

# Smash++: an alignment-free and memory-efficient tool to find genomic rearrangements

Morteza Hosseini<sup>1,\*</sup>, Diogo Pratas<sup>1,2</sup>,  
Burkhard Morgenstern<sup>3,4</sup>, Armando J. Pinho<sup>1</sup>

<sup>1</sup>IEETA/DETI, University of Aveiro, 3810-193 Aveiro, Portugal,

<sup>2</sup>Department of Virology, University of Helsinki, 00100 Helsinki, Finland,

<sup>3</sup>Department of Bioinformatics, University of Göttingen,  
Goldschmidtstr. 1, 37077 Göttingen, Germany,

<sup>4</sup>Göttingen Center of Molecular Biosciences (GZMB),  
Justus-von-Liebig-Weg 11, 37077 Göttingen, Germany

\*seyedmorteza@ua.pt

## Abstract

**Background:** The development of high-throughput sequencing technologies and, as its result, the production of huge volumes of genomic data, has accelerated biological and medical research and discovery. Study on genomic rearrangements is crucial due to their role in chromosomal evolution, genetic disorders and cancer; **Results:** We present Smash++, an alignment-free and memory-efficient tool to find and visualize small- and large-scale genomic rearrangements between two DNA sequences. This computational solution extracts information contents of the two sequences, exploiting a data compression technique, in order for finding rearrangements. We also present Smash++ visualizer, a tool that allows the visualization of the detected rearrangements along with their self- and relative complexity, by generating an SVG (Scalable Vector Graphics) image; **Conclusions:** Tested on several synthetic and real DNA sequences from bacteria, fungi, Aves and mammalia, the proposed tool was able to accurately find genomic rearrangements. The detected regions complied with previous studies which took alignment-based approaches or performed FISH (Fluorescence *in situ* hybridization) analysis. The maximum peak memory usage among all experiments was ~1 GB, which makes Smash++ feasible to run on present-day standard computers.

**keywords:** genomic rearrangement; alignment-free; genome comparison; genome duplication; data compression; information theory; probabilistic-algorithmic model; complexity; visualization; high-throughput sequencing

## 1 Background

With the ever-increasing development of high-throughput sequencing (HTS) technologies, a massive amount of genomic information is produced at much higher speed and lower cost than was possible before [1]. Analyses of such information has led to the advancement of our understanding of biology and disease, over the past decade [2, 3]. Computational solutions play a key role in dry-lab analysis of the deluge of HTS data by using efficient and fast algorithms.

Genome rearrangements are mutations that alter the arrangement of genes on a genome, and usually occur in the presence of errors in cell division following meiosis or mitosis. These structural abnormalities in chromosomes include, but are not limited to, deletions, duplications, translocations, inversions and insertions, mostly occur as an accident in the sperm or egg cell and hence are present in every cell of the body [4, 5].

Studies on chromosomal aberrations, which underlie many genetic diseases and cancer, are crucial for diagnostics, prognostics and targeted therapeutics [6, 7]. Examples of such diseases are the Wolf–Hirschhorn syndrome (WHS), that is caused by a partial deletion from human chromosome location 4p16.3 [8], the Charcot–Marie–Tooth disease (CMT), that is most commonly caused by duplication of the gene encoding peripheral myelin protein 22 (PMP22) on human chromosome 17 [9], and the acute myeloid leukemia (AML), that may be caused by translocations between human chromosome 8 and 21 [10].

In this paper, we present Smash++, an alignment-free tool that finds chromosomal rearrangements between two DNA sequences based on their information content, which is obtained by a data compression technique. This computational solution follows a combination of probabilistic and algorithmic approaches for having a quantitative definition of information, although it can be seen as more of a probabilistic one [11]. Associated with Smash++, we present a visualizer that is capable of visualizing as SVG images informationally similar regions between two genomic sequences. This tool also provides self- and relative redundancy (complexity) for the similar regions.

Smash++ is an improved version of Smash [12], featuring (1) improved accuracy, obtained by using multiple finite-context models along with substitution-tolerant Markov models to find fine-grained and coarse-grained chromosomal rearrangements, (2) presenting self-complexity (redundancy) and relative redundancy of informationally similar regions between two DNA sequences, (3) improved user interface (UI) in command line, by adding several options to customize the tool for running, and resulting SVG image, by adding markers for positions of DNA bases and also plotting self- and relative redundancy, and (4) improved performance, in terms of memory usage.

## 2 Results and Discussion

### 2.1 Implementation

Smash++ is implemented in the C++ language and is licensed under GNU GPLv3. It generates information maps for two sequences and, based upon that, finds similar regions in them, in which there can be potentially DNA rearrangements. Therefore, Smash++ gives an insight into positions of rearrangements that have happened between two sequences. The tool comes with a visualizer, that can be called in the command line with a flag called “-viz”. Similar regions in reference and target sequences are shown with the same color, that are chosen randomly using HSV color model. For more information about usage of the tool, see note S3 of the supplementary material.

The machine used for the tests had a 4-core 3.40 GHz Intel<sup>®</sup> Core™ i7-6700 CPU and 32 GB of RAM. The Python script “xp.py”, in the “experiment” directory, can be used to reproduce the results by switching False/True the variables associated with each dataset.

### 2.2 Dataset

Smash++ and several other methods have been tested on a collection of synthetic and real sequences, that are described in Table 1. We used the GOOSE toolkit (<https://github.com/pratas/goose>) to make the synthetic sequences of which the sizes vary from 1.5 Kb to 100 Mb. We applied mutations and reversely complemented parts of the sequences. For a real dataset, we chose different sequences from bacteria, Aves, mammalia and fungi, with the sizes of ~1 Mb to ~127 Mb.

## 2.3 Application on synthetic data

Figure 1 illustrates the result of running Smash++ and the associated visualizer on a synthetic dataset. The top sections show how we have built the reference and the target sequences. For example, to build the reference sequence in Figure 1a, we generated three random sequences of size 500 b, using GOOSE, and concatenated them. For building the target sequence, we made reverse complements of parts I and III from the reference, and also mutated part II 2%, then we concatenated the parts in the order shown in the figure. Figures 1b, c and d follow the same procedure. To build the target in Figure 1e, we mutated the first 1 Kb block of the reference 1%, the second block 2%, the third block 3%, up until the 60th block that we mutated 60%.

The bottom sections of Figure 1 show the output of the Smash++ visualizer, detecting similar regions regardless of their sizes. Note that for each detected region, the average value of redundancy and relative redundancy is illustrated. In Figure 1b, part II of the reference is mutated 90%, i.e., nine out of every ten bases is mutated, on average. As expected, Smash++ does not recognize similarity between this pair of regions. Also, in the case of parts III and IV of the reference, since we detect similarity between part III of the reference and I of the target, and also part IV of the reference and II of the target, and there is no space between these regions, we join them and consider them as a bigger region of size 50 Kb. Figure 1e shows that Smash++ is able to detect ~43% of mutation, which has been made possible by the usage of substitution-tolerant Markov models (see section “Methods”). Figure 1 shows that Smash++ can be employed to detect small-scale and large-scale similarities between DNA sequences.

## 2.4 Application on real data

Figure 2 shows similarities between real sequences, found by Smash++. Subfigures a and b show similarities of chromosomes 18 and 14 of *Gallus gallus* (chicken) with orthologous chromosomes 20 and 16 of *Meleagris gallopavo* (turkey), respectively. These avian species, that are of great agricultural and commercial importance, are estimated to have diverged at 37.2 MYA [13]. Figures 2a and b demonstrate that Smash++ was able to find the inversions confirmed by FISH analysis, reported at [14, 15].

In Figures S1 and S2 of the supplementary material, we have compared Smash++ with other methods, on GGA 18 / MGA 20 and GGA 14 / MGA 16 chromosomes, respectively. The methods included in these figures are: (a) Smash++; (b) progressiveMauve [16], that uses an alignment objective score to detect rearrangement breakpoints when genomes have unequal gene content. It also applies a probabilistic alignment filtering method in order for removing erroneous alignments of unrelated sequences; (c) the method proposed in [15], that takes a bacterial artificial chromosome (BAC)-based approach along with FISH analysis to develop an integrated physical, genetic and comparative map of chicken and turkey; (d) SynBrowser [17], that constructs synteny blocks using prebuilt alignments in the UCSC genome browser database; and (e) FISH analysis [14].

Figure 2c demonstrates similarities between chromosomes 12 of *Homo sapiens* and *Pan troglodytes*, that are estimated to have diverged at 6.7 MYA. A comparison to other methods is provided in Figures S3 of the supplementary material. The methods include: (a) Smash++; (b) progressiveMauve; (c) Cinteny [18], that performs sensitivity analysis for synteny block detection and for the subsequent computation of reversal distances, by means of an extended version of ternary search trees (TST). Embedded in this extension are “walks” through the leaves of the tree, that correspond to walks on the genome markers in their linear order; (d) SynBrowser; and (e) D-Genies [19], that works based on alignment of genomes by minimap2 software package [20].

Figure 2d illustrates similarities between *Xanthomonas oryzae* pv. *oryzae* PXO99A and *Xanthomonas oryzae* pv. *oryzae* MAFF 311018, two strains of *Xanthomonas oryzae* pv. *oryzae* (Xoo) pathogen, which causes the disease of bacterial blight of rice (*Oryza sativa* L.). That is the most serious bacterial disease

of rice that can reduce yields by as much as 50% [21]. Note that to have a clearer picture, we have not plotted the shades connecting similar regions. This can be achieved by “-l 6” option while calling the Smash++ visualizer. Figure S4 of the supplementary material provides the comparison of Smash++ with progressiveMauve and the study [21], which uses an alignment method to find genome rearrangements in Xoo. As can be seen, the result provided by Smash++ conforms to the one presented in the study [21], without performing an alignment.

## 2.5 Comparison to Smash

To have a better understanding of the improvement we have made over the first version, Smash, we compare the two tools on a synthetic and a real dataset (see Figure 3). In Table 1, the procedure of making the synthetic data (RefComp and TarComp) is described. Figure 3a shows the comparison of running Smash and Smash++ on the synthetic dataset. For Smash, we used an FCM with  $k$ -mer size of 14, and for Smash++, we used a cooperation of an FCM with  $k$ -mer size of 14 and an STMM with number of substitutions of 5. As the information profiles show, Smash++ is able to model better the data, since it uses less information (lower information contents) to describe the target based on the reference; this is possible because of employing a cooperation of the FCM and the STMM instead of using solely an FCM. We expect the output to have the following format: parts I, II, III and IV of the reference and the target are similar (including rearrangements), there is also inverted repeats between parts V, VI and VII of the sequences, and finally, there are rearrangements between parts VIII, IX and X of the sequences. When there is no space between consecutive regions, Smash++ joins them; therefore, we expect Smash++ to detect three similar regions: the one including parts I, II, III and IV, the one with parts V, VI and VII, and the one including parts VIII, IX and X. The rearrangements map shows that Smash++ fulfills our expectation. On the other side, Smash was not able to detect all rearrangements, showing that to model such dataset, we need more than a single FCM.

The result of running Smash and Smash++ on a real dataset, *Saccharomyces cerevisiae* chromosome VII and *Saccharomyces paradoxus* chromosome VII, is demonstrated in Figure 3b. *S. cerevisiae* is a species of yeast that plays a key role in winemaking, baking and brewing. It has been a eukaryotic model organism that gives insights into molecular functioning of human cells [22]. *S. paradoxus* is closest known species to the *S. cerevisiae*, that has proved its importance on different fields of the life sciences, including evolution, ecology and biotechnology [23]. For the experiment, we ran Smash using an FCM with  $k$ -mer size of 14, and Smash++ using an FCM with  $k$ -mer size of 14 cooperated with an STMM with number of substitutions of 5. As can be seen, using an FCM along with an STMM could drastically improve modeling the data, which led to find rearrangements more accurately. The rearrangements map of Smash++ conforms to the previous study [22].

## 2.6 Robustness against fragmented data

Inherited from Smash, Smash++ is capable of finding similarities between a fragmented reference and a target sequence. Figure S5 of the supplementary material shows robustness of the proposed tool against fragmented data, for different randomly permuted block sizes. As can be seen, the same three target regions are detected even when the reference is fragmented to 100,000 blocks of 30 bases. This capability might be of interest in case of non-assembled sequences or in presence of assembly errors; note that this approach can not be considered as an alternative to assembly.

## 2.7 Benchmarking

Figure 4 illustrates performance of the proposed tool in terms of memory and time usage for all datasets (for more details, see supplementary Table S1). Size of the datasets are mentioned on top of each circle, in black, and number of detected similar regions between each pair of sequences is mentioned inside the circles, in white. The legend shows the precise size of datasets in bases (b). Figure 4a shows the peak memory in gigabytes used by Smash++ on all synthetic and real datasets. The maximum peak memory usage,  $\sim 1.08$  GB, was when the proposed tool was ran on human and chimpanzee chromosomes 12, that are the biggest datasets with the total size of  $\sim 268$  Mb. It should be mentioned that the memory usage of Smash++ is related to the  $k$ -mer size that is used for modeling the data, since different data structures are employed for different  $k$ -mer sizes (see section “Methods”). The sizes 13 and 14 were used to perform the experiments. The maximum memory usage of  $\sim 1$  GB enables Smash++ to run on any computer, nowadays.

Figure 4b demonstrates elapsed (wall clock) times, in minutes. The elapsed times rely on the file sizes along with the number of detected similar regions, meaning that the more the number of regions is and/or the greater the dataset size is, the more the time will be taken. Note that it is not a linear relation. As an example, the pair datasets “Large” and the pair “PXO99A\_MAFF311018” have approximately the same total size of 10 Mb. In the former case that two similarities is detected, Smash++ takes  $\sim 26$  seconds, but in the latter case with 23 similarities, the proposed tool takes  $\sim 1.8$  minutes to run. As another example, carrying out Smash++ on the pair “XLarge” with the total size of 200 Mb and four similarities detected, takes approximately the same wall clock time as carrying out on the pair “HS12\_PT12” that has the total size of  $\sim 268$  Mb and three similar regions are detected. Regarding the pair “Perm30” with 11,607 similarities detected, we should notice that it has a massively fragmented reference sequence with 10,000 fragments of 30 b, therefore it is by far the most time-consuming dataset. Note that the difference between the values of 10,000 (number of reference fragments) and 11,607 (number of similar regions) arises from the fact that a number of the reference chunks are similar to more than one target region and vice versa. It is worth mentioning that due to the absence of a tool that provides relative and self-complexity in addition to detecting similarities, we cannot have a fair comparison to other tools in terms of time and memory usage; therefore, we have only provided the performance results for the proposed tool.

## 3 Conclusions

Finding genomic rearrangements is crucial, since they play an important role in genetic disorders, cancer and chromosomal evolution. We presented Smash++, an alignment-free tool that accurately finds small- and large-scale genomic rearrangements between pairs of DNA sequences, by employing a data compression approach. This memory-efficient tool was successfully tested on several synthetic and real data from bacteria, fungi, Aves and mammalia. The presented results showed that the detected rearrangements complied with previous studies which used alignment-based methods or performed FISH analysis. Smash++ consumed a maximum of  $\sim 1$  GB of memory, among all experiments, which showed that it can be run on any computer, nowadays. The proposed tool has the potential to improve accuracy of diagnostic and genetic counselling, and also to guide future investigations into development of personalized therapeutic.

## 4 Methods

The schema of the proposed method is illustrated in Figure 5. Smash++ takes as inputs a reference and a target sequence and produces as output a position file, including local similarities of the two sequences, which can then be used by the Smash++ visualizer to produce an SVG image illustrating the similarities. This process has eight major stages: (1) compression of the original target file, based on the model of the original reference file, (2) filtering the information profile, which is the output of stage 1, and segmenting the target sequence, (3) reference-free compression of the segmented sequences, obtained by the previous stage, (4) compression of the original reference file, based on the model of segmented sequences, which are obtained by stage 2, (5) filtering the information profile and segmenting the reference sequence, (6) reference-free compression of the segmented sequences, (7) aggregating positions, that are generated by stages 3 and 6, and (8) visualizing the positions. The following sections describe the process in detail.

### 4.1 Data modeling

We consider sequences over the nucleotide alphabet  $\Theta = \{A, C, G, T\}$ ; our goal is to measure the degree of local similarity between two such sequences. More specifically, we consider a reference sequence  $S = s_1, \dots, s_N$  over  $\Theta$ , and we want to measure the local *information content* of a target sequence, given this reference sequence. To this end, we employ a combination of finite-context models and substitution-tolerant Markov models to derive different probabilistic measures for observing a nucleotide  $x$  in a sequence, given the *context* of the previous  $k$  nucleotides (Fig. 6a); these probabilities are then mixed (by multiplications and additions shown in Fig. 6b) to provide the final probability ( $P$ ) of observing the nucleotide  $x$ . The following subsections describe the models we use in detail.

#### Finite-context model (FCM)

We consider the probability of observing a certain nucleotide, given the previous  $k$  nucleotides, by using the relative frequency of this event in the reference sequence  $S$ . For  $x \in \Theta$  and a  $k$ -mer  $Q \in \Theta^k$ , let  $N(x|Q)$  be the number of occurrences of  $Q$  in  $S$  that are followed by nucleotide  $x$ , and let  $N(Q)$  be the number of occurrences of  $Q$  in  $S$ . As in [11, 24, 25], we then define

$$P_{\text{FCM}}(x|Q) = \frac{N(x|Q) + \alpha}{N(Q) + 4 \cdot \alpha}, \quad (1)$$

where “4” is the size of alphabet  $\Theta$  and  $\alpha$  is a pseudo-count parameter. For  $\alpha = 1$ , Eq. 1 turns into the *Laplace estimator*. Note that an FCM has the Markov property, in which the conditional probability distribution of observing a nucleotide depends only upon the state of preceding  $k$ -mer.

#### Substitution-tolerant Markov model (STMM)

Given the reference sequence  $S$ , we use the aforementioned probability distribution  $P_{\text{FCM}}$  to define a sequence  $S' = s'_{-k}, s'_{-k+1}, \dots, s'_N$  recursively by

$$s'_i = \begin{cases} A & \text{if } i < 1 \\ \arg \max_{x \in \Theta} P_{\text{FCM}}(x|s'_{i-k}, \dots, s'_{i-1}) & \text{if } i \geq 1. \end{cases}$$

For  $x \in \Theta$  and a  $k$ -mer  $Q \in \Theta^k$ , we then define  $N'(x|Q)$  as the number of occurrences of  $Q$  followed by  $x$  and  $N'(Q)$  as the number of occurrences of  $Q$ , respectively, in the sequence  $S'$ . Finally, we define

$$P_{\text{STMM}}(x|Q) = \frac{N'(x|Q) + \alpha}{N'(Q) + 4 \cdot \alpha}. \quad (2)$$

STMMs, that are probabilistic-algorithmic models [26, 11], can be used along with FCMs to modify the behavior of Smash++ when confronted with nucleotide substitutions in genomic sequences. These models can 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. Observing a symbol in the sequence, 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 nucleotide in the sequence.

The following example shows the distinction between an FCM and an STMM. Assume that the current context at a certain position is AGACGTAC, and the number of occurrences of symbols saved in memory is 10, 6, 15 and 8 for A, C, G and Ts, respectively; also, the symbol to appear in the sequence is T. An FCM considers the next context as GACGTACT, while an STMM considers it as GACGTACG, since the nucleotide 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 observing a nucleotide  $x$  in a sequence  $S$  can be estimated as

$$P(x) = \sum_{1 \leq i \leq m} P_{\text{FCM}_i}(x|Q) w_i + \sum_{0 \leq j < n} P_{\text{STMM}_j}(x|Q) w_j, \quad \forall x \in S, \quad (3)$$

in which  $m$  and  $n$  denote the number of FCMs and STMMs, respectively, and  $w_i$  and  $w_j$  are weights assigned to each FCM and STMM, respectively, based on its performance. We have

$$\begin{aligned} w_{i_p} &\propto (w_{i_{p-1}})^{\gamma_i} P_{\text{FCM}}(x|Q_{p-1}), & 1 \leq i \leq m, \\ w_{j_p} &\propto (w_{j_{p-1}})^{\gamma_j} P_{\text{STMM}}(x|Q_{p-1}), & 0 \leq j < n, \end{aligned} \quad (4)$$

where  $p$  denotes a certain position, and  $\gamma_i$  and  $\gamma_j \in [0, 1)$  are forgetting factors predefined for each model. Also,

$$\sum_{1 \leq i \leq m} w_i + \sum_{0 \leq j < n} w_j = 1. \quad (5)$$

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  $k = 6$ ,  $\gamma_i$  or  $\gamma_j \simeq 0.9$  and when  $k = 18$ ,  $\gamma_i$  or  $\gamma_j \simeq 0.95$  would be appropriate choices. These values show that forgetting factor and complexity of a model are inversely related.

## 4.2 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:

- table of 64 bit counters, for  $1 \leq k \leq 11$ ,
- table of 32 bit counters, for  $k = 12, 13$ ,
- table of 8 bit approximate counters, for  $k = 14$ , and
- Count-Min-Log sketch of 4 bit counters, for  $k \geq 15$ .

The table of 64 bit counters, that is shown in Figure 7a, simply saves the 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 the maximum value  $2^{32} - 1 = 4294967295$ , all the counts will be renormalized by dividing by two, as shown in Figure 7b.

Approximate counting is a method that employs probabilistic techniques to count large number of events, while using a small amount of memory [27]. Figure 8 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. Figure 7c shows the difference between arithmetic and approximate counting, and also the values which are actually stored in memory. 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 in memory  $\log_2 8 = 3$ .

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 [28]. The algorithm for updating and querying the counter is shown in Figure 9. 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 Figure 7d.

CMLS requires a family of pairwise independent hash functions  $H = \{h : U \rightarrow [m]\}$ , in which each function  $h$  maps some universe  $U$  to  $m$  bins. In this family of functions, the probability that all  $x, y \in U$ ,  $x \neq y$  will hash to any pair of hashed values  $z_1, z_2$  is as if they were perfectly random, i.e.,  $P_{h \in H}[h(x) = z_1 \wedge h(y) = z_2] = 1/m^2$ . A hash function in this family can be obtained by

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m, \quad (6)$$

where  $p \geq m$  is a prime number and  $a$  and  $b$  are randomly chosen integers modulo  $p$  with  $a \neq 0$ . Note that if the number of bins is a power of two,  $m = 2^M$ , multiply-add-shift scheme [29] can be used to avoid modular arithmetic. A hash function in this scheme can be obtained by:

$$h_{a,b}(x) = ((ax + b) \bmod 2^w) \operatorname{div} 2^{w-M}, \quad (7)$$

in which  $w$  is the number of bits in a machine word, e.g., 64,  $a$  is a random positive integer less than  $2^w$  and  $b$  is a random non-negative integer less than  $2^{w-M}$ . Such hash function can be implemented in the C++ language by

$$h_{a,b}(x) = (\text{uint64\_t}) (\mathbf{a} * \mathbf{x} + \mathbf{b}) \gg (\mathbf{w} - \mathbf{M}).$$



### 4.3 Finding similar regions

To find similar regions in reference and target sequences, a quantity is required for measuring the similarity. We use “per symbol information content”, in bpb (bit per base), which can be calculated as

$$I(x) = -\log_2 P(x), \quad \forall x \in S, \quad (8)$$

where  $P(x)$  denotes the probability of observing a nucleotide  $x$  in the sequence  $S$ , obtained by Equation 3.

The information content is the amount of information required to represent a symbol in the target sequence, based on the model of the reference sequence. The less the value of this measure is for two regions, the more amount of information is shared between them, and, therefore, the more similar are the two regions. Note that a version of this measure has been introduced in [12], which employs a single FCM to calculate the probabilities. In this paper, however, we exploit a cooperation between multiple FCMs and STMMs for highly accurate calculation of such probabilities.

The procedure of finding similar regions in a reference and a target sequence, illustrated in Figure 10, is as follows: after creating the model of the reference, the target is compressed based on that model and the information content is calculated for each symbol in the target. Then, the content of the whole target sequence is smoothed by a Hann window [30], which is a discrete window function given by  $w[n] = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right)$ , where  $0 \leq n \leq N$  and length of the window is  $N + 1$ . Next, the smoothed information content is segmented considering a predefined threshold, meaning that the regions with the content greater than the threshold are filtered out. This is carried out for both regular and inverted repeat homologies and, at the end, the result would be the regions in the target sequence that are similar to the reference sequence (Figure 10a). The described phase repeats for all of the target regions found, in the way that after creating the model for each region, the whole reference sequence is compressed to find those regions in the reference that are similar to each of the target regions (Figure 10b). The final result would have the form of Figure 10c.

### 4.4 Computing complexity

After finding the similar regions in reference and target sequences, we evaluate redundancy in each region, knowing that it is inversely related to Kolmogorov complexity, i.e., the more complex a sequence is, the less redundant it will be [31]. The Kolmogorov complexity,  $K$ , of a binary string  $s$ , of finite length, is the length of the smallest binary program  $p$  that computes  $s$  in a universal Turing machine and halts. In other words,  $K(s) = |p|$  is the minimum number of bits required to computationally retrieve the string  $s$  [32, 33].

The Kolmogorov complexity is not computable, hence, an alternative is required to compute it approximately. It has been shown in the literature that a compression algorithm can be employed for this purpose [34, 35, 36]. In this paper, we employ a reference-free compressor to approximate the complexity and, consequently, the redundancy of the found similar regions in the reference and the target sequences. This compressor works based on cooperation of FCMs and STMMs, which has been previously described in detail. Note that the difference between reference-based and reference-free version of such compressor is that, in the former mode, a model is first created for the reference sequence and, then, the target sequence is compressed based on that model, while in the latter mode, the model is progressively created at the time of compressing the target sequence.

## Availability of source code and requirements

- Project name: Smash++

- Project home page: <https://github.com/smortezah/smashpp>
- Operating system(s): Linux, macOS, Windows
- Programming language: C++, Python
- Other requirements: C++ 14, Python 3
- License: GNU GPLv3

## Availability of supporting data and materials

The data sets supporting the results of this article are available in the Smash++ Github repository, <https://github.com/smortezah/smashpp/experiment/dataset>.

## Additional files

A single Supplementary notes file, including:

**Supplementary Figure S1.** Comparison of Smash++ and other methods on *G. gallus* chromosome 18 and *M. gallopavo* chromosome 18.

**Supplementary Figure S2.** Different methods running on *G. gallus* chromosome 14 and *M. gallopavo* chromosome 16.

**Supplementary Figure S3.** Comparing with other methods on *H. sapiens* chromosome 12 and *P. troglodytes* chromosome 12.

**Supplementary Figure S4.** Result of running different methods on *X. oryzae* pv. *oryzae* PXO99A and *X. oryzae* pv. *oryzae* MAFF 311018.

**Supplementary Figure S5.** Similarity of a target sequence to a fragmented reference sequence, that is randomly permuted by different block sizes.

**Supplementary Table S1.** Performance of Smash++ running on all synthetic and real datasets.

**Supplementary Note S1.** Software manual for Smash++.

## Declarations

### List of abbreviations

AML: acute myeloid leukemia; BAC: bacterial artificial chromosome; CMT: Charcot–Marie–Tooth; CMLS: Count-Min-Log Sketch; CPU: central processing unit; FCM: finite-context model; FISH: Fluorescence *in situ* hybridization; GB: gigabyte; GGA: *Gallus gallus*; GHz: gigahertz; HS: *Homo sapiens*; HSV: Hue, Saturation, Value; HTS: high-throughput sequencing; KB: kilobyte; MB: megabyte; MGA: *Meleagris gallopavo*; MYA: million years ago; NCBI: national center for biotechnology information; PMP22: peripheral myelin protein 22; PRNG: pseudo-random number generator; PT: *Pan troglodytes*; RAM: random access memory; Sc: *Saccharomyces cerevisiae*; Sp: *Saccharomyces paradoxus*; STMM: substitution-tolerant Markov model; SVG: Scalable Vector Graphics; TST: ternary search tree; UI: user interface; WHS: Wolf–Hirschhorn syndrome; UCSC: University of California, Santa Cruz.

### Ethical Approval

Not applicable.

## Consent for publication

Not applicable.

## Competing Interests

The authors declare that they have no competing interests.

## Funding

This work was supported by Programa Operacional Factores de Competitividade – COMPETE (FEDER); and by national funds through the Foundation for Science and Technology (FCT), in the context of the projects [UID/CEC/00127/2014, PTCD/EEL-SII/6608/2014] and the grants [PD/BD/113969/2015, UID/CEC/00127/2019].

## Author’s Contributions

M.H. developed the software and wrote the manuscript. D.P. and A.J.P. contributed to and tested the software. D.P., B.M. and A.J.P. provided guidance. All authors contributed to the manuscript.

## Acknowledgements

We would like to thank everyone who has contributed to the development of Smash++, through testing and feedback.

## References

- [1] J. Reuter, D. V. Spacek, and M. Snyder, “High-throughput sequencing technologies,” *Molecular Cell*, vol. 58, no. 4, pp. 586–597, 2015.
- [2] D. E. V. Villamor, T. Ho, M. Al Rwahnih, R. R. Martin, and I. E. Tzanetakis, “High throughput sequencing for plant virus detection and discovery,” *Phytopathology*, vol. 109, no. 5, pp. 716–725, 2019.
- [3] S. M. Rego and M. P. Snyder, “High throughput sequencing and assessing disease risk,” *Cold Spring Harbor perspectives in medicine*, vol. 9, no. 1, p. a026849, 2019.
- [4] T. Hartmann, M. Middendorf, and M. Bernt, “Genome rearrangement analysis: Cut and join genome rearrangements and gene cluster preserving approaches,” in *Comparative Genomics*. Springer, 2018, pp. 261–289.
- [5] R. Gardner, R. M. Gardner, and D. J. Amor, *Gardner and Sutherland’s Chromosome Abnormalities and Genetic Counseling*. Oxford University Press, 2018, no. 70.
- [6] A. Theisen and L. G. Shaffer, “Disorders caused by chromosome abnormalities,” *The application of clinical genetics*, vol. 3, p. 159, 2010.
- [7] J. Damas, D. C. Samuels, J. Carneiro, A. Amorim, and F. Pereira, “Mitochondrial DNA rearrangements in health and disease—a comprehensive study,” *Human mutation*, vol. 35, no. 1, pp. 1–14, 2014.

- [8] A. Dufke, J. Seidel, M. Schöning, M. Döbler-Neumann, C. Kelbova, T. Liehr, V. Beensen, C. Backsch, U. Klein-Vogler, and H. Enders, “Microdeletion 4p16.3 in three unrelated patients with Wolf-Hirschhorn syndrome,” *Cytogenetic and Genome Research*, vol. 91, no. 1-4, pp. 81–84, 2000.
- [9] V. Timmerman, E. Nelis, W. Van Hul, B. Nieuwenhuijsen, K. Chen, S. Wang, K. B. Othman, B. Cullen, R. J. Leach, C. Hanemann *et al.*, “The peripheral myelin protein gene PMP-22 is contained within the Charcot-Marie-Tooth disease type 1A duplication,” *Nature genetics*, vol. 1, no. 3, p. 171, 1992.
- [10] L. Huang, L. V. Abruzzo, J. R. Valbuena, L. J. Medeiros, and P. Lin, “Acute myeloid leukemia associated with variant t(8;21) detected by conventional cytogenetic and molecular studies: a report of four cases and review of the literature,” *American Journal of Clinical Pathology*, vol. 125, no. 2, pp. 267–272, 2006.
- [11] 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.
- [12] 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.
- [13] S. Kumar, G. Stecher, M. Suleski, and S. B. Hedges, “Timetree: a resource for timelines, timetrees, and divergence times,” *Molecular Biology and Evolution*, vol. 34, no. 7, pp. 1812–1819, 2017.
- [14] R. A. Dalloul, J. A. Long, A. V. Zimin, L. Aslam, K. Beal, L. A. Blomberg, P. Bouffard, D. W. Burt, O. Crasta, R. P. Crooijmans *et al.*, “Multi-platform next-generation sequencing of the domestic turkey (*Meleagris gallopavo*): genome assembly and analysis,” *PLoS Biology*, vol. 8, no. 9, p. e1000475, 2010.
- [15] Y. Zhang, X. Zhang, T. H. O’Hare, W. S. Payne, J. J. Dong, C. F. Scheuring, M. Zhang, J. J. Huang, M.-K. Lee, M. E. Delany *et al.*, “A comparative physical map reveals the pattern of chromosomal evolution between the turkey (*Meleagris gallopavo*) and chicken (*Gallus gallus*) genomes,” *BMC Genomics*, vol. 12, no. 1, p. 447, 2011.
- [16] A. E. Darling, B. Mau, and N. T. Perna, “progressivemauve: multiple genome alignment with gene gain, loss and rearrangement,” *PloS One*, vol. 5, no. 6, p. e11147, 2010.
- [17] J. Lee, W.-y. Hong, M. Cho, M. Sim, D. Lee, Y. Ko, and J. Kim, “Synteny portal: a web-based application portal for synteny block analysis,” *Nucleic Acids Research*, vol. 44, no. W1, pp. W35–W40, 2016.
- [18] A. U. Sinha and J. Meller, “Cinteny: flexible analysis and visualization of synteny and genome rearrangements in multiple organisms,” *BMC Bioinformatics*, vol. 8, no. 1, p. 82, 2007.
- [19] F. Cabanettes and C. Klopp, “D-genies: dot plot large genomes in an interactive, efficient and simple way,” *PeerJ*, vol. 6, p. e4958, 2018.
- [20] H. Li, “Minimap2: pairwise alignment for nucleotide sequences,” *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.
- [21] S. L. Salzberg, D. D. Sommer, M. C. Schatz, A. M. Phillippy, P. D. Rabinowicz, S. Tsuge, A. Furutani, H. Ochiai, A. L. Delcher, D. Kelley *et al.*, “Genome sequence and rapid evolution of the rice pathogen *Xanthomonas oryzae* pv. *oryzae* pxo99a,” *BMC Genomics*, vol. 9, no. 1, p. 204, 2008.

- [22] G. Fischer, E. P. Rocha, F. Brunet, M. Vergassola, and B. Dujon, “Highly variable rates of genome rearrangements between hemiascomycetous yeast lineages,” *PLoS Genetics*, vol. 2, no. 3, p. e32, 2006.
- [23] G. Charron, J.-B. Leducq, C. Bertin, A. K. Dubé, and C. R. Landry, “Exploring the northern limit of the distribution of *Saccharomyces cerevisiae* and *Saccharomyces paradoxus* in north america,” *FEMS yeast research*, vol. 14, no. 2, pp. 281–288, 2014.
- [24] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2017.
- [25] A. J. Pinho and D. Pratas, “MFCCompress: a compression tool for FASTA and multi-FASTA data,” *Bioinformatics*, vol. 30, no. 1, pp. 117–118, 2013.
- [26] 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.
- [27] R. Morris, “Counting large numbers of events in small registers,” *Commun. ACM*, vol. 21, no. 10, pp. 840–842, 1978.
- [28] G. Pitel and G. Fouquier, “Count-min-log sketch: Approximately counting with approximate counters,” in *International Symposium on Web Algorithms*, Deauville, France, Jun 2015.
- [29] P. Woelfel, “Efficient strongly universal and optimally universal hashing,” in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1999, pp. 262–272.
- [30] 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.
- [31] M. Hosseini, D. Pratas, and A. J. Pinho, “Cryfa: a secure encryption tool for genomic data,” *Bioinformatics*, vol. 35, no. 1, pp. 146–148, 2018.
- [32] A. Turing, “On computable numbers, with an application to the Entscheidungsproblem,” *Proceedings of the London Mathematical Society*, vol. 42, no. 2, pp. 230–265, 1936.
- [33] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications*, 3rd ed. Springer, 2009.
- [34] H. Zenil, F. Soler-Toscano, J.-P. Delahaye, and N. Gauvrit, “Two-dimensional Kolmogorov complexity and an empirical validation of the Coding theorem method by compressibility,” *PeerJ Computer Science*, vol. 1, p. e23, Sep. 2015.
- [35] R. Antão, A. Mota, and J. A. T. Machado, “Kolmogorov complexity as a data similarity metric: application in mitochondrial DNA,” *Nonlinear Dynamics*, vol. 93, no. 3, pp. 1059–1071, Aug 2018.
- [36] C. Faloutsos and V. Megalooikonomou, “On data mining, compression, and kolmogorov complexity,” *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 3–20, Aug 2007.
- [37] H. Ochiai, Y. Inoue, M. Takeya, A. Sasaki, and H. Kaku, “Genome sequence of *Xanthomonas oryzae* pv. *oryzae* suggests contribution of large numbers of effector genes and insertion sequences to its race diversity,” *Japan Agricultural Research Quarterly: JARQ*, vol. 39, no. 4, pp. 275–287, 2005.

Table 1: Synthetic and real dataset used in the experiments. The real dataset can be download from NCBI via accession number (access.) provided in the descriptions.

Sequence	Group	Length (b)	Description
GGA18	Aves	11,373,140	Access.: CM000110 – <i>Gallus gallus</i> chromosome 18.
MGA20	Aves	10,730,484	Access.: CM000981 – <i>Meleagris gallopavo</i> isolate NT-WF06-2002-E0010 breed Aviagen turkey brand Nicholas breeding stock chromosome 20.
GGA14	Aves	16,219,308	Access.: CM000106 – <i>Gallus gallus</i> chromosome 14.
MGA16	Aves	14,878,991	Access.: CM000977 – <i>Meleagris gallopavo</i> isolate NT-WF06-2002-E0010 breed Aviagen turkey brand Nicholas breeding stock chromosome 16.
HS12	Mammalia	133,275,309	Access.: NC_000012 – <i>Homo sapiens</i> chromosome 12, GRCh38.p13 Primary Assembly.
PT12	Mammalia	130,995,916	Access.: NC_036891 – <i>Pan troglodytes</i> isolate Yerkes chimp pedigree #C0471 (Clint) chromosome 12.
PXO99A	Bacteria	5,238,555	Access.: CP000967 – <i>Xanthomonas oryzae</i> pv. <i>oryzae</i> causes the major disease of bacterial blight of rice ( <i>Oryza sativa</i> L.). <i>X. oryzae</i> pv. <i>oryzae</i> PXO99A strain is virulent toward a large number of rice varieties representing diverse genetic sources of resistance [21].
MAFF 311018	Bacteria	4,940,217	Access.: AP008229 – <i>X. oryzae</i> pv. <i>oryzae</i> MAFF 311018 is a Japanese race 1 strain [37].
ScVII	Fungi	1,090,940	Access.: NC_001139 – <i>Saccharomyces cerevisiae</i> S288C chromosome VII.
SpVII	Fungi	1,105,967	Access.: CP020299 – <i>Saccharomyces paradoxus</i> strain UFRJ50816 chromosome VII.
RefS	Synthetic	1,500	It consists of three segments of 500 base size.
TarS	Synthetic	1,500	To build TarS, segment I is mutated 2%, II is inversely repeated and III is duplicated.
RefM	Synthetic	100,000	It has four segments of 25 kilobase size.
TarM	Synthetic	100,000	For building TarM, segment I of RefM (out of total four) is inversely repeated, II is mutated 90%, III is duplicated and IV is mutated 3%.
RefL	Synthetic	5,000,000	It includes two segments, 2,500,000 bases each.
TarL	Synthetic	5,000,000	Segment I is inversely repeated and II is mutated 2% for building TarL.
RefXL	Synthetic	100,000,000	It is made of four segments, 25,000,000 bases each.
TarXL	Synthetic	100,000,000	Segment I is mutated 1%, segments II and III are inversely repeated and segment IV is duplicated to make TarXL.
RefMut	Synthetic	60,000	It includes 60 segments of 1 kilobase size.
TarMut	Synthetic	60,000	To build TarMut, the first segment (I) is mutated 1%, the second segment is mutated 2%, the third one is mutated 3%, and so on.
RefComp	Synthetic	1,000,000	It consists of 10 segments of 100 kilobases.
TarComp	Synthetic	1,000,000	For building it, the first segment (I) of RefComp is duplicated, the second, third and fourth segments are mutated 1%, 2% and 3%, respectively. The segments V, VI and VII of RefComp are inversely repeated, then mutated 4%, 5% and 6%, respectively. Finally, the segments VIII, IX and X are mutated 7%, 8% and 9%, respectively.
RefPerm	Synthetic	3,000,000	It includes three segments of 1 megabase size. In addition to the original sequence, it is permuted, using GOOSE toolkit, by blocks of sizes 450 Kb, 30 Kb, 1 Kb and 30 b.
TarPerm	Synthetic	3,000,000	To build TarPerm, the first segment is mutated 1%, the second segment is inversely repeated and the third one is mutated 2%.

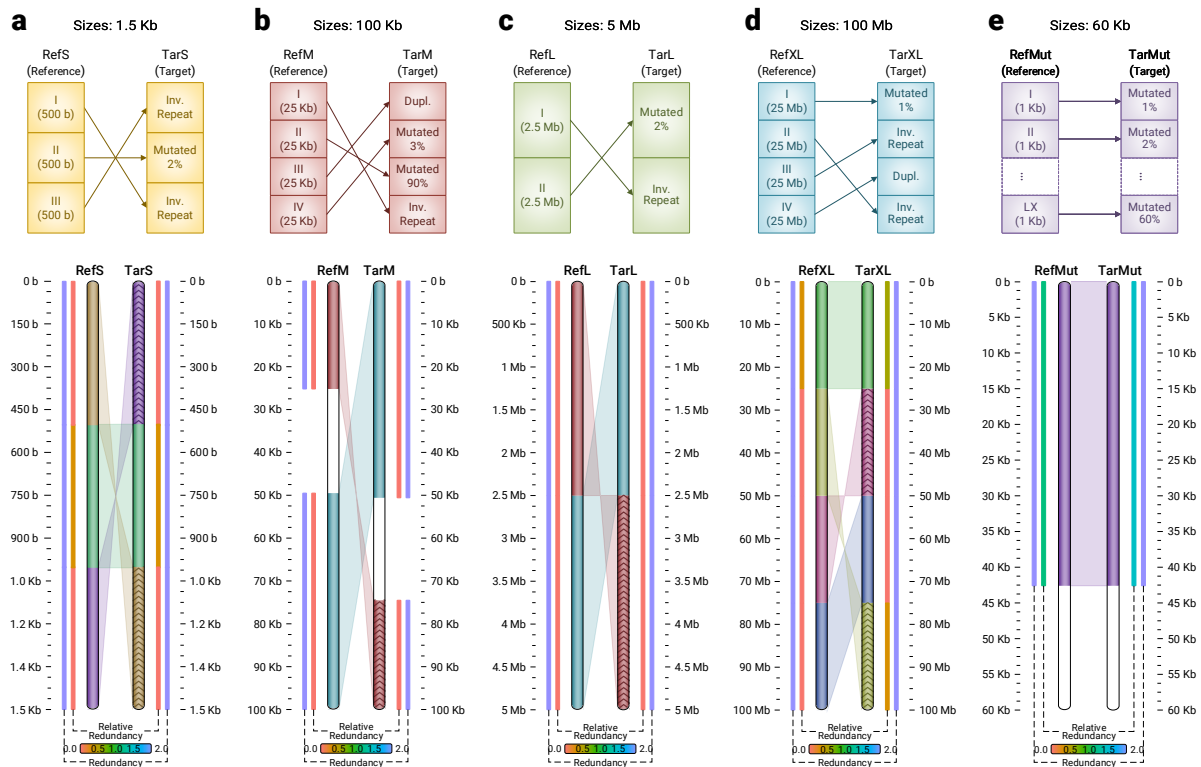


Figure 1: Similarities between synthetic sequences with different sizes, detected by Smash++. The parameters used are:  $k$ -mer size = 14 and number of substitutions in STMM = 5, which are the default parameters used by Smash++. For the threshold, the default value of 1.5 and 1.97 are used for subfigures a-d and e, respectively. (a) 1.5 Kb sequences; (b) 100 Kb sequences. No similarity is detected for part II of the reference, since it is mutated 90%. Parts III and IV of the reference and I and II of the target are joined, since there is no space between consecutive regions; (c) 5 Mb sequences; (d) 100 Mb sequences; (e) 60 Kb sequences. ~43% of mutation is detected.

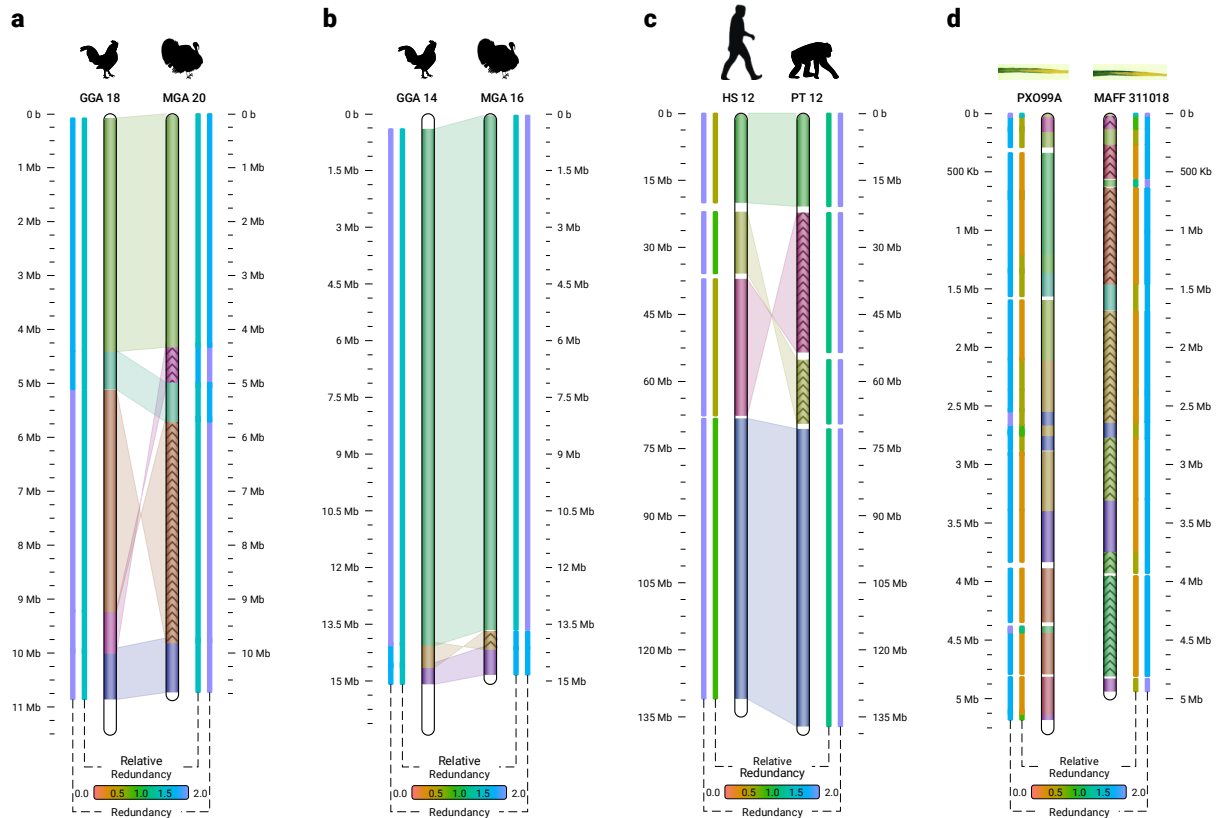


Figure 2: Similarities in a real dataset, detected by Smash++. (a) *G. gallus* (chicken) chr. 18 and *M. gallopavo* (turkey) chr. 20. The parameters were  $k$ -mer size = 14, no. substitutions in STMM = 5, threshold = 1.9 and min block size ( $m$ ) = 500,000, i.e., the regions smaller than 500,000 bases were not considered for further processing; (b) *G. gallus* chr. 14 and *M. gallopavo* chr. 16. The result is obtained by setting  $k = 14$ , no. substitutions = 5, threshold = 1.95 and  $m = 400,000$ ; (c) *H. sapiens* (human) chr. 12 and *P. troglodytes* (chimpanzee) chr. 12. The parameters were  $k = 14$ , without using STMM, threshold = 1.9 and  $m = 100,000$ ; (d) *X. oryzae* pv. *oryzae* PXO99A (a rice pathogen) and *X. oryzae* pv. *oryzae* MAFF 311018 (a rice pathogen). The result obtained by setting  $k = 13$ , threshold = 1.55 and  $m = 10,000$ .



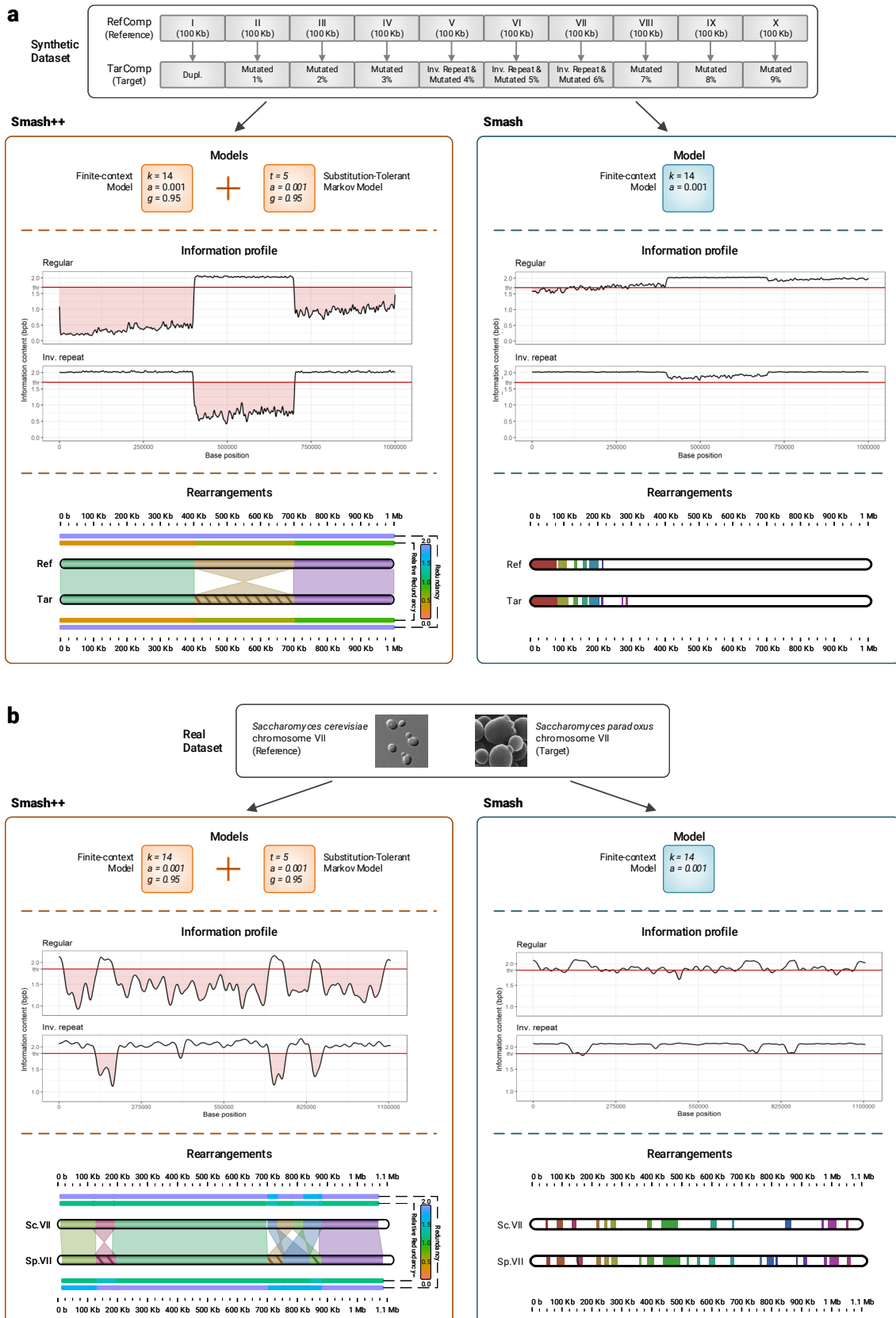


Figure 3: Comparison of Smash++ and Smash on (a) synthetic dataset. Using cooperation of an FCM and an STMM (in Smash++) produces more accurate results rather than using a single FCM (in Smash); and (b) real dataset, including *S. cerevisiae* chr. VII and *S. paradoxus* chr. VII. The rearrangements maps clearly show the improvement made over Smash, using an FCM along with an STMM.

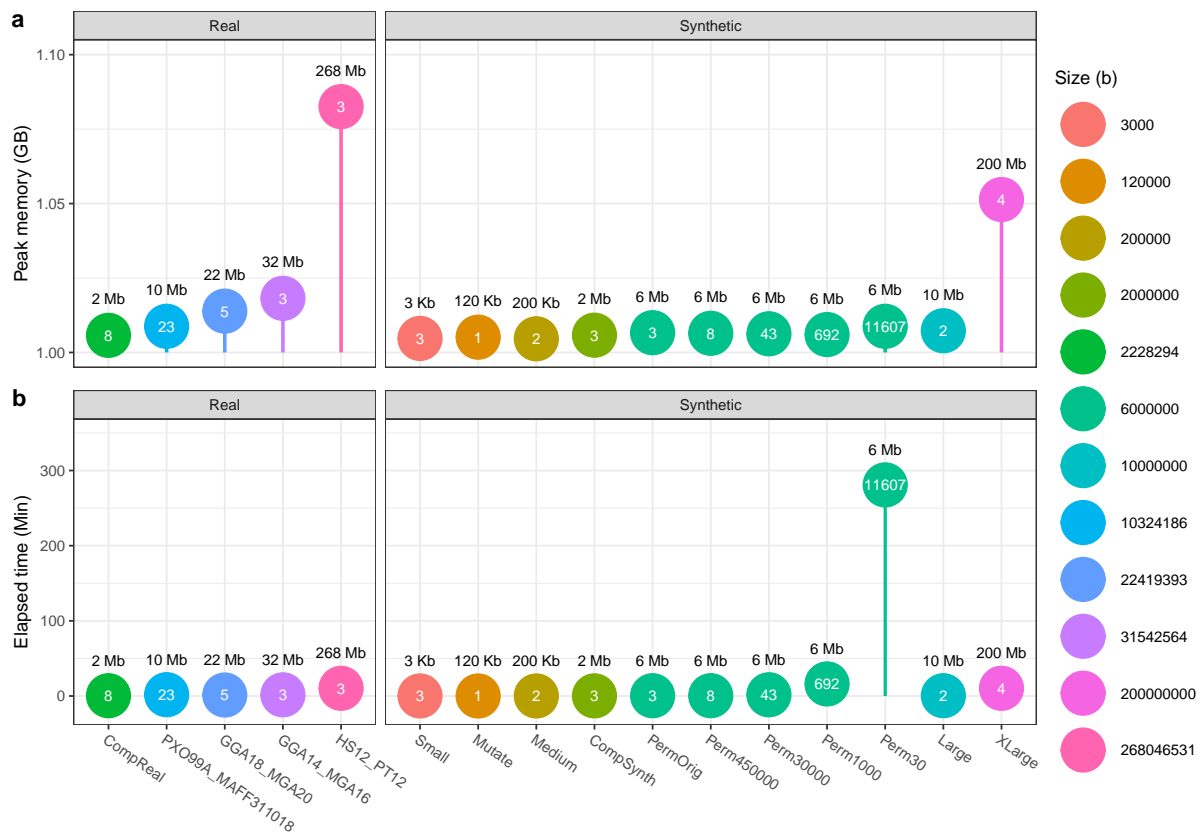


Figure 4: The peak memory consumption, in gigabytes, and elapsed (wall clock) time usage, in minutes, of Smash++ obtained by carrying out on all synthetic and real datasets described in Table 1.

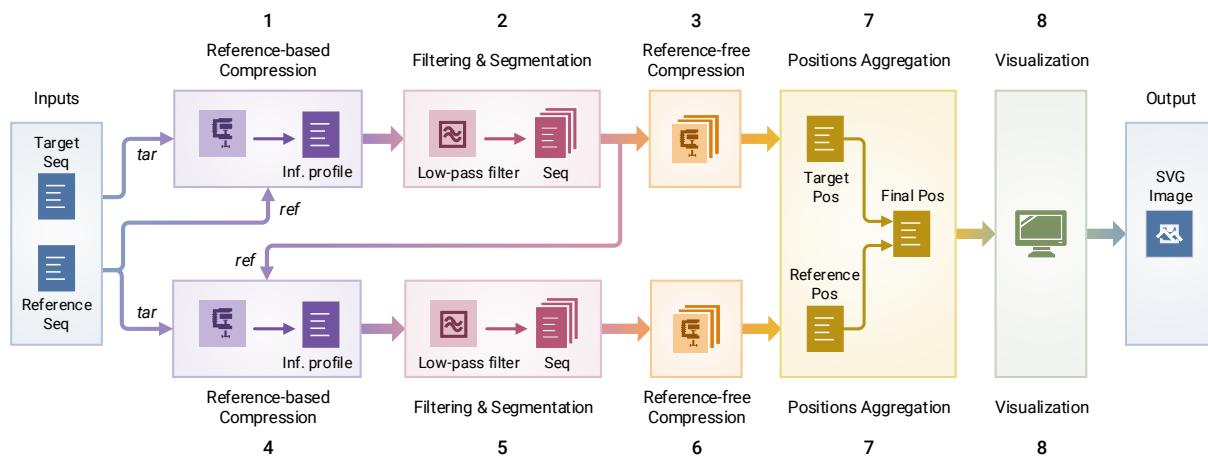


Figure 5: The schema of Smash++. The process of finding similar regions in reference and target sequences. Computing the redundancy in each region includes eight stages. Finally, Smash++ outputs a \*.pos file that includes the positions of the similar regions, and can be then visualized, resulting in an SVG image.

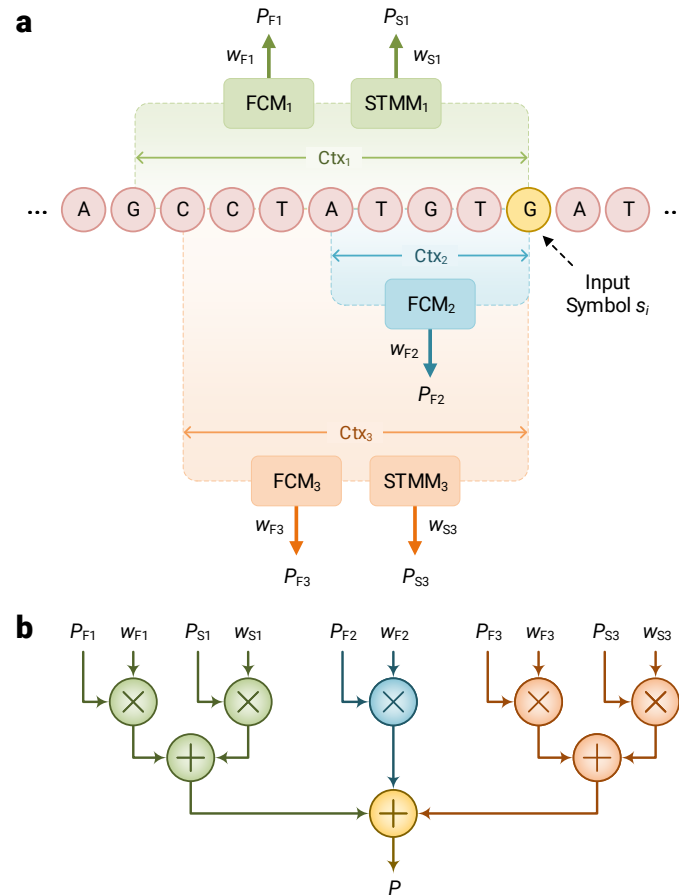


Figure 6: Data model used by Smash++. (a) cooperation between finite-context models (FCMs) and substitution-tolerant Markov models (STMMs). Note that each STMM needs to be associated with an FCM. (b) probability of an input symbol is estimated by employing the probability and weight values that have been obtained from processing previous symbols.

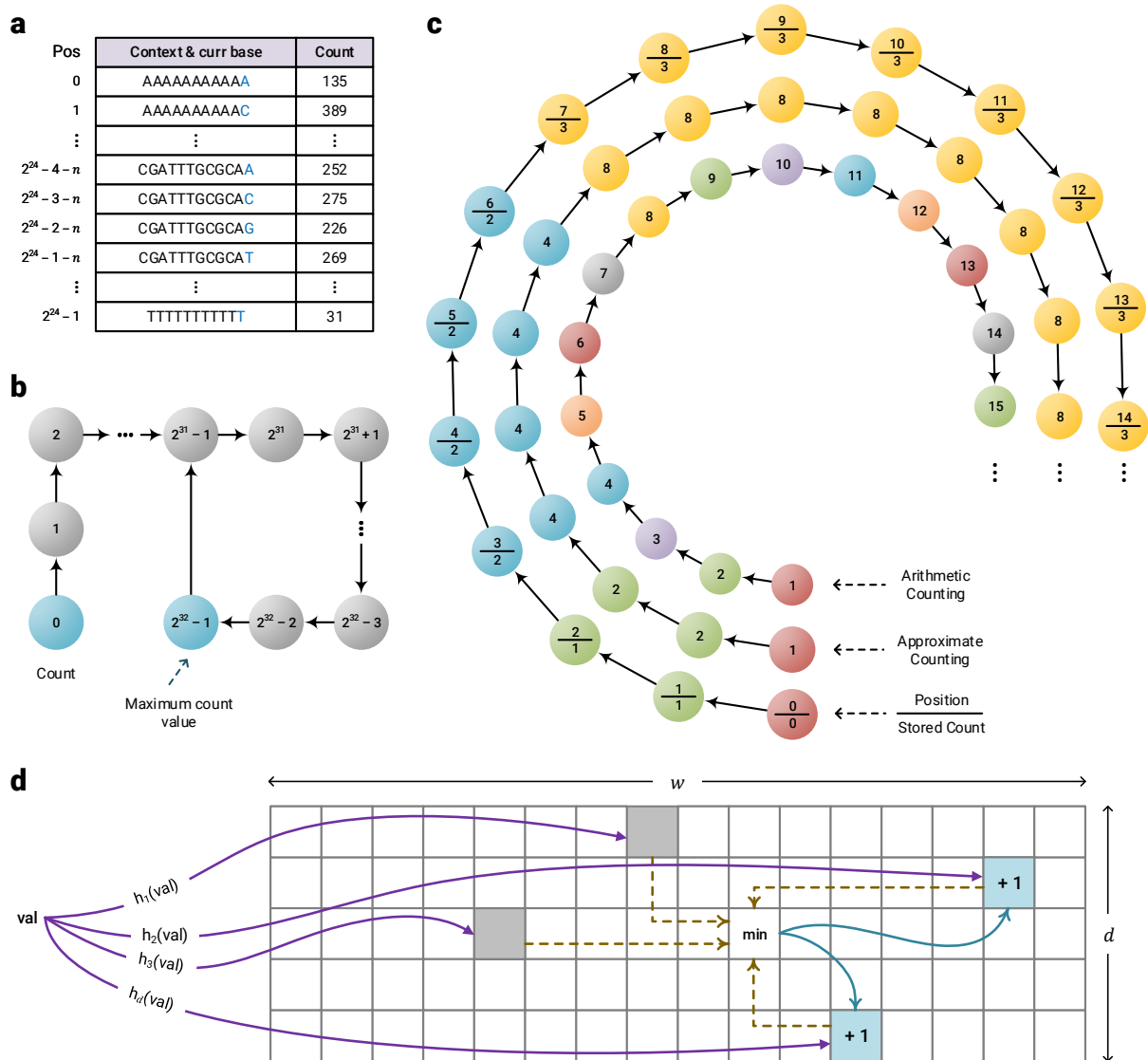


Figure 7: The data structures used by Smash++ to store the models in memory. (a) table of 64 bit counters that uses up to 128 MB of memory, (b) table of 32 bit counters that consumes at most 960 MB of memory, (c) table of 8 bit approximate counters with memory usage of up to 1 GB and (d) Count-Min-Log sketch of 4 bit counters which consumes up to  $\frac{1}{2}w \times d$  B of memory, e.g., if  $w = 2^{30}$  and  $d = 4$ , it uses 2 GB of memory.

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

4: function UPDATE( $x$ )
5:    $c \leftarrow \text{table}[x]$ 
6:   if INCREASEDECISION( $c$ ) = True then
7:      $\text{table}[x] \leftarrow c + 1$ 
8:   end if
9: end function

10: function QUERY( $x$ )
11:    $c \leftarrow \text{table}[x]$ 
12:   return  $2^c - 1$ 
13: end function
```

Figure 8: Approximate counting update and query.

```
Require: sketch width  $w$ , sketch depth  $d$ ,  $m$  bins, prime  
 $p \geq m$ , randomly chosen integers  $a_{1..d}$  and  $b_{1..d}$  modulo  
 $p$  with  $a \neq 0$   
  
1: function HASH( $k, x$ ) ▷ Universal hash family  
2:   return  $((a_k x + b_k) \bmod p) \bmod m$   
3: end function  
  
4: function MINCOUNT( $x$ )  
5:   minimum  $\leftarrow 15$  ▷ Biggest 4 bit number  
6:   for  $k \leftarrow 1$  to  $d$  do  
7:      $h \leftarrow$  HASH( $k, x$ )  
8:     if sketch[ $k$ ][ $h$ ] < minimum then  
9:       minimum  $\leftarrow$  sketch[ $k$ ][ $h$ ]  
10:    end if  
11:  end for  
12:  return minimum  
13: end function  
  
14: function INCREASEDECISION( $x$ )  
15:   return True with probability  $1/2^x$ , else False  
16: end function  
  
17: function UPDATE( $x$ )  
18:    $c \leftarrow$  MINCOUNT( $x$ )  
19:   if INCREASEDECISION( $c$ ) = True then  
20:     for  $k \leftarrow 1$  to  $d$  do  
21:        $h \leftarrow$  HASH( $k, x$ )  
22:       if sketch[ $k$ ][ $h$ ] =  $c$  then  
23:         sketch[ $k$ ][ $h$ ]  $\leftarrow c + 1$   
24:       end if  
25:     end for  
26:   end if  
27: end function  
  
28: function QUERY( $x$ )  
29:    $c \leftarrow$  MINCOUNT( $x$ )  
30:   return  $2^c - 1$   
31: end function
```

Figure 9: Count-Min-Log Sketch update and query.

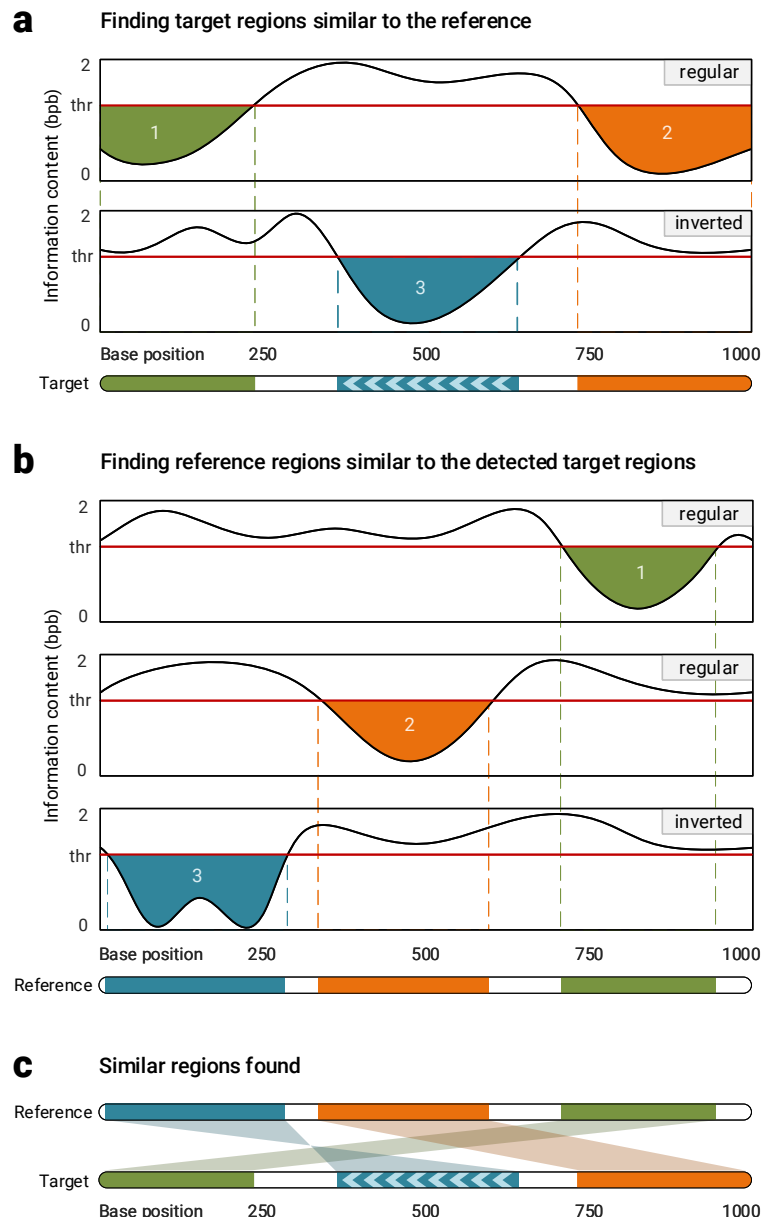


Figure 10: Finding similar regions in reference and target sequences. Smash++ finds, first, the regions in the target that are similar to the reference, and then, finds the regions in the reference that are similar to the detected target regions. This procedure is performance for both regular and inverted homologies.