Note S1 Methods

The schema of the proposed method is illustrated in Fig. S1. Smash++ takes a reference and a target file, as inputs, and produces a position file, as output, which is then fed to the Smash++ visualizer to produce an SVG image. This process has eight major stages: (1) compression of the original target file based on the model of original reference file, (2) filtering and segmentation of the compressed file, (3) reference-free compression of the segmented files that are obtained by the previous stage, (4) compression of the original reference file based on the model of segmented files obtained by stage 2, (5) filtering and segmentation of the compressed files, (6) reference-free compression of the segmented files which are obtained by the stage 5, (7) aggregating positions based on the results of stages 3 and 6, and (8) visualizing the positions. The following sections describe the process in detail.

Fig. S1. The schema of Smash++.

S1.1 Data modeling

Smash++ works on the basis of cooperation between finite-context models (FCMs) and substitutional tolerant Markov models (STMMs), as illustrated in Fig. S2. Applying these models on various contexts, seen in the past, provides probability and weight values, which are then mixed to provide the final probability (P) of the next symbol, G in this case. The following subsections describe FCMs and STMMs in detail.

Fig. S2. model.

Finite-context model (FCM) A finite-context model considers Markov property to estimate the probability of the next symbol in an information source, based on the past k symbols (a context of size k) [1–3]. Denoting the context as $x_{i-k}^{i-1} = x_{i-k}x_{i-k+1}\cdots x_{i-2}x_{i-1}$, the probability of the next symbol s, which is posed at i, can be estimated as

$$P_m(s|x_{i-k}^{i-1}) = \frac{N(s|x_{i-k}^{i-1}) + \alpha}{N(x_{i-k}^{i-1}) + \alpha|\Theta|}, \qquad \textbf{(Eq. S1)}$$

in which m stands for model (FCM in this case), $N(s|x_{i-k}^{i-1})$ shows the number of times that the information source has generated symbol s in the past, $|\Theta|$ denotes size of the alphabet Θ, $N(x_{i-k}^{i-1}) = \sum_{j\in\Theta} N(j|x_{i-k}^{i-1})$ represents the total number of events occurred for the context x_{i-k}^{i-1} and α allows to keep a balance between the maximum likelihood estimator and the uniform distribution. Eq. S1 turns to the Laplace estimator, for $\alpha = 1$, and also behaves as a maximum likelihood estimator, for large number of events i [4].

Substitutional tolerant Markov model (STMM) A substitutional tolerant Markov model [5] is a probabilistic-algorithmic model that assumes at each position, the next symbol in the information source is the symbol which has had the highest probability of occurrence in the past. This way, an STMM ignores the real next symbol in the source. Denoting the past k symbols as $x_{i-k}^{i-1} = x_{i-k}x_{i-k+1}\cdots x_{i-2}x_{i-1}$, the probability of the next symbol s, at position i, can be estimated as

$$P_m(s|{x'}_{i-k}^{i-1}) = \frac{N(s|{x'}_{i-k}^{i-1}) + \alpha}{N({x'}_{i-k}^{i-1}) + \alpha|\Theta|}, \qquad \textbf{(Eq. S2)}$$

where N represents the number of occurrences of symbols, that is saved in memory, and ${x'}_{i-k}^{i-1}$ is a copy of the context x_{i-k}^{i-1} which is modified as

$$x'_i = \underset{\forall s \in \Theta}{\operatorname{argmax}} P_m(s|{x'}_{i-k}^{i-1}). \qquad \textbf{(Eq. S3)}$$

STMMs can be used along with FCMs to modify the behavior of Smash++ in confronting with nucleotide substitutions in genomic sequences. These models have the potential to be disabled, to reduce the number of mathematical calculations and consequently, increase the performance of the proposed method. Such operation is automatically performed using an array of size k (the context size), named history, which preserves the past k hits/misses. Seeing a symbol in the information source, the memory is checked for the symbol with the highest number of occurrences. If they are equal, a hit is saved in the history array; otherwise, a miss is inserted into the array. Before getting to store a hit/miss in the array, it is checked for the number of misses and in the case they are more than a predefined threshold t, the STMM will be disabled and also the history array will be reset. This process is performed for each symbol in the sequence.

This example shows the distinction between a finite-context model and a substitutional tolerant Markov model. Assume, the current context at position i, is c_0 = GGCTAACGTAC, and the number of occurrences of symbols saved in memory is A = 10, C = 12, G = 13 and T = 11. Also, the symbol to appear in the sequence is T. An FCM would consider the next context as c_1 = GCTAACGTACT, while an STMM would consider it as c'_1 = GCTAACGTACG, since the base G is the most probable symbol, based on the number of occurrences stored in memory.

Cooperation of FCMs and STMMs When FCMs and STMMs are in cooperation, the probability of the next symbol s, at position i, can be estimated as

$$P(s_i) = \sum_{m \in \mathcal{F}} P_m(s|x_{i-k}^{i-1})\, w_{m,i} + \sum_{m \in \mathcal{S}} P_m(s|{x'}_{i-k}^{i-1})\, w'_{m,i}, \qquad \textbf{(Eq. S4)}$$

in which $\mathcal{F}$ and $\mathcal{S}$ denote sets of FCMs and STMMs, respectively, $P_m(s|x_{i-k}^{i-1})$ shows the probability of the next symbol estimated by the FCM, $P_m(s|{x'}_{i-k}^{i-1})$ represents this probability estimated by the STMM, and $w_{m,i}$ and $w'_{m,i}$ are weights assigned to each model based on its performance. We have

$$\begin{aligned} \forall m \in \mathcal{F}: &\quad w_{m,i} \propto (w_{m,i-1})^{\gamma_m} P_m(s|x_{i-k-1}^{i-2}) \\ \forall m \in \mathcal{S}: &\quad w'_{m,i} \propto (w'_{m,i-1})^{\gamma'_m} P_m(s|{x'}_{i-k-1}^{i-2}), \end{aligned} \qquad \textbf{(Eq. S5)}$$

where γ_m and $\gamma'_m \in [0, 1)$ are forgetting factors predefined for each model. Also,

$$\sum_{m \in \mathcal{F}} w_{m,i} + \sum_{m \in \mathcal{S}} w'_{m,i} = 1. \qquad \textbf{(Eq. S6)}$$

By experimenting different forgetting factors for models, we have found that higher factors should be assigned to models that have higher context-order sizes (less complexity) and vice versa. As an example, when the context size $k = 6$, γ_m or $\gamma'_m \simeq 0.9$ and when $k = 18$, γ_m or $\gamma'_m \simeq 0.95$ would be appropriate choices. These values show that forgetting factor and complexity of a model are inversely related.

Storing models in memory The FCMs and STMMs include, in fact, count values which need to be saved in memory. For this purpose, four different data structures have been employed considering the context-order size k, as follows:

$$\text{data structure} = \begin{cases} \text{table of 64 bit counters,} & 1 \le k \le 11 \\ \text{table of 32 bit counters,} & k = 12, 13 \\ \text{table of 8 bit approximate counters,} & k = 14 \\ \text{Count-Min-Log sketch of 4 bit counters.} & k \ge 15 \end{cases}$$

Fig. S3. data structure.

The table of 64 bit counters, that is shown in Fig. S3a, simply saves number of events for each context. The table of 32 bit counters saves in each position the number of times that the associated context is observed. When a counter reaches to the maximum value $2^{32} - 1 = 4294967295$, all the counts will be renormalized by dividing by two, as shown in Fig. S3b.

The approximate counting is a method that employs probabilistic techniques to count large number of events, while using small amount of memory [6]. Fig. S4 shows the algorithm for two major functions associated with this method, UPDATE and QUERY. In order to update the counter, a pseudo-random number generator (PRNG) is used the number of times of the counter's current value to simulate flipping a coin. If it comes up 0/Heads each time or 1/Tails each time, the counter will be incremented. In Fig. S3c, the difference between arithmetic and approximate counting, and also the values which are actually stored in memory are shown. Note that since an approximate counter represents the actual count by an order of magnitude estimate, one only needs to save the exponent. For example, if the actual count is 8, we store it as $\log_2 8 = 3$ in memory.

The Count-Min-Log Sketch (CMLS) is a probabilistic data structure to save frequency of events in a table by means of a family of independent hash functions [7]. The algorithm for updating and querying the counter is shown in Fig. S5. In order to update the counter, its current value is hashed with d independent hash functions. Then, a coin is flipped the number of times of the counter's current value, employing a pseudo-random number generator. If it comes up 0/Heads each time or 1/Tails each time, the minimum hashed values (out of d values) will be updated, as shown in Fig. S3d.

The CMLS requires a family of pairwise independent hash functions $H = \{h : U \to [m]\}$, in which each function h maps some universe U to m bins. To have this family, we use universal hashing by randomly selecting a hash function from a universal family in which $\forall x, y \in U,\ x \neq y: \quad \Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{m}$. The hash function can be obtained by

$$h_{a,b}(x) = ((ax + b) \mod p) \mod m, \qquad \textbf{(Eq. S7)}$$

where $p \geq m$ is a prime number and a and b are randomly chosen integers modulo p with $a \neq 0$.

S1.2 Finding similar regions

In order to smooth the profile information, we use Hann window [8], which is a discrete window function given by

$$w[n] = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right) = \sin^2\left(\frac{\pi n}{N}\right), \qquad \textbf{(Eq. S8)}$$

function INCREASEDECISION(x)
    return True with probability $\frac{1}{2^x}$, else False
end function

function UPDATE(x)
    $c \leftarrow$ table[x]
    if INCREASEDECISION(c) = True **then**
        table[x] $\leftarrow c + 1$
    end if
end function

function QUERY(x)
    $c \leftarrow$ table[x]
    return $2^x - 1$
end function

Fig. S4. Approximate counting update and query.

```
Input: sketch width w, sketch depth d, m bins, prime
    p ≥ m, randomly chosen integers a_{1..d} and b_{1..d}
    modulo p with a ≠ 0

function HASH(k, x)                      ▷ Universal hash family
    return ((a_k x + b_k) mod p) mod m
end function

function MINCOUNT(x)
    minimum ← 15                         ▷ Biggest 4 bit number
    for k ← 1 to d do
        h ← HASH(k, x)
        if sketch[k][h] < minimum then
            minimum ← sketch[k][h]
        end if
    end for
    return minimum
end function

function INCREASEDECISION(x)
    return True with probability 1/2^x, else False
end function

function UPDATE(x)
    c ← MINCOUNT(x)
    if INCREASEDECISION(c) = True then
        for k ← 1 to d do
            h ← HASH(k, x)
            if sketch[k][h] = c then
                sketch[k][h] ← c + 1
            end if
        end for
    end if
end function

function QUERY(x)
    c ← MINCOUNT(x)
    return 2^c − 1
end function
```

Fig. S5. Count-Min-Log Sketch update and query.

in which, $0 \leq n \leq N$ and length of the window is $N+1$ (Fig. S6).

Fig. S6. Hann window for 101 samples.

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. S7), including Blackman [8], Hamming [9], Nuttall [10], rectangular [11], sine [12], triangular [13] and Welch [14] windows. These functions are given by

$$
\begin{aligned}
w[n] &= 1, && \text{(rectangular)}\\
w[n] &= 1 - \left|\tfrac{n-N/2}{L/2}\right|, \quad L = N, && \text{(triangular/Bartlett)}\\
w[n] &= 1 - \left(\tfrac{n-N/2}{N/2}\right)^2, && \text{(Welch)}\\
w[n] &= \sin\left(\tfrac{\pi n}{N}\right), && \text{(sine)}\\
w[n] &= 0.54348 - 0.45652\ \cos\left(\tfrac{2\pi n}{N}\right), && \text{(Hamming)}\\
w[n] &= 0.42659 - 0.49656\ \cos\left(\tfrac{2\pi n}{N}\right) + 0.07685\ \cos\left(\tfrac{4\pi n}{N}\right), && \text{(Blackman)}\\
w[n] &= 0.35577 - 0.48740\ \cos\left(\tfrac{2\pi n}{N}\right) + 0.14423\ \cos\left(\tfrac{4\pi n}{N}\right) - 0.01260\ \cos\left(\tfrac{6\pi n}{N}\right). && \text{(Nuttall)}
\end{aligned}
\tag{Eq. S9}
$$

Fig. S7. Window functions.

References

[1] K. Sayood, *Introduction to data compression.* Morgan Kaufmann, 2017.

[2] M. Hosseini, D. Pratas, and A. J. Pinho, "AC: A compression tool for amino acid sequences," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 11, no. 1, pp. 68–76, 2019.

[3] A. J. Pinho and D. Pratas, "MFCompress: a compression tool for FASTA and multi-FASTA data," *Bioinformatics*, vol. 30, no. 1, pp. 117–118, 2013.

[4] D. Pratas, R. M. Silva, A. J. Pinho, and P. J. Ferreira, "An alignment-free method to find and visualise rearrangements between pairs of DNA sequences," *Scientific reports*, vol. 5, p. 10203, 2015.

[5] D. Pratas, M. Hosseini, and A. J. Pinho, "Substitutional tolerant Markov models for relative compression of DNA sequences," in *International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB).* Springer, 2017, pp. 265–272.

[6] R. Morris, "Counting large numbers of events in small registers," *Commun. ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[7] G. Pitel and G. Fouquier, "Count-min-log sketch: Approximately counting with approximate counters," in *International Symposium on Web AlGorithms*, Deauville, France, Jun 2015.

[8] 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.

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

[10] 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.

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

[12] 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.

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

[14] 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.