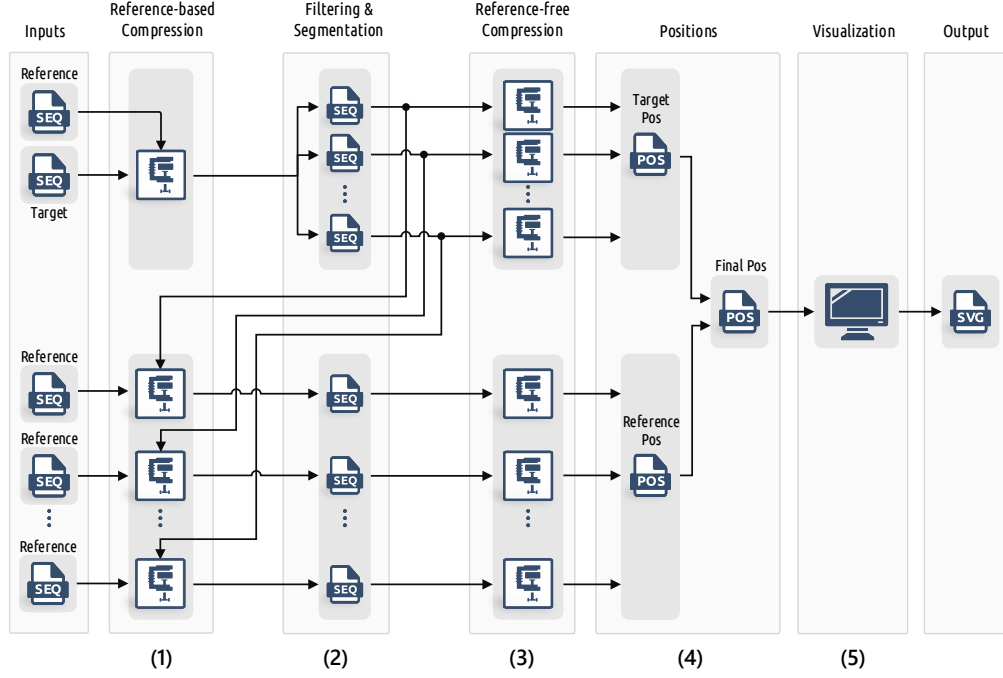# Note S1   Methods

Fig. S1. The schema of Smash++.

S1.1   Building models of the data

S1.2   finding similar regions

In order to smooth the profile information, we use Hann window, which is a discrete window function given by

$$w[n] = 0.5 \left[1 - \cos\left(\frac{2\pi n}{N}\right) \right] = \sin^2\left(\frac{\pi n}{N}\right), \qquad \textbf{(Eq. S1)}$$

in which $0 \leq n \leq N$ and length of the window is $N + 1$.

S1.3   Computing complexities

S1.4   The software

Besides Hann window, that is used as default to filter the profile information obtained by the reference-based compression, we have implemented several other window functions.

Note S2   Experiment setup

S2.1   Datasets

Table S1. 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

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:

```
1  git clone https://github.com/smortezah/smashpp.git
2  cd smashpp
3  cmake .
4  make
```

S4.2   Run

Running

```
1  ./smashpp
```

provides the following guide:

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

```
42                                      0: regular (not inverted)
43                                      1: inverted, solely
44                                      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++:

```
1  git clone https://github.com/smortezah/smashpp.git
2  cd smashpp
3  cmake .
4  make
```

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

```
1  cp smashpp example/
2  cd example/
```

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