Note S1 Tool availability and implementation

Smash++ is implemented in C++ language and is available at [1]. 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 the Smash++ tool.

S1.1 Install

To install Smash++ on various operating systems, follow the instructions below. Note that the precompiled executables are available for 64 bit operating systems in the "bin/" directory.

Linux

- Install "git" and "cmake":

```
sudo apt update
sudo apt install git cmake
```

- Clone Smash++ and install it:

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

macOS

- Install "Homebrew", "git" and "cmake":

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
    install/master/install)"
brew install git cmake
```

- Clone Smash++ and install it:

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

Windows

- Download and install "CMake", e.g., from https://github.com/Kitware/CMake/releases/download/v3.14.4/cmake-3.14.4-win64-x64.msi. Make sure to add it to the system PATH. For example, if CMake is installed in "C:\Program Files", add "C:\Program Files\CMake\bin" to the system PATH.
- Download and install "mingw-w64", e.g., from https://sourceforge.net/projects/mingw-w64/files/latest/download. Make sure to add it to the system PATH. For example, if it is installed in "C:\mingw-w64", add "C:\mingw-w64\mingw64\bin" to the system PATH.
- Download and install "git", from https://git-scm.com/download/win.
- Clone Smash++ and install it:

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

S1.2 Run Smash++

Various options are provided with the interface of the proposed tool, which are described in Table S1. The commands for running Smash++ have the form of the following:

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

Table S1. Options provided by Smash++ interface.

Flag	Input	Description
Required		
`-r`	Seq/FASTA/FASTQ	Reference file.
`-t`	Seq/FASTA/FASTQ	Target file.
		It is highly recommended to have short names.
Optional		
`-l`	Integer: $[0, 5]$ Default: 0	Level of compression.
`-m`	Integer: $[1, 2^{32} - 1]$ Default: 50	Minimum segment size. Only those regions in the reference file that are bigger than this value would be able to be considered for compression.
`-e`	Float: $[0.0, 100.0]$ Default: 2.0	Entropy of 'N's. 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 [2].
`-n`	Integer: $[1, 8]$ Default: 4	Number of threads. Creating multiple finite-context models and substitutional-tolerant Markov models can be done in a multi-threaded fashion.
`-f`	Integer: $[1, 2^{32} - 1]$ Default: 256	Filter size. In the process of finding similar regions in the reference and the target sequences, the information content that would be obtained by compression needs to be filtered.
`-ft`	Integer or String: {0 | rectangular, 1 | hamming, 2 | hann, 3 | blackman, 4 | triangular, 5 | welch, 6 | sine, 7 | nuttall} Default: hann	Filter type (windowing function). Besides Hann window that is used by default to smooth the information content (profile), we have implemented several other windowing functions: Blackman [3], Hamming [4], Nuttall [5], rectangular [6], sine [7], triangular [8] and Welch [9] windows. These functions are given by Equation S1 and are plotted in Fig. S1.
`-fs`	Char or String: {S | small, M | medium, L | large} Default: large	Filter scale. Automatically chooses filter size.
`-d`	Integer: $[1, 2^{64} - 1]$ Default: 1	Sampling steps. Instead of considering the whole information content, this option can be used to make samples of it.
`-th`	Float: $[0.0, 20.0]$ Default: 1.5	threshold. For the purpose of segmenting the filtered information content, the average entropy of reference-based compression is used by default as the threshold, but the threshold can be altered by this option.
`-rb`	Integer: $[-2^{15}, 2^{15} - 1]$	Reference beginning guard.
`-re`	Integer: $[-2^{15}, 2^{15} - 1]$	Reference ending guard.
`-tb`	Integer: $[-2^{15}, 2^{15} - 1]$	Target beginning guard.
`-te`	Integer: $[-2^{15}, 2^{15} - 1]$	Target ending guard.

	Default: 0	Smash++ is capable of finding even very small similar regions in two sequences. However, we have found experimentally that when it is running in a very sensitive mode, there might be some cases in which the size of similar regions in the reference and the target are not balanced. These cases can be handled by "-rb", "-re", "-tb" and "-te" options, that can resize the beginning and ending guards of reference and target regions, respectively. For example, if "-tb 10" is used, the first 10 bases in each target region will be ignored, which results in smaller regions. Note that when the guard sizes of target regions are increased, the models built from these regions would be slightly different than the original models; consequently, the sizes of reference regions that are detected as being similar to the ones from the target would be modified, as well. Therefore, changing the guard sizes of target regions will affect the sizes of reference regions. In the case of activating deep compression, by "-dp", changing the guard sizes of reference regions would affect the sizes of target regions, as well.
`-dp`	—	Deep compression. The "deep compression" means that similar regions in target and reference sequences are found in three phases instead of two: 1) the model of the reference is built and the target is compressed based upon that model, 2) the model of each detected region is built and the whole reference is compressed based on these models and 3) the model of each detected reference region is built and the corresponding target regions will be compressed based on that model.
`-nr`	—	Do not compute self complexity. It makes the tool not to perform the reference-free compression (self-complexity computation).
`-sb`	—	Save sequence (input: FASTA/FASTQ). Smash++ accepts FASTA and FASTQ files as input, in addition to sequences. In these cases, the input files are first converted to sequences and then processed further. It is possible to save these sequences by this option.
`-sp`	—	Save profile, *.prf. When the information profile is obtained, Smash++ smoothens then removes it by default. However, it can be preserved by this option.
`-sf`	—	Save filtered file, *.fil. The filtered profile is segmented then removed, by default; however, it can be preserved by this option.
`-ss`	—	Save segmented files, *.s$_i$.
`-sa`	—	Save profile, filtered and segmented files.
`-rm` `-tm`	k,[w,d,]ir,α,γ/t,ir,α,γ:... k,[w,d,]ir,α,γ/t,ir,α,γ:... Default: $14, 0, 0.005, 0.95$	Parameters of reference models. Parameters of target models: • k (integer > 1): context size, • w (integer $< 2^{64} - 1$): sketch width in $\log_2$ form, e.g., set 10 for $w = 2^{10} = 1024$, • d (integer > 0): sketch depth, • ir (integer: $\{0, 1, 2\}$): inverted repeat, including 0: regular (not inverted), 1: inverted solely, and 2: both regular and inverted, • α (float > 0): estimator, • γ (float: $[0.0, 1.0)$): forgetting factor, • t (integer > 0): threshold (number of substitutions). It is recommended for compression 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, respectively. Parameters of the models are described in detail in section **??**.
`-ll`	—	Show list of parameters that would be chosen automatically for each model.
`-h`	—	Usage guide.
`-v`	—	More information (verbose).
`--version`		Show version.

$$
\begin{aligned}
&w[n] = 1, && \text{(rectangular)}\\
&w[n] = 1 - \left|\frac{n-N/2}{L/2}\right|, \quad L = N, && \text{(triangular/Bartlett)}\\
&w[n] = 1 - \left(\frac{n-N/2}{N/2}\right)^2, && \text{(Welch)}\\
&w[n] = \sin\left(\frac{\pi n}{N}\right), && \text{(sine)}\\
&w[n] = 0.54348 - 0.45652\ \cos\left(\frac{2\pi n}{N}\right), && \text{(Hamming)}\\
&w[n] = 0.42659 - 0.49656\ \cos\left(\frac{2\pi n}{N}\right) + 0.07685\ \cos\left(\frac{4\pi n}{N}\right), && \text{(Blackman)}\\
&w[n] = 0.35577 - 0.48740\ \cos\left(\frac{2\pi n}{N}\right) + 0.14423\ \cos\left(\frac{4\pi n}{N}\right) - 0.01260\ \cos\left(\frac{6\pi n}{N}\right), && \text{(Nuttall)}
\end{aligned}
\qquad \textbf{(Eq. S1)}
$$

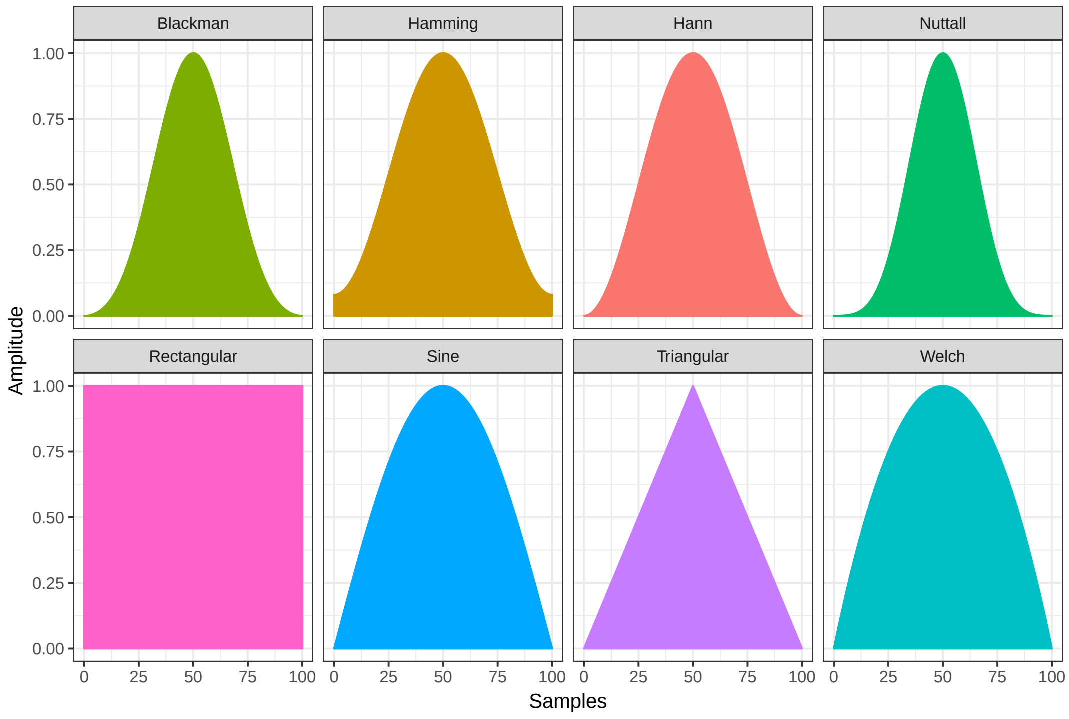

Fig. S1. Various windowing functions implemented and embedded in Smash++.

By running Smash++, positions of the similar regions in reference and target sequences, and also complexity of the regions is saved in a ".pos" file. This tab-separated file has a header including:

1. The string "#SMASHPP" as a specifier for the Smash++ tool,
2. The name of reference sequence,
3. The size of reference sequence,
4. The name of target sequence,
5. The size of target sequence,

and a body including:

1. Initial position of the reference sequence,

2. Final position of the reference sequence,
3. Average entropy of compressing the associated target block considering this reference block as the reference,
4. Complexity (average entropy) of the detected reference block, calculated by reference-free compression,
5. Initial position of the target sequence,
6. Final position of the target sequence,
7. Average entropy of compressing the associated reference block considering this target block as the reference,
8. Complexity of the detected target block, calculated by reference-free compression.

As an example, the header of the following ".pos" file shows that the reference "Ref" and the target "Tar" are 5,000,000 bases long. The body shows that there is a block in the Ref, from the position 2000000 up to 3000000, which is similar to a block in the Tar, from the position 3000000 to 4000000. Average entropy of compressing the Tar block, using the Ref block as reference, is 0.255555. This number is 0.26666 when the Ref block is compressed based on the model of the Tar block. Also, complexities of the Ref and the Tar blocks are 1.97777 and 1.98888, respectively.

```
#SMASHPP Ref      5000000  Tar      5000000
2000000  3000000  0.26666  1.97777  3000000  4000000  0.25555 1.98888
```

S1.3 Run Smash++ visualizer

The position file obtained by running Smash++ can be visualized by

```
./smashpp -viz
```

The visualizer provides various options that are described in Table S2. The commands for running Smash++ visualizer are of the form

```
  ./smashpp -viz [OPTIONS]  -o <SVG-FILE>  <POS-FILE>
```

Table S2. Options provided by Smash++ visualizer interface.

Flag	Input	Description
Required		
	*.pos file	Position file, generated by Smash++ tool.
Optional		
`-o`	*.svg file Default: map.svg	Output image name.
`-rn` `-tn`	String String Default: names in position file's header	Reference name shown on output. Target name shown on output. If it has space, use double quotes, e.g. "Seq label".
`-l`	Integer: $[1, 6]$ Default: 1	Type of the link between maps.
`-c`	Integer: $[0, 1]$ Default: 0	Color mode.
`-p`	Float: $[0.0, 1.0]$ Default: 0.9	Opacity.
`-w`	Integer: $[15, 100]$ Default: 16	Width of the sequence.

`-s`	Integer: $[15, 200]$ Default: 62	Space between sequences.
`-f`	Integer: $[1, 255]$ Default: 43	Multiplication factor for color ID.
`-b`	Integer: $[0, 255]$ Default: 0	Beginning of color ID.
`-rt` `-tt`	Integer: $[1, 2^{32} - 1]$ Integer: $[1, 2^{32} - 1]$	Reference tick size. Target tick size.
`-th`	Integer: $\{0, 1\}$ Default: 1	Tick human readable: 0=false, 1=true. If it is true, the sizes on axes are printed in the format 1K, 2M, etc. Note that here, 1K is equivalent to 1000 and not 1024, and so on.
`-m`	Integer: $[1, 2^{32} - 1]$ Default: 1	Minimum block size. Only the regions that are bigger than this value will be illustrated.
`-vv`	—	Vertical view of the output image.
`-nn` `-nr`	— —	Do not show normalized relative compression (NRC). Do not show self complexity. Smash++ performs reference-based and reference-free compressions to calculate the 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.
`-ni` `-ng`	— —	Do not show inverse maps. Do not show regular (not inverse) maps. Smash++ considers by default both regular and reverse complement maps in its calculations.
`-h`	—	Usage guide.
`-v`	—	More information (verbose).
`--version`		Show version.

S1.4 Example

This section guides step-by-step employing Smash++ to find and visualize rearrangements in a sample genomic data. Note that the commands can be run on Linux and macOS, however, they are similar in Windows.

Install Smash++ and provide the required files

First, install Smash++:

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

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

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

There is a 1000 byte reference sequence, named `refs`, and a 1000 byte target sequence, named `tars`, in this directory. Running

```
./smashpp -r refs -t tars -f 45 -l 3
./smashpp -viz -p 1 -s 50 -w 15 refs.tars.pos
```

results in Fig. S2, which has been saved as "map.svg".

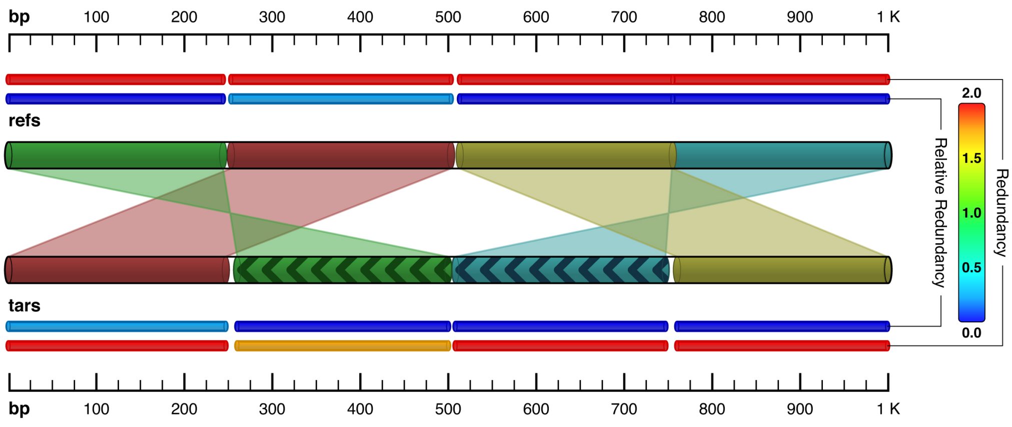

Fig. S2. An example of running Smash++ on two 1000 base sequences. Two similar regions in regular mode and two similar ones in inverted mode are detected.

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

[3] 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.

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

[5] 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.

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

[7] 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.

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

[9] 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.