Note S1. Methods

Note S1 Methods

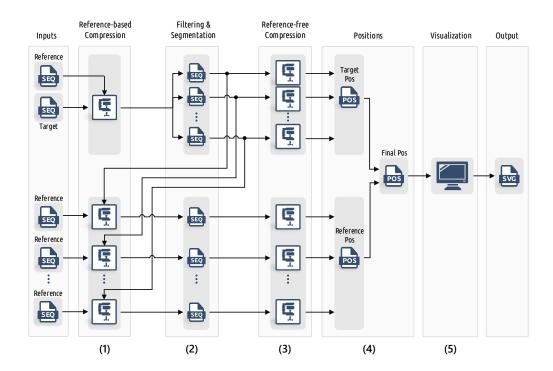


Fig. S1. The schema of Smash++.

Note S2 Experiment setup

S2.1 Datasets

 ${\bf Table~S1.}~{\rm Datasets~used~in~the~experiments}.$

Category	Reference	Length (base)	Target	Length (base)	Description
Synthetic	RefS	1,000	TarS	1,000	
Synthetic	RefM	100,000	TarM	100,000	
Synthetic	RefL	5,000,000	TarL	5,000,000	
Synthetic	RefXL	100,000,000	TarXL	100,000,000	

Note S3. Results 3

Note S3 Results

Smash++ and several other methods have been carried out on a collection of synthetic and real sequences. The machine used for the tests had an 8-core 3.40 GHz Intel[®] Core[™] i7-6700 CPU with 32 GB RAM.

Note S4 Tool availability and implementation

Smash++ is implemented in C++ language and is available at [1]. This tool is able to find rearrangements in sequences, Fasta and Fastq files; although, it is highly recommended to use sequences as input. To work with Smash++, it should be first installed. In the following sections, we describe installing and running Smash++.

S4.1 Install

In order to install Smash++, we run the following commands:

```
git clone https://github.com/smortezah/smashpp.git
cd smashpp
cmake .

make
```

S4.2 Run

Running

```
1 ./smashpp
```

provides the following guide:

```
SYNOPSIS
     ./smashpp [OPTIONS]... -r [REF-FILE] -t [TAR-FILE]
2
3
4
   SAMPLE
5
6
   DESCRIPTION
7
     Mandatory arguments
     -r, --ref FILE
                                reference file (Seq/Fasta/Fastq)
8
     -t, --tar FILE
9
                                target file
                                              (Seq/Fasta/Fastq)
10
11
     Options
12
     -v, --verbose
                                 more information
     -1, --level INT
13
                                level of compression: [0, 5]
14
     -m, --min INT
                                min segment size: [1, 4294967295]
15
     -nr, --no-redun
                                do NOT compute self complexity
                                Entropy of 'N's: [0.0, 100.0]
16
     -e, --ent-n FLOAT
     -n, --nthr INT
                                number of threads: [1, 8]
17
18
     -fs, --filter-scale {\tt S|M|L} scale of the filter:
19
                                 {S|small, M|medium, L|large}
     -w, --wsize INT
20
                                 window size: [1, 4294967295]
                                type of windowing function:
     -wt, --wtype INT/STRING
21
22
                                 {O|rectangular, 1|hamming, 2|hann,
23
                                 3|blackman, 4|triangular, 5|welch,
24
                                 6|sine, 7|nuttall}
     -d, --step
25
                  INT
                                sampling steps
26
     -th, --thresh FLOAT
                                threshold: [0.0, 20.0]
     -sp, --save-profile
27
                                save profile (*.prf)
28
     -sf, --save-filter
                                 save filtered file (*.fil)
     -sb, --save-seq
                                save sequence (input: Fasta/Fastq)
29
     -ss, --save-segment
                                save segmented files (*-s_i)
30
                                 save profile, filetered and
     -sa, --save-all
31
32
                                 segmented files
     -h, --help
33
                                 usage guide
     -rm, --ref-model k, [w,d,]ir,a,g/t,ir,a,g:...
34
     -tm, --tar-model k,[w,d,]ir,a,g/t,ir,a,g:...
35
36
                                 parameters of models
                           (INT) k: context size
37
38
                           (INT) w: width of sketch in log2 form,
39
                                     e.g., set 10 for w=2^10=1024
                           (INT) d: depth of sketch
40
                           (INT) ir: inverted repeat: {0, 1, 2}
41
```

S4.3. Example 5

```
0: regular (not inverted)
1: inverted, solely
2: both regular and inverted
45 (FLOAT) a: estimator
46 (FLOAT) g: forgetting factor: [0.0, 1.0)
47 (INT) t: threshold (no. substitutions)
```

The arguments "-r" and "-t" are used to specify the reference and the target files. It is highly recommended to choose short names for these files.

Here, on reference-based compression, we have replaced 'N' bases in the references with 'A's and 'N' bases in the targets with 'T's. Also, on reference-free compression, we have replaced 'N's in the references and the targets with 'A's. If a user tends to replace 'N' bases in a sequence with a normal distribution of 'A', 'C', 'G' and 'T's, he/she can employ GOOSE toolkit [2].

S4.3 Example

This section guides, step-by-step, employing Smash++ to find rearrangements.

Install Smash++ and provide the required files

First, we install Smash++:

```
git clone https://github.com/smortezah/smashpp.git
cd smashpp
cmake .
```

Then, we copy Smash++'s binary file into example/ directory and go to that directory:

```
cp smashpp example/
cd example/
```

References 6

References

[1] M. Hosseini, D. Pratas, and A. J. Pinho. Smash++. [Online]. Available: https://github.com/smortezah/smashpp

 $[2]\,$ D. Pratas. Goose. [Online]. Available: https://github.com/pratas/goose