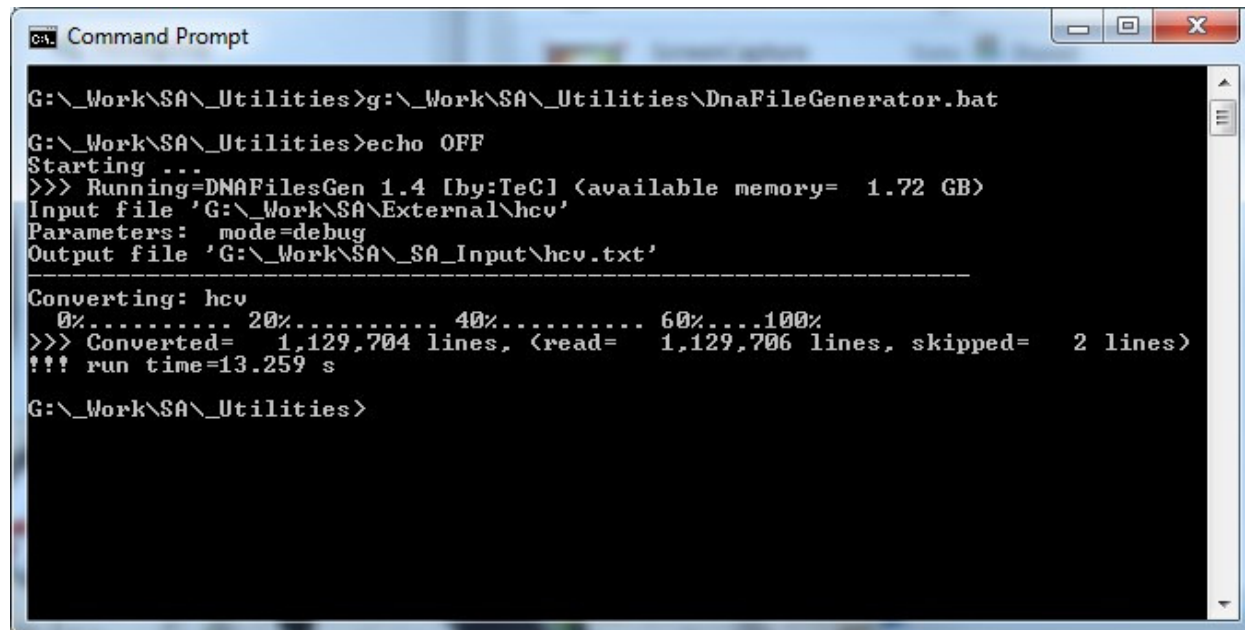
The tests were conducted on Windows 7 and Windows 10, and on Ubuntu flavor of Linux.

The supported input files' encoding is UTF-8.

8. Execution examples

The tested performance of a 0.5G input file ranges between 30m to 60m runtime (depending on hardware).

- DNAFileGen.jar



```
cmd - Command Prompt

G:\_Work\SA\_Utilities>g:\_Work\SA\_Utilities\DnaFileGenerator.bat

G:\_Work\SA\_Utilities>echo OFF
Starting ...
>>> Running=DNAFilesGen 1.4 [hy:TeC] (available memory= 1.72 GB)
Input file 'G:\_Work\SA\External\hcv'
Parameters: mode=debug
Output file 'G:\_Work\SA\_SA_Input\hcv.txt'
-----
Converting: hcv
 0%..... 20%..... 40%..... 60%...100%
>>> Converted= 1,129,704 lines, (read= 1,129,706 lines, skipped= 2 lines)
!!! run time=13.259 s

G:\_Work\SA\_Utilities>
```


○ SuffixArrayBuilder.jar

```

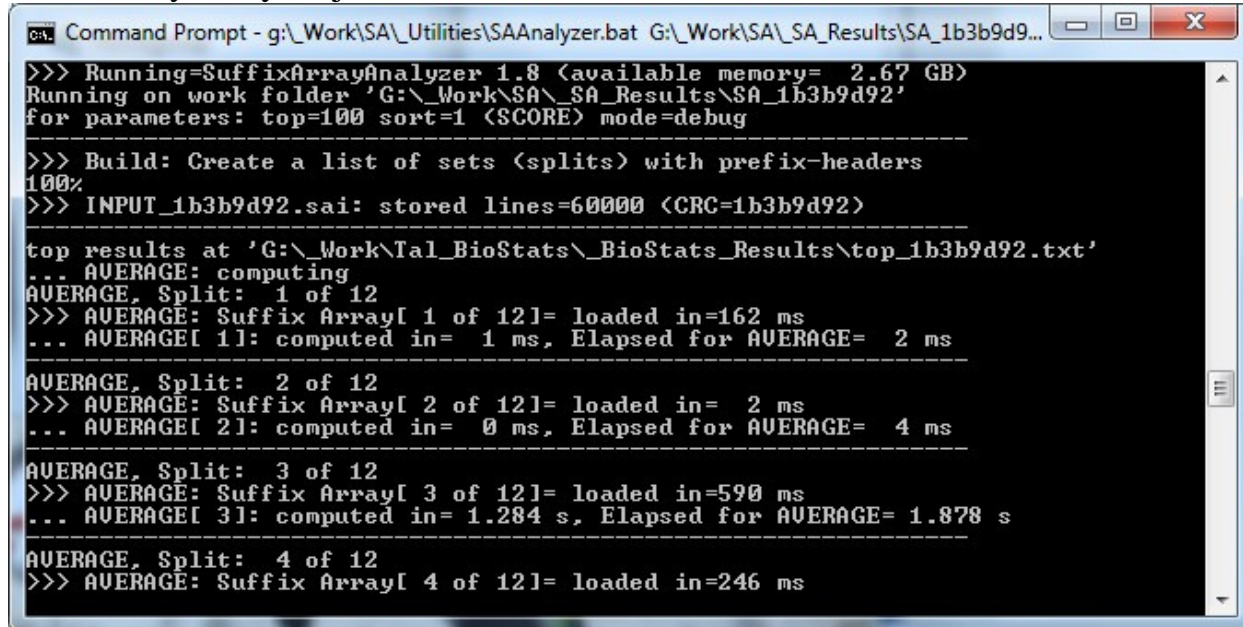
Command Prompt - g:\_Work\SA\Utilities\SABuilder_050.bat m
G:\_Work\SA\Utilities>java -Xmx3G -jar ..\SuffixArrayBuilder.jar G:\_Work\SA\SA_Input G:\_Work\SA\SA_Results -size m -maxlen 50 -debug -progress
>>> Running=SuffixArrayBuilder 1.8 (available memory= 2.67 GB)
Running on input folder 'G:\_Work\SA\SA_Input'
for parameters: maxlen=50 file=M mode=debug

-----
Running file 'G:\_Work\SA\SA_Input\TEST_x600_x60000_B.txt'
>>> TEST_x600_x60000_B.txt: stored lines=60,000
-----
>>> Work folder ready=G:\_Work\SA\SA_Results\SA_1b3b9d92
>>> Total stored lines=60,000 (storage time= 1.528 s)
-----
>>> Build: Create a list of sets (splits) with prefix-headers
0%..... 20%..... 40%..... 60%..... 80%.....100%
>>> Build: Suffix Array saving Input
0%..... 20%..... 40%..... 60%..... 80%.....100%
... Saved [INPUT_1b3b9d92.sai, size= 7.99 MB], saved in= 5.749 s
-----
... Building
0%..... 20%..... 40%..... 60%..... 80%.....100%
>>> Build: Suffix Array[ 1 of 12]= built in= 3.493 s
... Saved [SPLIT_1b3b9d92_001.saf, size=116 B], total saved=116 B
... SA Build [ 1]: computed in= 3.494 s, Elapsed for SA build= 3.497 s
-----
0%....

Command Prompt
0%..... 20%..... 40%..... 60%..... 80%.....100%
>>> Build: Suffix Array[10 of 12]= built in= 4.654 s
... Saved [SPLIT_1b3b9d92_010.saf, size= 6.48 MB], total saved= 53.01 MB
... SA Build [10]: computed in= 5.734 s, Elapsed for SA build=00:01:05
-----
0%..... 20%..... 40%..... 60%..... 80%.....100%
>>> Build: Suffix Array[11 of 12]= built in= 4.208 s
... Saved [SPLIT_1b3b9d92_011.saf, size= 6.28 MB], total saved= 59.29 MB
... SA Build [11]: computed in= 5.311 s, Elapsed for SA build=00:01:11
-----
0%..... 20%..... 40%..... 60%..... 80%.....100%
>>> Build: Suffix Array[12 of 12]= built in= 1.006 s
... Saved [SPLIT_1b3b9d92_012.saf, size= 1.71 MB], total saved= 61.00 MB
... SA Build [12]: computed in= 1.303 s, Elapsed for SA build=00:01:12
-----
>>> BUILD : Done in=00:01:12 (saving=16.432 s, SA=56.029 s) total elapsed=00:01:22
-----
!!! Total run time=00:01:22
FIN
G:\_Work\SA\Utilities>popd
G:\_Work\SA\Utilities>

```

○ SuffixArrayAnalyzer.jar



```
CH. Command Prompt - g:\_Work\SA\_Utilities\SAAnalyzer.bat G:\_Work\SA\_SA_Results\SA_1b3b9d9...
>>> Running=SuffixArrayAnalyzer 1.8 (available memory= 2.67 GB)
Running on work folder 'G:\_Work\SA\_SA_Results\SA_1b3b9d92'
for parameters: top=100 sort=1 (SCORE) mode=debug
-----
>>> Build: Create a list of sets (splits) with prefix-headers
100%
>>> INPUT_1b3b9d92.sai: stored lines=60000 (CRC=1b3b9d92)
-----
top results at 'G:\_Work\Tal_BioStats\_BioStats_Results\top_1b3b9d92.txt'
... AVERAGE: computing
AVERAGE, Split: 1 of 12
>>> AVERAGE: Suffix Array[ 1 of 12]= loaded in=162 ms
... AVERAGE[ 1]: computed in= 1 ms, Elapsed for AVERAGE= 2 ms
-----
AVERAGE, Split: 2 of 12
>>> AVERAGE: Suffix Array[ 2 of 12]= loaded in= 2 ms
... AVERAGE[ 2]: computed in= 0 ms, Elapsed for AVERAGE= 4 ms
-----
AVERAGE, Split: 3 of 12
>>> AVERAGE: Suffix Array[ 3 of 12]= loaded in=590 ms
... AVERAGE[ 3]: computed in= 1.284 s, Elapsed for AVERAGE= 1.878 s
-----
AVERAGE, Split: 4 of 12
>>> AVERAGE: Suffix Array[ 4 of 12]= loaded in=246 ms
```

```
ca. Command Prompt - g:\_Work\SA\_Utilities\SAAnalyzer.bat G:\_Work\SA\_SA_Results\SA_1b3b9d9...
... STD.DEU: computing
STD.DEU, Split: 1 of 12
>>> STD.DEU: Suffix Array[ 1 of 12]= loaded in= 2 ms
... STD.DEU[ 1]: computed in= 0 ms, Elapsed for STD.DEU= 1 ms
-----
STD.DEU, Split: 2 of 12
>>> STD.DEU: Suffix Array[ 2 of 12]= loaded in= 2 ms
... STD.DEU[ 2]: computed in= 0 ms, Elapsed for STD.DEU= 3 ms
-----
STD.DEU, Split: 3 of 12
>>> STD.DEU: Suffix Array[ 3 of 12]= loaded in=196 ms
... STD.DEU[ 3]: computed in= 1.318 s, Elapsed for STD.DEU= 1.517 s
-----
STD.DEU, Split: 4 of 12
>>> STD.DEU: Suffix Array[ 4 of 12]= loaded in=203 ms
... STD.DEU[ 4]: computed in= 1.177 s, Elapsed for STD.DEU= 2.897 s
-----
STD.DEU, Split: 5 of 12
>>> STD.DEU: Suffix Array[ 5 of 12]= loaded in=143 ms
... STD.DEU[ 5]: computed in=876 ms, Elapsed for STD.DEU= 3.916 s
-----
STD.DEU, Split: 6 of 12
>>> STD.DEU: Suffix Array[ 6 of 12]= loaded in=185 ms
```

```
ca. Command Prompt
-----
SCORE, Split: 10 of 12
>>> SCORE: Suffix Array[10 of 12]= loaded in=198 ms
... SCORE[10]: computed in= 1.409 s, Elapsed for SCORE=13.248 s
-----
SCORE, Split: 11 of 12
>>> SCORE: Suffix Array[11 of 12]= loaded in=156 ms
... SCORE[11]: computed in= 1.371 s, Elapsed for SCORE=14.774 s
-----
SCORE, Split: 12 of 12
>>> SCORE: Suffix Array[12 of 12]= loaded in= 48 ms
... SCORE[12]: computed in=377 ms, Elapsed for SCORE=15.199 s
-----
>>> SCORE      : computed in=15.227 s total elapsed=39.532 s
-----
... Top list saved!
=====
!!! Total run time=39.534 s
FIN
G:\_Work\Tal_BioStats\_Utilities>popd
G:\_Work\SA\_Utilities>echo ON
G:\_Work\SA\_Utilities>
```