Note S1 Methods

The schema of the proposed method is illustrated in Fig. S1. There are five major stages in Smash++: (1) reference-based compression, (2) filtering and segmentation, (3) reference-free compression, (4) positions aggregation and (5) visualization. In stage (1), model of the reference is built and the target is compressed using that model.

The following sections describe the process in detail.

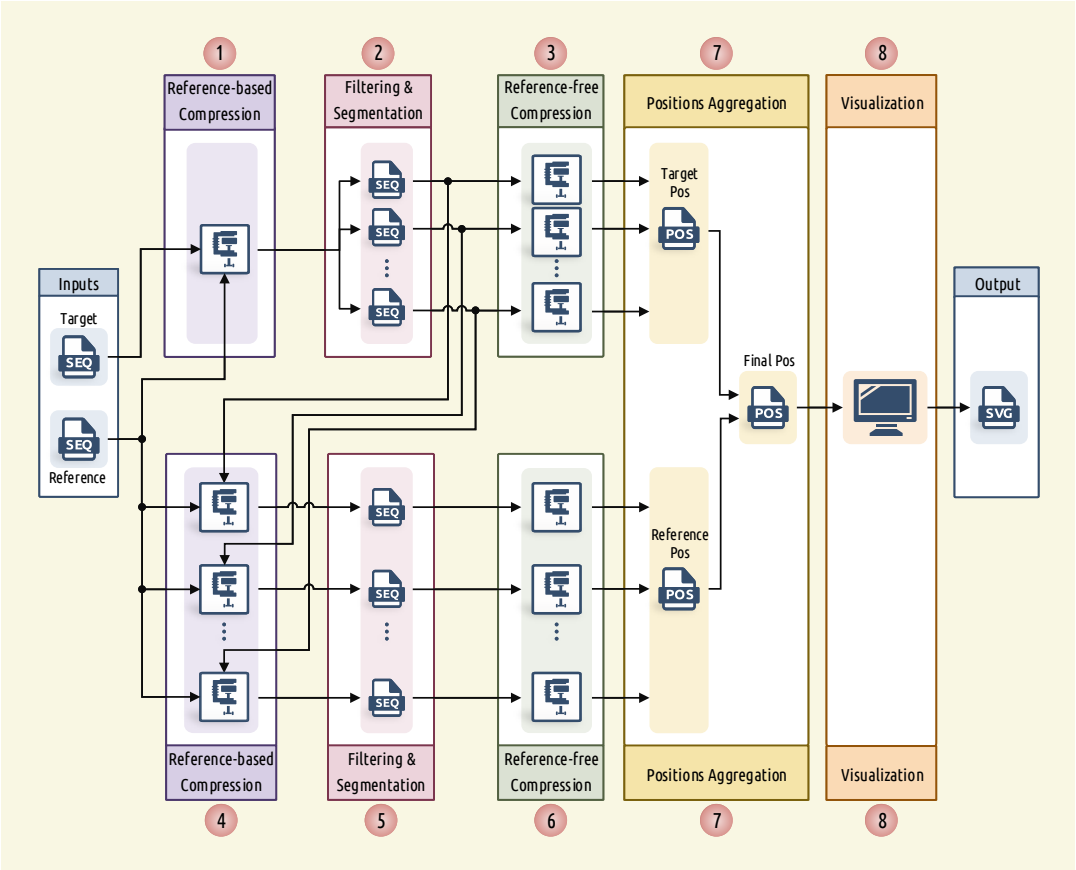

Fig. S1. The schema of Smash++.

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

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

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]

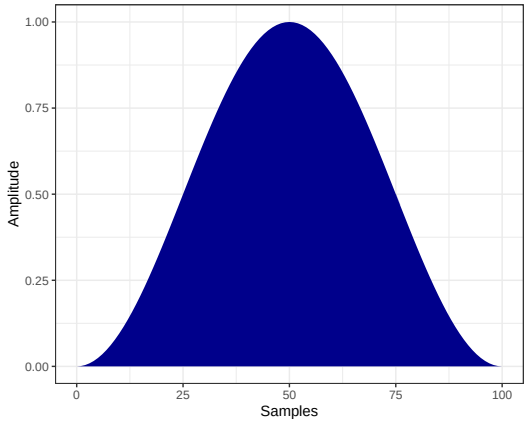

Fig. S2. Hann window for 101 samples.

windows. These functions are given by

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

(Eq. S2)

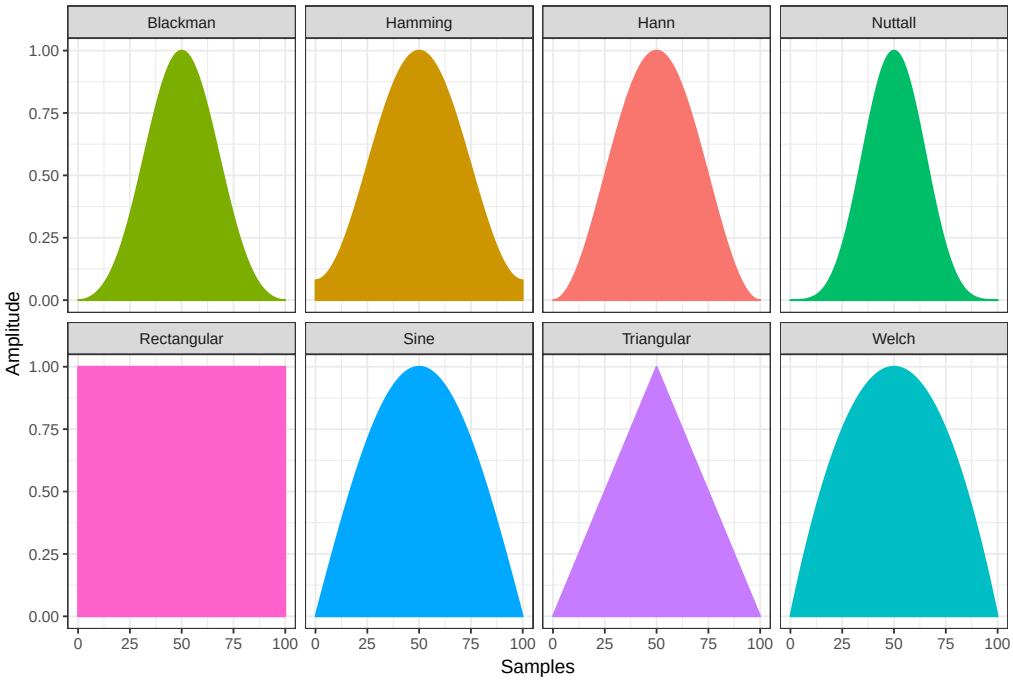

Fig. S3. Window functions.

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 and visualize rearrangements in sequences, FASTA and FASTQ files; although, it is highly recommended to use sequences as input. In the following sections, we describe installing and running Smash++.

S4.1 Install

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

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

S4.2 Run

A reference file and a target file are clearly mandatory to run Smash++ (without visualization). 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,  --filt_size INT           window size: [1, 4294967295]
  -wt, --filt_type 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]
  -sb, --save-seq            save sequence (input: Fasta/Fastq)
  -sp, --save-profile        save profile (*.prf)
  -sf, --save-filter         save filtered file (*.fil)
  -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, respectively, which are highly recommended to have short names. Level of compression, that is an integer between 0 and 5, can be determined with "-l". By setting "-m" to an integer value, only those regions in the reference file that are bigger than that value can be considered for compression. Triggering "-nr" makes the tool not to perform the reference-free compression (self-complexity computation) part.

In implementation of the reference-based compression, we have replaced 'N' bases in the references and the targets with 'A's and 'T's, respectively. On reference-free compression, they are replaced with 'A's, in both references and targets. 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]. Note that we have set the entropy of 'N's to 2.0, by default, but it is possible for the user to set them to another value of interest, by "-e" option.

Building different finite-context models can be done in the multi-threaded fashion, setting "-n" to an integer. To find similar regions in the reference and the target, information profile (obtained by compression) needs to be filtered, of which the scale can be set as S (small), M (medium) or L (large). Size of the window and type of the windowing function, described in S1.4, can be set by "-w" and "-wt" options, respectively. Instead of considering the complete profile information, the user is able to make samples of it by steps of which size can be determined by "-d". For the purpose of segmenting the filtered information profile, the average entropy of reference-based compression is used as the threshold, by default. However, this threshold can be altered by "-th" option.

Smash++ accepts FASTA and FASTQ files as input, in addition to sequences. In these cases, the input files are converted to sequences and then processed further. It is possible to save these sequences by "-sb" option.

After obtaining the information profile, Smash++ filters it and then removes it, by default. However, it is possible to save the profile by "-sp" option. The same thing happens to the filtered file, i.e., it is segmented and then is removed. But, the user can use "-sf" to save the filtered file. Also, the segmented files can be saved using "-ss". The user can save all the information profile, filtered and segmented files, by triggering "-sa" option.

For the purpose of compression, either reference-based or reference-free, it is recommended to use "-l" option, since it configures the models automatically. However, using "-rm" and "-tm", the user would be able to manually configure the reference model, for reference-based compression, and the target model, for reference-free compression. Parameters of the models are described in detail in section S1.

Running Smash++ (without visualization), positions of the similar regions in the reference and the target, and also complexity of the regions is saved in a "*.pos" file. To visualize this file, one can type

```
./smashpp -viz
```

which gives

```
SYNOPSIS
  ./smashpp -viz  OPTIONS...  -o SVG-FILE  POS-FILE

SAMPLE

DESCRIPTION
  Mandatory arguments:
  POS-FILE                positions file, generated by
                          Smash++ tool (*.pos)

  Options:
  -v,  --verbose          more information
  -o,  --out SVG-FILE     output image name (*.svg)
```

```
  -rn, --ref-name STRING   reference name shown on output. If name
                           has space, use "s, e.g. "Seq label".
                           Default: name in header of position file.
  -tn, --tar-name STRING   target name shown on output
  -vv, --vertical          vertical view
  -nn, --no-nrc            do NOT show normalized
                           relative compression (NRC)
  -nr, --no-redun          do NOT show self complexity
  -ni, --no-inv            do NOT show inverse maps
  -ng, --no-reg            do NOT show regular maps
  -l,  --link     INT      type of the link between maps: [1, 6]
  -c,  --color    INT      color mode: [0, 2]
  -p,  --opacity  FLOAT    opacity: [0.0, 1.0]
  -w,  --width    INT      width of the sequence: [15, 100]
  -s,  --space    INT      space between sequences: [15, 200]
  -f,  --mult     INT      multiplication factor for
                           color ID: [1, 255]
  -b,  --begin    INT      beginning of color ID: [0, 255]
  -rt, --ref-tick INT      reference tick: [1, 4294967295]
  -tt, --tar-tick INT      target tick: [1, 4294967295]
  -th, --tick-human 0|1    tick human readable: 0=false, 1=true
  -m,  --min      INT      minimum block size: [1, 4294967295]
  -h,  --help              usage guide
```

The output of Smash++ visualizer is an "SVG" file, which its name is determined by "-o" option. By default, it is named "map.svg". Names of the reference and the target, which are going to be printed on the output image, can be altered by "-rn" and "-tn", respectively. They are by default the names written in the positions file. To have a vertical view of the image, instead of the default horizontal view, one can use "-vv" trigger.

Smash++ performs reference-based and reference-free compressions to calculate the normalized relative compression (NRC) and redundancy (self complexity), respectively. If the user is not interested in showing them, he/she can turn them off by "-nn" and "-nr" triggers. In addition, Smash++ considers both regular and reverse complement maps by default in its calculations. Triggering "-ni" and "-ng" will stop showing inverted and regular maps, respectively.

Options "-l", "-c", "-p", "-w", "-s", "-f" and "-b" can be used to change the appearance of the image. Assigning integers to "-rt" and "-tt" options will change the tick sizes of the reference and the target, respectively. Smash++ prints the sizes on axes in human readable format, e.g., 1K, 2M, etc. However, it can be triggered by "-th" option. Note that, here, 1K is equivalent to 1000 and not 1024. Finally, by setting "-m" to an integer value, only the regions that are bigger than that value will be illustrated.

S4.3 Example

This section guides, step-by-step, employing Smash++ to find and visualize rearrangements in a sample genomic data.

Install Smash++ and provide the required files

First, we install Smash++:

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

Then, we copy `smashpp` binary file into `example/` directory and go to that directory:

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

In this directory, a 1000 byte reference sequence, `ref`, and a 1000 byte target sequence, `tar`, are provided. Running

```
./smashpp -r refs -t tars -w 45 -l 3
./smashpp -viz refs.tars.pos
```

results in Fig. S4, which is saved as “map.svg”.

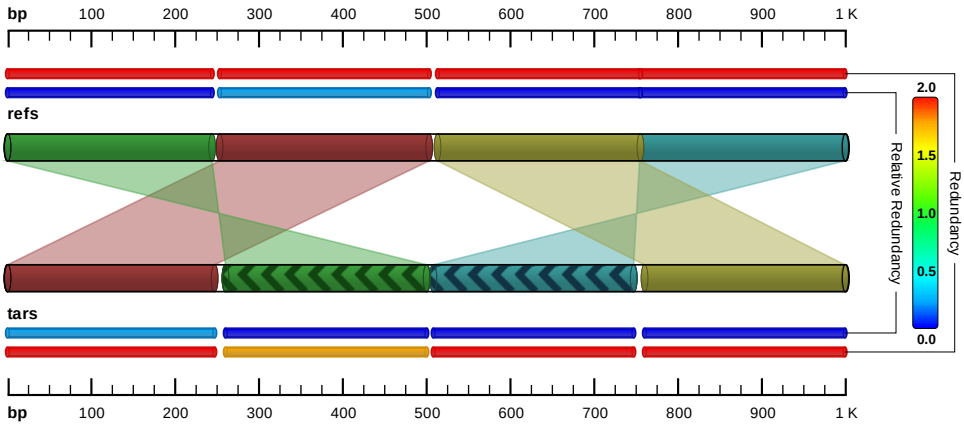

Fig. S4. The result of running Smash++ on ...

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