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":

```
1  sudo apt update
2  sudo apt install git cmake
```

- Clone Smash++ and install it:

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

macOS

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

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

- Clone Smash++ and install it:

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

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

S1.2   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>

OPTIONS
  Required:
  -r  <FILE>          = reference file (Seq/FASTA/FASTQ)
  -t  <FILE>          = target file    (Seq/FASTA/FASTQ)

  Optional:
  -l  <INT>           = level of compression: [0, 5]. Default -> 0
  -m  <INT>           = min segment size: [1, 4294967295]    -> 50
  -e  <FLOAT>         = entropy of 'N's: [0.0, 100.0]        -> 2.0
  -n  <INT>           = number of threads: [1, 8]            -> 4
  -f  <INT>           = filter size: [1, 4294967295]         -> 256
  -ft <INT/STRING>    = filter type (windowing function):    -> hann
                        {0|rectangular, 1|hamming, 2|hann,
                        3|blackman, 4|triangular, 5|welch,
                        6|sine, 7|nuttall}
  -fs [S][M][L]       = filter scale:                        -> L
                        {S|small, M|medium, L|large}
  -d  <INT>           = sampling steps                       -> 1
  -th <FLOAT>         = threshold: [0.0, 20.0]               -> 1.5
  -rb <INT>           = ref beginning guard: [-32768, 32767] -> 0
  -re <INT>           = ref ending guard: [-32768, 32767]    -> 0
  -tb <INT>           = tar beginning guard: [-32768, 32767] -> 0
  -te <INT>           = tar ending guard: [-32768, 32767]    -> 0
  -dp                 = deep compression                     -> no
  -nr                 = do NOT compute self complexity       -> no
  -sb                 = save sequence (input: FASTA/FASTQ)   -> no
  -sp                 = save profile (*.prf)                 -> no
  -sf                 = save filtered file (*.fil)           -> no
  -ss                 = save segmented files (*.s[i])        -> no
  -sa                 = save profile, filetered and          -> no
                        segmented files
  -rm k,[w,d,]ir,a,g/t,ir,a,g:...
  -tm 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)
  -ll                 = list of compression levels
  -h                  = usage guide
  -v                  = more information
  --version           = show version
```

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 would be able to be considered for compression.

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]. Note that we have set by default the entropy of 'N's to 2.0, however, it can be changed to a value of interest using "-e" option.

Creating multiple finite-context models and substitutional-tolerant Markov models can be done in a multi-threaded fashion by setting "-n" to an integer.

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. Size of the window and type of the windowing function can be set by "-f" and "-ft" options, respectively. Besides Hann window that is used by default to smooth the information content (profile), we have implemented several other windowing functions, including Blackman [3], Hamming [4], Nuttall [5], rectangular [6], sine [7], triangular [8] and Welch [9] windows. These functions are given by

$$w[n] = 1, \hspace{6cm} \text{(rectangular)}$$
$$w[n] = 1 - \left| \frac{n - N/2}{L/2} \right|, \quad L = N, \hspace{3.5cm} \text{(triangular/Bartlett)}$$
$$w[n] = 1 - \left(\frac{n - N/2}{N/2} \right)^2, \hspace{5cm} \text{(Welch)}$$
$$w[n] = \sin\left(\frac{\pi n}{N} \right), \hspace{6cm} \text{(sine)}$$
$$w[n] = 0.54348 - 0.45652 \, \cos\left(\frac{2\pi n}{N} \right), \hspace{3.5cm} \text{(Hamming)}$$
$$w[n] = 0.42659 - 0.49656 \, \cos\left(\frac{2\pi n}{N} \right) + 0.07685 \, \cos\left(\frac{4\pi n}{N} \right), \hspace{1.5cm} \text{(Blackman)}$$
$$w[n] = 0.35577 - 0.48740 \, \cos\left(\frac{2\pi n}{N} \right) + 0.14423 \, \cos\left(\frac{4\pi n}{N} \right) - 0.01260 \, \cos\left(\frac{6\pi n}{N} \right), \hspace{0.5cm} \text{(Nuttall)}$$
$$\textbf{(Eq. S1)}$$

and are plotted in Fig. S1. Scale of the filter can be set as S (small), M (medium) or L (large), using "-fs". Also, instead of considering the whole information content, 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 content, the average entropy of reference-based compression is used by default as the threshold, but the threshold can be altered by "-th" option.

Smash++ is capable of finding even small similar regions in two sequences. However, there are some corner cases that the size of similar regions in reference and target are not balanced. Smash++ handles these cases using "-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 Smash++ wants to create the model of each target region to use it for compression of the reference
    -dp

Triggering "-nr" makes the tool not to perform the reference-free compression, self-complexity computation.

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 "-sb" option. When the information profile is obtained, Smash++ smoothens then removes it by default. However, it can be preserved by "-sp" option. The same thing happens to the smoothed file, i.e., it is segmented then 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 (content), filtered and segmented files, by triggering "-sa" option.

For the purpose of compression, 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, respectively. Parameters of the models are described in detail in section **??**. Note that
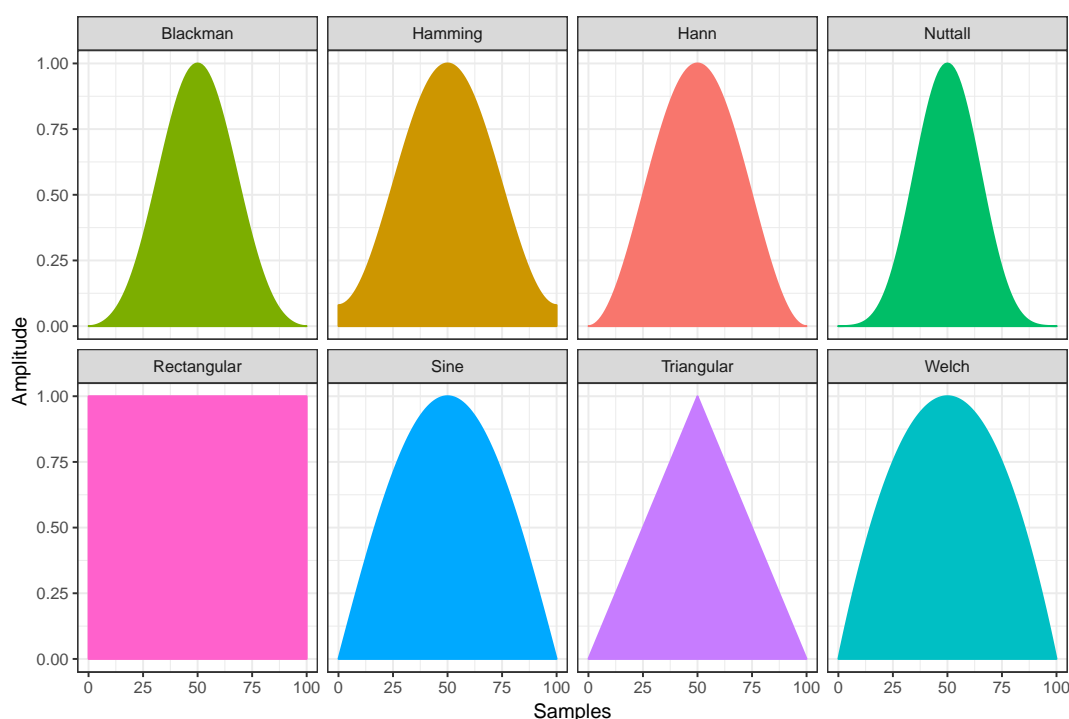
Fig. S1. Window functions.

using "-ll" option, the list of parameters that would be chosen for each model automatically, will be shown.

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. This file can be visualized by

```
1    ./smashpp -viz
```

which gives

```
1    SYNOPSIS
2      ./smashpp -viz [OPTIONS]  -o <SVG-FILE>  <POS-FILE>
3
4    OPTIONS
5      Required:
6      <POS-FILE>          = position file, generated by
7                            Smash++ tool (*.pos)
8
9      Optional:
10     -o  <SVG-FILE>      = output image name (*.svg)          -> map.svg
11     -rn <STRING>        = reference name shown on output. If it
12                           has space, use double quotes, e.g.
13                           "Seq label". Default: name in header
14                           of position file
15     -tn <STRING>        = target name shown on output
16     -l  <INT>           = type of the link between maps: [1, 6] -> 1
17     -c  <INT>           = color mode: [0, 1]                  -> 0
18     -p  <FLOAT>         = opacity: [0.0, 1.0]                 -> 0.9
19     -w  <INT>           = width of the sequence: [15, 100]    -> 16
20     -s  <INT>           = space between sequences: [15, 200]  -> 62
21     -f  <INT>           = multiplication factor for          -> 43
22                           color ID: [1, 255]
23     -b  <INT>           = beginning of color ID: [0, 255]     -> 0
24     -rt <INT>           = reference tick: [1, 4294967295]
25     -tt <INT>           = target tick: [1, 4294967295]
26     -th [0][1]          = tick human readable: 0=false, 1=true -> 1
27     -m  <INT>           = minimum block size: [1, 4294967295] -> 1
```

```
28    -vv              = vertical view                      -> no
29    -nn              = do NOT show normalized relative     -> no
30                       compression (NRC)
31    -nr              = do NOT show self complexity         -> no
32    -ni              = do NOT show inverse maps            -> no
33    -ng              = do NOT show regular maps            -> no
34    -h               = usage guide
35    -v               = more information
36    --version        = show version
```

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.

S1.3   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++:

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

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

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

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

```
1    ./smashpp -r refs -t tars -w 45 -l 3
2    ./smashpp -viz refs.tars.pos
```
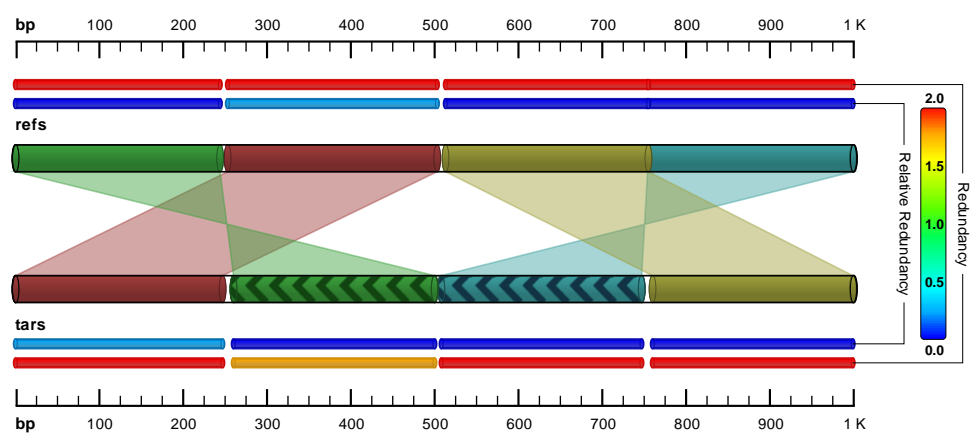
results in Fig. S2, which is saved as "map.svg".

Fig. S2. The result of running Smash++ on ...

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.