Note S1 Methods

The schema of the proposed method is illustrated in Fig. S1. There are five major stages in this method, including

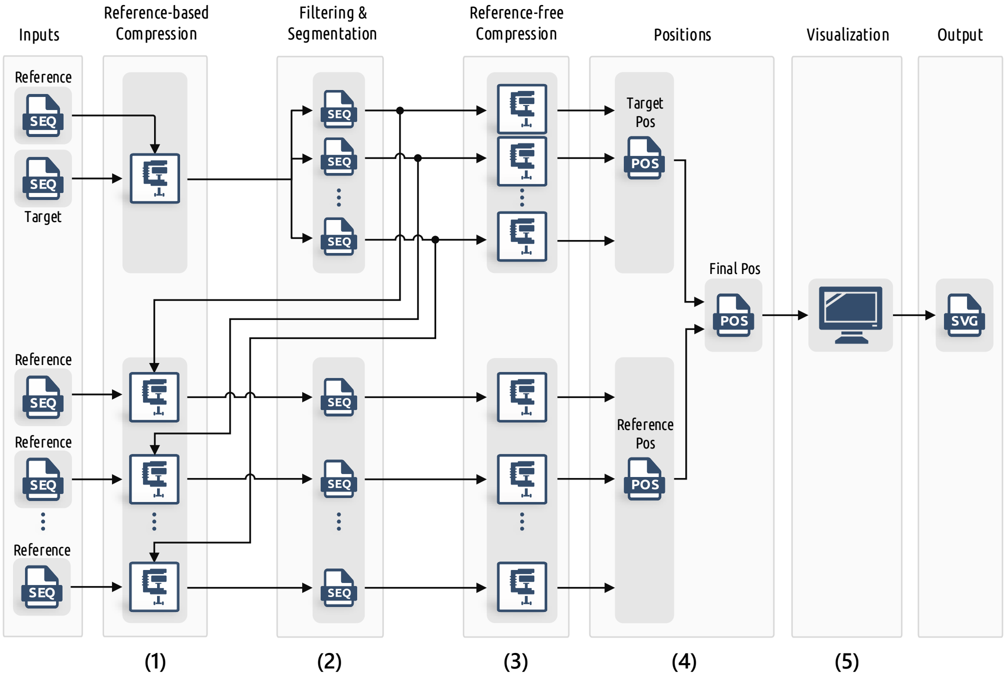

Fig. S1. The schema of Smash++.

S1.1 Building models of the data

S1.2 finding similar regions

In order to smooth the profile information, we use Hann window [1], which is a discrete window function given by

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

in which, $0 \leq n \leq N$ and length of the window is $N+1$ (Fig. S2).

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 (Fig. S3), including Blackman [1], Hamming [2], Nuttall [3], rectangular [4], sine [5], triangular [6] and Welch [7]

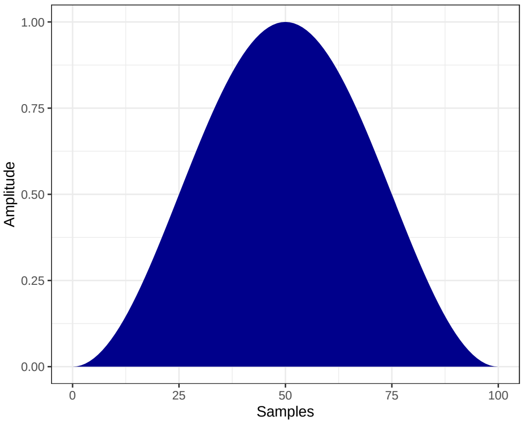

Fig. S2. Hann window for 101 samples.

windows. These functions are given by

$$
\begin{aligned}
w[n] &= 1, && \text{(rectangular)}\\
w[n] &= 1 - \left|\frac{n-N/2}{L/2}\right|, \quad L = N, && \text{(triangular/Bartlett)}\\
w[n] &= 1 - \left(\frac{n-N/2}{N/2}\right)^2, && \text{(Welch)}\\
w[n] &= \sin\left(\frac{\pi n}{N}\right), && \text{(sine)}\\
w[n] &= 0.54348 - 0.45652\ \cos\left(\frac{2\pi n}{N}\right), && \text{(Hamming)}\\
w[n] &= 0.42659 - 0.49656\ \cos\left(\frac{2\pi n}{N}\right) + 0.07685\ \cos\left(\frac{4\pi n}{N}\right), && \text{(Blackman)}\\
w[n] &= 0.35577 - 0.48740\ \cos\left(\frac{2\pi n}{N}\right) + 0.14423\ \cos\left(\frac{4\pi n}{N}\right) - 0.01260\ \cos\left(\frac{6\pi n}{N}\right). && \text{(Nuttall)}
\end{aligned}
$$

(Eq. S2)

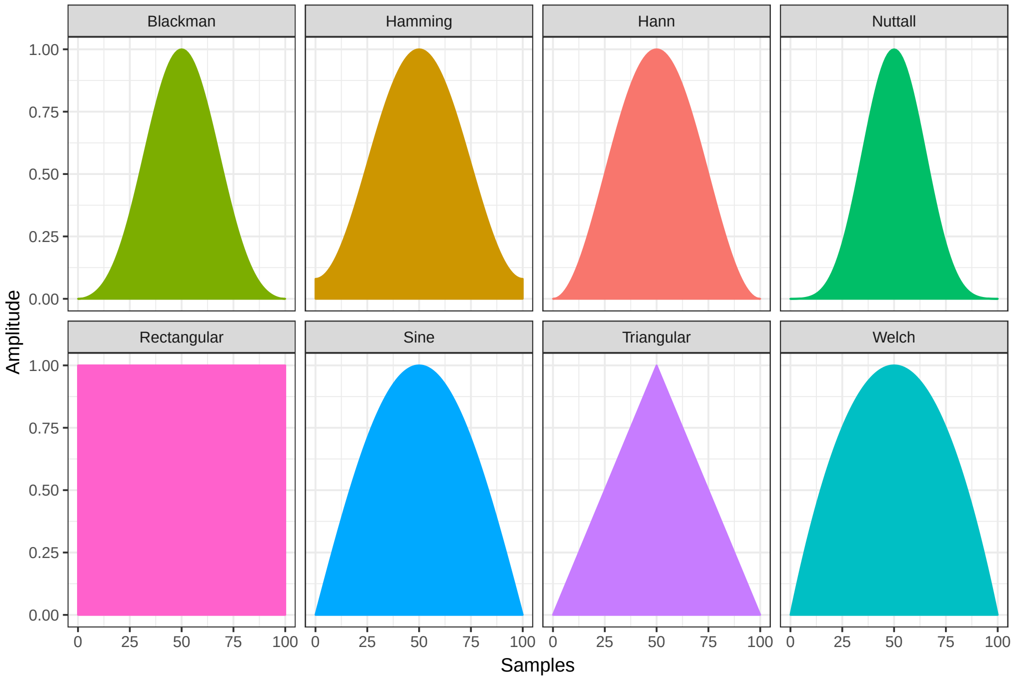

Fig. S3. Window functions.

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 [8]. 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

```
./smashpp
```

provides the following guide:

```
SYNOPSIS
  ./smashpp [OPTIONS]...  -r [REF-FILE] -t [TAR-FILE]

SAMPLE

DESCRIPTION
  Mandatory arguments
  -r,  --ref FILE            reference file (Seq/Fasta/Fastq)
  -t,  --tar FILE            target file    (Seq/Fasta/Fastq)

  Options
  -v,  --verbose             more information
  -l,  --level INT           level of compression: [0, 5]
  -m,  --min   INT           min segment size: [1, 4294967295]
  -nr, --no-redun            do NOT compute self complexity
  -e,  --ent-n FLOAT         Entropy of ’N’s: [0.0, 100.0]
  -n,  --nthr  INT           number of threads: [1, 8]
  -fs, --filter-scale S|M|L  scale of the filter:
                             {S|small, M|medium, L|large}
  -w,  --wsize INT           window size: [1, 4294967295]
  -wt, --wtype INT/STRING    type of windowing function:
                             {0|rectangular, 1|hamming, 2|hann,
                             3|blackman, 4|triangular, 5|welch,
                             6|sine, 7|nuttall}
  -d,  --step   INT          sampling steps
  -th, --thresh FLOAT        threshold: [0.0, 20.0]
  -sp, --save-profile        save profile (*.prf)
  -sf, --save-filter         save filtered file (*.fil)
  -sb, --save-seq            save sequence (input: Fasta/Fastq)
  -ss, --save-segment        save segmented files (*-s_i)
  -sa, --save-all            save profile, filetered and
                             segmented files
  -h,  --help                usage guide
  -rm, --ref-model  k,[w,d,]ir,a,g/t,ir,a,g:...
  -tm, --tar-model  k,[w,d,]ir,a,g/t,ir,a,g:...
                             parameters of models
                       (INT) k:  context size
                       (INT) w:  width of sketch in log2 form,
                                 e.g., set 10 for w=2^10=1024
                       (INT) d:  depth of sketch
                       (INT) ir: inverted repeat: {0, 1, 2}
```

```
                                 0: regular (not inverted)
                                 1: inverted, solely
                                 2: both regular and inverted
                     (FLOAT) a:  estimator
                     (FLOAT) g:  forgetting factor: [0.0, 1.0)
                       (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 [9].

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 .
make
```

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

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

References

[1] R. Blackman and J. Tukey, “Particular pairs of windows,” *The measurement of power spectra, from the point of view of communications engineering*, pp. 95–101, 1959.

[2] J. W. Tukey and R. W. Hamming, *Measuring noise color.* Bell Telephone Laboratories, 1949.

[3] A. Nuttall, “Some windows with very good sidelobe behavior,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 1, pp. 84–91, 1981.

[4] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing.* Upper Saddle River, NJ: Prentice Hall, 1999.

[5] F. J. Harris, “On the use of windows for harmonic analysis with the discrete Fourier transform,” *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[6] M. S. Bartlett, “Periodogram analysis and continuous spectra,” *Biometrika*, vol. 37, no. 1/2, pp. 1–16, 1950.

[7] P. Welch, “The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms,” *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

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

[9] D. Pratas. Goose. [Online]. Available: https://github.com/pratas/goose