tssv Documentation

Release latest

Nov 13, 2021

Contents:

TSSV is a program that does targeted characterisation of short structural variation. It can be used for STR analysis, or any other type of targeted analysis. It characterises any variation between a set of user-defined markers.

TSSV is platform-independent. It has been tested on Linux, macOS, and Windows.

Please see ReadTheDocs for the latest documentation.

CHAPTER 1

Introduction

CHAPTER 2

Installation

TSSV depends on `gcc` for the compilation of one of the core libraries. Also a package that provides `Python.h` should be installed. For Debian based systems, the following command will install these packages:

```
apt-get install libpython3-dev gcc
```

On Windows and macOS, you will be prompted to install a C compiler while installing TSSV if you do not have one.

The software is distributed via PyPI, it can be installed with `pip`:

```
pip install tssv
```

If there are multiple versions of Python installed, it might be better to use the command `pip3` instead of `pip`.

2.1 From source

The source is hosted on GitHub, to install the latest development version, use the following commands.

```
git clone https://github.com/jfjlaros/tssv.git
cd tssv
pip install .
```

CHAPTER 3

Usage

The `tssv` program does targeted characterisation of short structural variation. See the help (`-h`) of this program for a full description of the parameters.

3.1 Paired-end read support

With paired-end data in which the insert size was less than twice the read length, the two reads of each pair may overlap partially or entirely. In this setting, it is possible to use a read merging tool to combine the two reads of each pair, possibly giving rise to a longer combined read. The combined read may contain a known allele that does not fit completely in a single read.

Because repetitive sequences like STRs may perfectly align in many ways, such tools will be unable to detect whether the combined read actually contains all repeats that were originally present in the DNA molecule. If the molecule was too long, the two reads may not overlap at all; however, with a highly repetitive sequence, they may still appear to overlap significantly. It is only certain that all repeats have been captured if all of them fitted on a single read - but the flanking sequences may not be present in their entirety for such long alleles. Merging the two reads together may still yield the complete flanking sequence, allowing the allele to be detected by TSSV.

To make sure that any potentially truncated sequences are rejected, the combined reads can be provided to TSSV with upper case letters for the overlapped part (i.e., the middle, where both reads in the pair overlap) and lower case letters for the non-overlapping parts (i.e., those parts only present in one of the two reads). If TSSV aligns a flanking sequence to a completely lower case part of the combined read, this means that the flanking sequence was only present in one of the two reads and had fallen off the end of the other. In this case, it is impossible to tell whether the sequence between the flanking sequences contains all repeats that were originally present in the DNA molecule and therefore it is not meaningful to perform a regular expression match against it.

One particular paired-end read merging tool that supports this kind of output is a fork of FLASH 1.2.11 that was specifically made for this purpose, which is hosted on GitHub. This fork introduces an optional command-line argument `--lowercase-overhang` that, when specified, enables output compatible with the paired-end read support of TSSV.

3.1.1 Cheat sheet: Interpretation of letter case in TSSV input and output

Letter case in the library file:

- All sequences in the library file should be in upper case completely. Regular expressions containing lower case letters will never match. Lower case letters in flanking sequences will always count as a mismatch in alignments.

Interpretation of letter case in input sequences:

- Completely upper case reads are processed normally.
- TSSV will be unable to detect any flanking sequences in lower case sequences.
- In mixed case sequences, the regions to which the flanking sequences are aligned should contain at least one upper case letter. When a flanking sequence aligns to a completely lower case region, the match is rejected. When a pair of flanking sequences is found in the read (and neither match is rejected), the corresponding regular expression is matched case insensitively to the sequence between the flanks.

Letter case in output files:

- Sequences in output FASTA files have the same case as the corresponding input sequences.
- Sequences in output CSV files will be completely in upper case.

3.1.2 Implementation notes

TSSV will do all alignments and pattern matching against an explicitly upper case copy of each read. The library should therefore contain only upper case sequences. When a flanking sequence (as provided in the library file) has been detected in a read, TSSV will check whether the section of the read that corresponds to this flanking sequence contains any upper case letters. If this section of the read was completely lower case, the match is rejected.

3.2 References

Seyed Yahya Anvar, Kristiaan J. van der Gaag, Jaap W. F. van der Heijden, Marcel H. A. M. Veltrop, Rolf H. A. M. Vossen, Rick H. de Leeuw, Cor Breukel, Henk P. J. Buermans, J. Sjef Verbeek, Peter de Knijff, Johan T. den Dunnen, and Jeroen F. J. Laros, **TSSV: a tool for characterization of complex allelic variants in pure and mixed genomes.** Bioinformatics, first published online February 13, 2014. doi:10.1093/bioinformatics/btu068

- Abstract.
- Full text.

CHAPTER 4

Benchmarking test for the C library

First we make datasets, these are multiples of the `test.fa` file in the `data` directory. The original file contains 9 records, so the largest generated dataset will contain 9000 records.

..code:: sh

for j in 1 2 3 4 5 10 20 30 40 100 200 300 400 1000; do i=0 while [$i -le $j]; do

cat data/test.fa >> /tmp/test_$j.fa i=$((i + 1))

done

done

The tests are done with this snippet. We print the relative size of the dataset and the amount of (user) seconds it takes for the program to run.

..code:: sh

for i in 1 2 3 4 5 10 20 30 40 100 200 300 400 1000; do echo -n "$i " /usr/bin/time -f "%U" tssv /tmp/test_$i.fa data/library.csv -r /dev/null

done

Save the files in `old.dat` and `new.dat` respectively.

4.1 Speedup estimation

To estimate the speedup, we calculate the ratio of the raw run times for both tests. This results in a file containing the relative size of the dataset and the ratio of run times per test.

..code:: sh

IFS=" " for i in *paste -d ' ' old.dat new.dat*; do

echo -n "$(echo $i | cut -f 1 -d ' ') " echo 3k $(echo $i | cut -f 2 -d ' ') $(echo $i | cut -f 4 -d ' ') / p | dc

done > speed.dat

4.2 Plotting

We use `gnuplot` for visualisation:

The raw run times:

```
set terminal postscript color
set output "benchmark.eps"
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 7 ps 2
set style line 2 lc rgb '#dd181f' lt 1 lw 2 pt 5 ps 2
set xlabel "dataset size"
set ylabel "run time"
plot "old.dat" with linespoints ls 1, "new.dat" with linespoints ls 2
```

And the speedup.

```
set terminal postscript color
set output "speed.eps"
set style line 1 lc rgb '#0060ad' lt 1 lw 2 pt 7 ps 2
set xlabel "dataset size"
set ylabel "speedup"
plot "speed.dat" with linespoints ls 1
```

4.3 Results

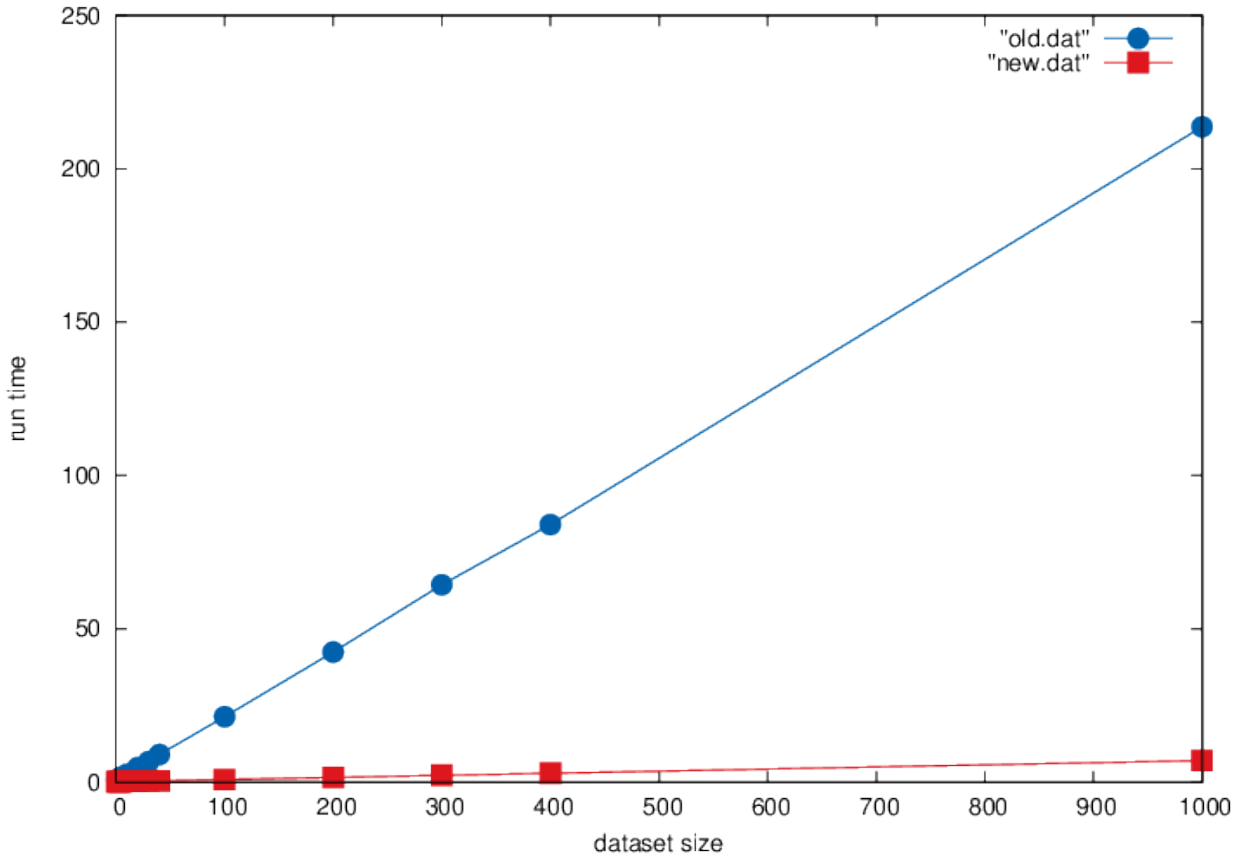
4.3.1 Raw run times

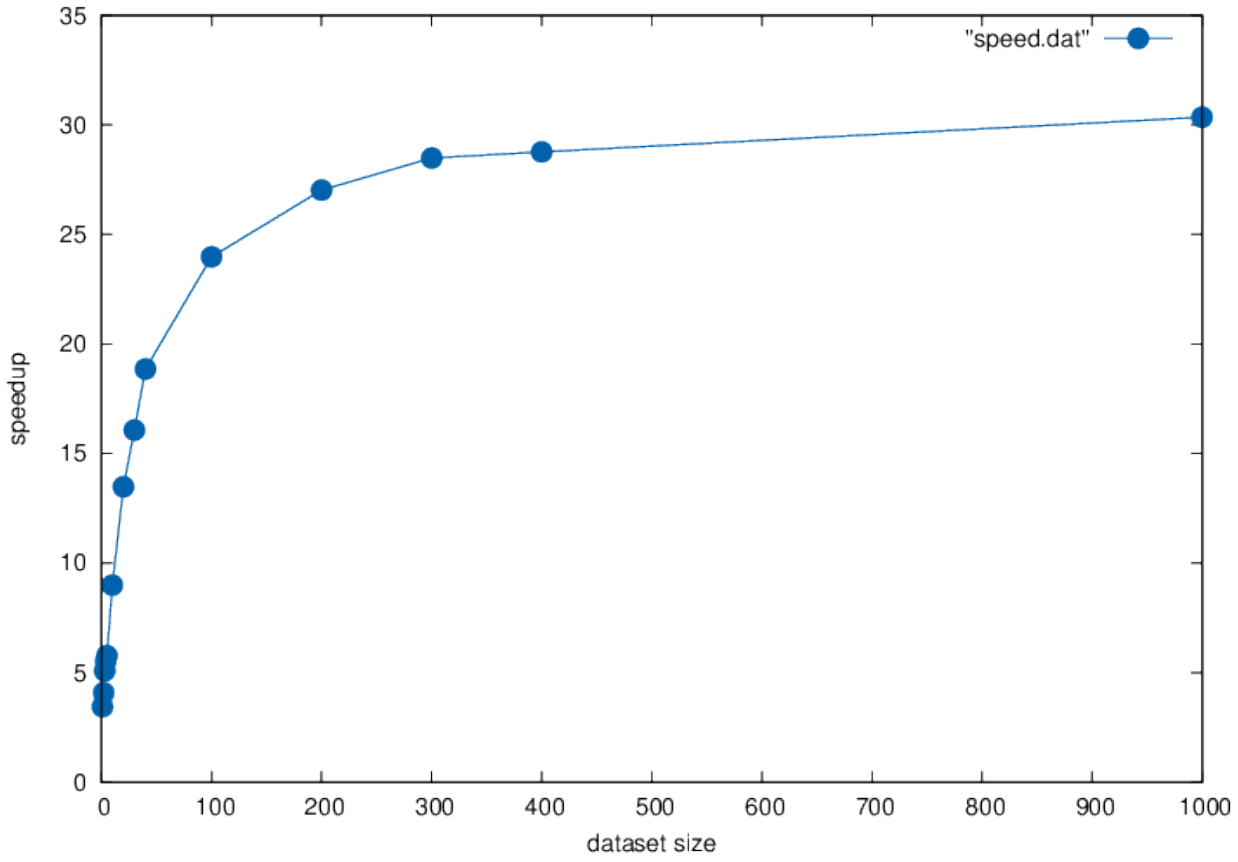
Both programs behave linearly in the size of the dataset (as expected). For an input of 9000 records, the old method took 3 minutes and 34 seconds, the new method takes 7 seconds.

The average run time per record is 0.023741 seconds for the old method and 0.000782 seconds for the new method. The speedup is around 30.35 times.

4.3.2 Speedup

We see that because of the overhead (loading libraries, reading files, etc.) the speedup is not constant in the size of the dataset. There seems to be an asymptote at around 31 or 32, which we think will be the average speedup for the new implementation.


"old.dat"
"new.dat"
run time
dataset size
0
50
100
150
200
250
0
100
200
300
400
500
600
700
800
900
1000


"speed.dat"
speedup
dataset size
0
5
10
15
20
25
30
35
0
100
200
300
400
500
600
700
800
900
1000

CHAPTER 5

Contributors

- Jaap W.F. van der Heijden <jwfvanderheijden@gmail.com> (Original author)
- Jerry Hoogenboom <j.hoogenboom@nfi.minvenj.nl> (TSSV lite, SSE2 alignment)
- Jeroen F.J. Laros <J.F.J.Laros@lumc.nl> (Original author, maintainer)
- Redmar van den Berg <Redmar@ubuntu.com> (Support for compressed input)

Find out who contributed:

```
git shortlog -s -e
```