

<https://helda.helsinki.fi>

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

Hosseini, Morteza

2020-05

Hosseini , M , Pratas , D , Morgenstern , B & Pinho , A J 2020 , ' Smash plus plus : an alignment-free and memory-efficient tool to find genomic rearrangements ' , GigaScience , vol. 9 , no. 5 , 048 . <https://doi.org/10.1093/gigascience/giaa048>

<http://hdl.handle.net/10138/316325>

<https://doi.org/10.1093/gigascience/giaa048>

cc_by

publishedVersion

Downloaded from Helda, University of Helsinki institutional repository.


This is an electronic reprint of the original article.

This reprint may differ from the original in pagination and typographic detail.


Please cite the original version.

TECHNICAL NOTE

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

Morteza Hosseini ^{1,*}, Diogo Pratas^{1,2}, Burkhard Morgenstern^{3,4} and Armando J. Pinho¹

¹IEETA/DETI, University of Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal; ²Department of Virology, University of Helsinki, Haartmaninkatu 3, 00014 Helsinki, Finland; ³Department of Bioinformatics, University of Göttingen, Goldschmidtstr. 1, 37077 Göttingen, Germany and ⁴Göttingen Center of Molecular Biosciences (GZMB), Justus-von-Liebig-Weg 11, 37077 Göttingen, Germany

*Correspondence address. Morteza Hosseini, IEETA/DETI, University of Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal. E-mail: seyedmorteza@ua.pt  <http://orcid.org/0000-0001-8962-8985>

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 owing 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 2 DNA sequences. This computational solution extracts information contents of the 2 sequences, exploiting a data compression technique to find 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 were in accordance 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

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]. Analysis 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 they usually occur in the presence of errors in cell division following meiosis or mitosis. These structural abnormalities in chromosomes, which 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, prognos-

Received: 11 January 2020; Revised: 6 April 2020; Accepted: 20 April 2020

© The Author(s) 2020. Published by Oxford University Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

tics, and targeted therapeutics [6, 7]. Examples of such diseases are Wolf-Hirschhorn syndrome, which is caused by a partial deletion from human chromosome location 4p16.3 [8]; Charcot-Marie-Tooth disease, which is most commonly caused by duplication of the gene encoding peripheral myelin protein 22 on human chromosome 17 [9]; and acute myeloid leukemia, which may be caused by translocations between human chromosomes 8 and 21 [10].

Various computational methods have been proposed in the literature that perform alignment, i.e., aligning regions that are conserved in 2 (or more) genomic sequences, for the purpose of detecting chromosomal rearrangements (comparing sequences) [11–16]. Alignment methods cannot be solely used to detect rearrangements because they follow the assumption that order of homology is maintained between the sequences to be compared [17, 18]. Alignment-free (AF) approaches, on the other hand, do not have this limitation; in addition, they offer computational speed-up advantages over alignment algorithms [19].

Among AF methods are information theory-based ones, which measure the amount of shared information within the sequences to quantify the similarity/dissimilarity between them. Information theory-based approaches have a broad range of applications, including but not limited to global and local characterization of DNA, e.g., prediction of transcription factor binding sites and classification of motifs, and gene mapping [20]. In 2015 another application of such approaches, namely, finding rearrangements between DNA sequences, was introduced [21]. Here, we provide a significant improvement over the mentioned method. It should be noted that the 2 alignment-based and AF approaches can be accompanied by genomic data visualization tools, which provide researchers with facilities to explore and analyze genomic data [22].

In this article we present Smash++, an AF tool that finds chromosomal rearrangements between 2 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 [23]. Associated with Smash++, we present a visualizer that is capable of visualizing as SVG images informationally similar regions between 2 genomic sequences. This tool also provides self- and relative redundancy (complexity) for the similar regions.

Smash++ is an improved version of Smash [21], featuring (i) improved accuracy, obtained by using multiple finite-context models (FCMs) along with substitution-tolerant Markov models to find fine-grained and coarse-grained chromosomal rearrangements; (ii) presentation of self-complexity (redundancy) and relative redundancy of informationally similar regions between 2 DNA sequences; (iii) improved user interface in the command line, by adding several options to customize the tool for running, and the resulting SVG image, by adding markers for positions of DNA bases and also plotting self- and relative redundancy; and (iv) improved performance, in terms of memory and time usage.

Results and Discussion

Implementation

Smash++ is implemented in the C++ language and is licensed under GNU GPLv3. It generates information maps for 2 sequences and, based on that, finds similar regions in them, in which there can potentially be DNA rearrangements. Therefore, Smash++ provides insight into positions of rearrangements

that have happened between 2 sequences. The tool comes with a visualizer, which 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, which is chosen randomly using an HSV color model. For more information about use of the tool, see Note S3 of the supplementary material.

The machine used for the tests had a 2-core 2.90 GHz Intel® Core™ i7-3520M CPU and 8 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.

Dataset

Smash++ and several other methods have been tested on a collection of synthetic and real sequences, which are described in Table 1. We used the GOOSE toolkit [24] 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 sizes of ~1 to ~127 Mb.

Application on synthetic data

Fig. 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 Fig. 1a, we generated 3 random sequences of size 500 bp, using GOOSE, and concatenated them. To build 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. Fig. 1b, c, and d follow the same procedure. To build the target in Fig. 1e, we mutated the first 1 kb block of the reference 1%, the second block 2%, and the third block 3%, up until the 60th block, which we mutated 60%.

The bottom sections of Fig. 1 show the output of the Smash++ visualizer, detecting similar regions regardless of their size. Note that for each detected region, the average value of redundancy and relative redundancy is illustrated. In Fig. 1b, Part II of the reference is mutated 90%; i.e., 9 of every 10 bases are 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, because 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. Fig. 1e shows that Smash++ is able to detect ~43% of mutation, which has been made possible by the use of substitution-tolerant Markov models (see Methods). Fig. 1 shows that Smash++ can be used to detect small-scale and large-scale similarities between DNA sequences.

Application on real data

Fig. 2 shows similarities between real sequences, found by Smash++. Fig. 2a 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, which are of great agricultural and commercial importance, are estimated to have diverged 37.2 million years ago (MYA) [27]. Fig. 2a and b demonstrate that Smash++ was

Table 1: Synthetic and real dataset used in the experiments

Sequence	Group	Length (bp)	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>G. 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 [25]
MAFF 311018	Bacteria	4,940,217	Access. AP008229 – <i>X. oryzae</i> pv. <i>oryzae</i> MAFF 311018 is a Japanese race 1 strain [26]
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 3 segments of 500 bp 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 4 segments of 25 kb size.
TarM	Synthetic	100,000	For building TarM, segment I of RefM (out of total 4) is inversely repeated, II is mutated 90%, III is duplicated, and IV is mutated 3%
RefL	Synthetic	5,000,000	It includes 2 segments, 2,500,000 bp 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 4 segments, 25,000,000 bp 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 kb 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 kb
TarComp	Synthetic	1,000,000	To build it, the first segment (I) of RefComp is duplicated, and the second, third, and fourth segments are mutated 1%, 2%, and 3%, respectively. Segments V, VI, and VII of RefComp are inversely repeated, then mutated 4%, 5%, and 6%, respectively. Finally, segments VIII, IX, and X are mutated 7%, 8%, and 9%, respectively.
RefPerm	Synthetic	3,000,000	It includes 3 segments of 1 Mb 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 bp.
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%.

The real dataset can be download from NCBI via accession number (access.) provided in the descriptions.

able to find the inversions confirmed by fluorescence *in situ* hybridization (FISH) analysis [28, 29].

In Figs 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 as follows: (a) Smash++; (b) progressiveMauve [11], which uses an alignment objective score to detect rearrangement breakpoints when genomes have unequal gene content. It also applies a probabilistic alignment filtering method to remove erroneous alignments of unrelated sequences; (c) the method proposed in [29], which takes a bacterial artificial chromosome-based approach along with FISH analysis to develop an integrated physical, genetic, and comparative map of chicken and turkey; (d) SynBrowser [15], which constructs synteny blocks using prebuilt alignments in the UCSC genome browser database; and (e) FISH analysis [28].

Fig. 2c demonstrates similarities between chromosomes 12 of *Homo sapiens* and *Pan troglodytes*, which are estimated to have diverged 6.7 MYA. A comparison with other methods is provided in Fig. S3 of the supplementary material. The methods include (a) Smash++; (b) progressiveMauve; (c) Cinteny [16], which 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. Embedded in this extension are “walks” through the leaves of the tree, which correspond to walks on the genome markers in their linear order; (d) SynBrowser; and (e) D-Genies [30], which works on the basis of alignment of genomes by minimap2 software package [31].

Fig. 2d illustrates similarities between *Xanthomonas oryzae* pv. *oryzae* PXO99A and *Xanthomonas oryzae* pv. *oryzae* MAFF 311018, 2 strains of *Xanthomonas oryzae* pv. *oryzae* (Xoo) pathogen, which causes the disease of bacterial blight of rice (*Oryza sativa* L.). It is the most serious bacterial disease of rice and can reduce yields by as much as 50% [25]. Note that to have a clearer picture, we have not plotted the shades connecting similar regions. This can be achieved using the “-l 6” option while calling the Smash++ visualizer. Figure S4 of the supplementary material provides the comparison of Smash++ with progressiveMauve and Salzberg et al. [25], 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 by Salzberg et al. [25], without performing an alignment.

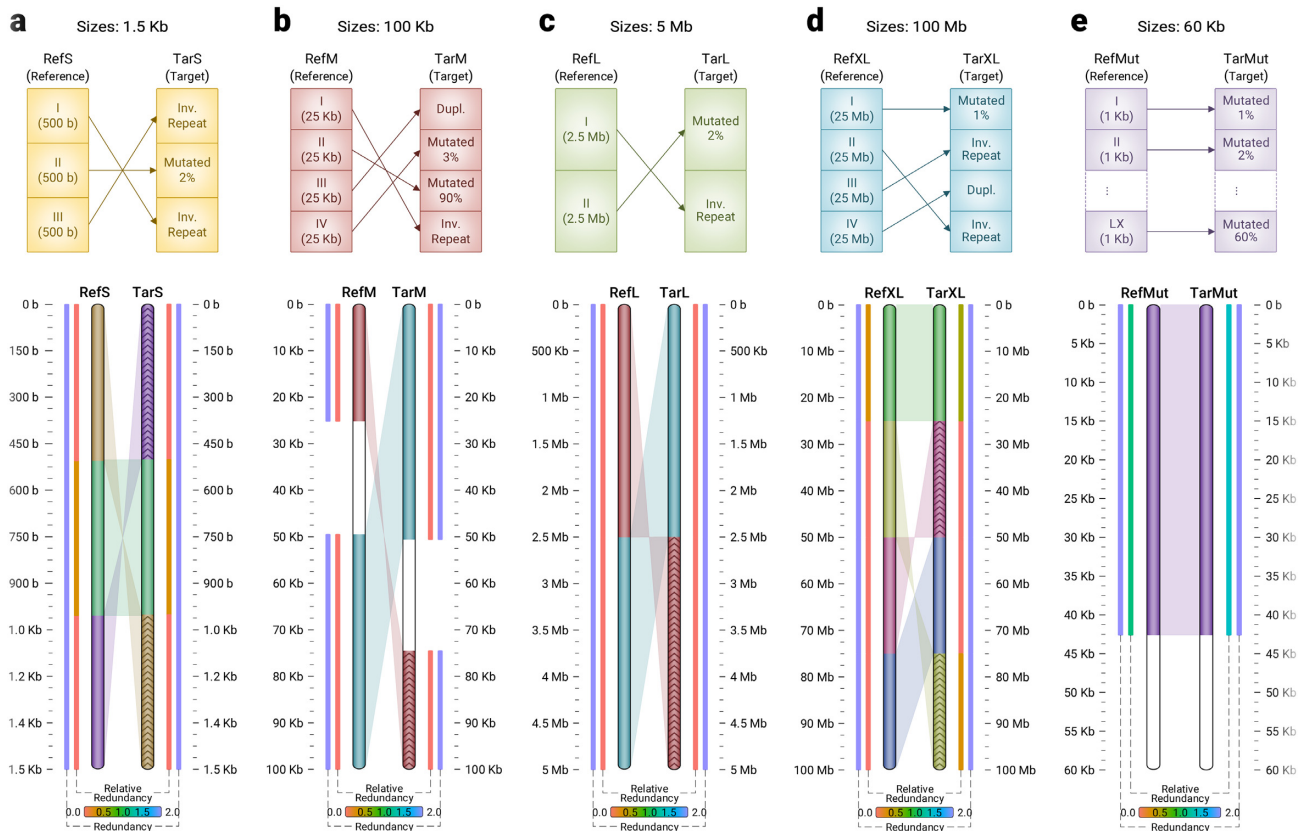


Figure 1: Similarities between synthetic sequences with different sizes, detected by Smash++. The parameters used are k -mer size = 14 and number of substitutions in substitution-tolerant Markov model (STMM) = 5, which are the default parameters used by Smash++. For the threshold, the default values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

Comparison with Smash

To better understand the improvement we have made over the first version, Smash, we compare the 2 tools on a synthetic and a real dataset (see Fig. 3). In Table 1, the procedure for creating the synthetic data (RefComp and TarComp) is described. Fig. 3a compares 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 combination 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 better able to model the data because it uses less information (lower information content) to describe the target based on the reference; this is possible because of using a combination 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 are 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 3 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 rearrangement map shows that Smash++ fulfills our expectation. On the other side, Smash was not able to de-

tect all rearrangements, showing that to model such a 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 Fig. 3b. *S. cerevisiae* is a species of yeast that plays a key role in winemaking, baking, and brewing. Its use as a eukaryotic model organism has provided insights into the molecular functioning of human cells [32]. *S. paradoxus* is the closest known species to *S. cerevisiae*, which has proved its importance in different fields of the life sciences, including evolution, ecology, and biotechnology [33]. 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 combined 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 finding rearrangements more accurately. The rearrangement map of Smash++ conforms to the previous study [32].

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 the robustness of the proposed tool against fragmented data, for different ran-

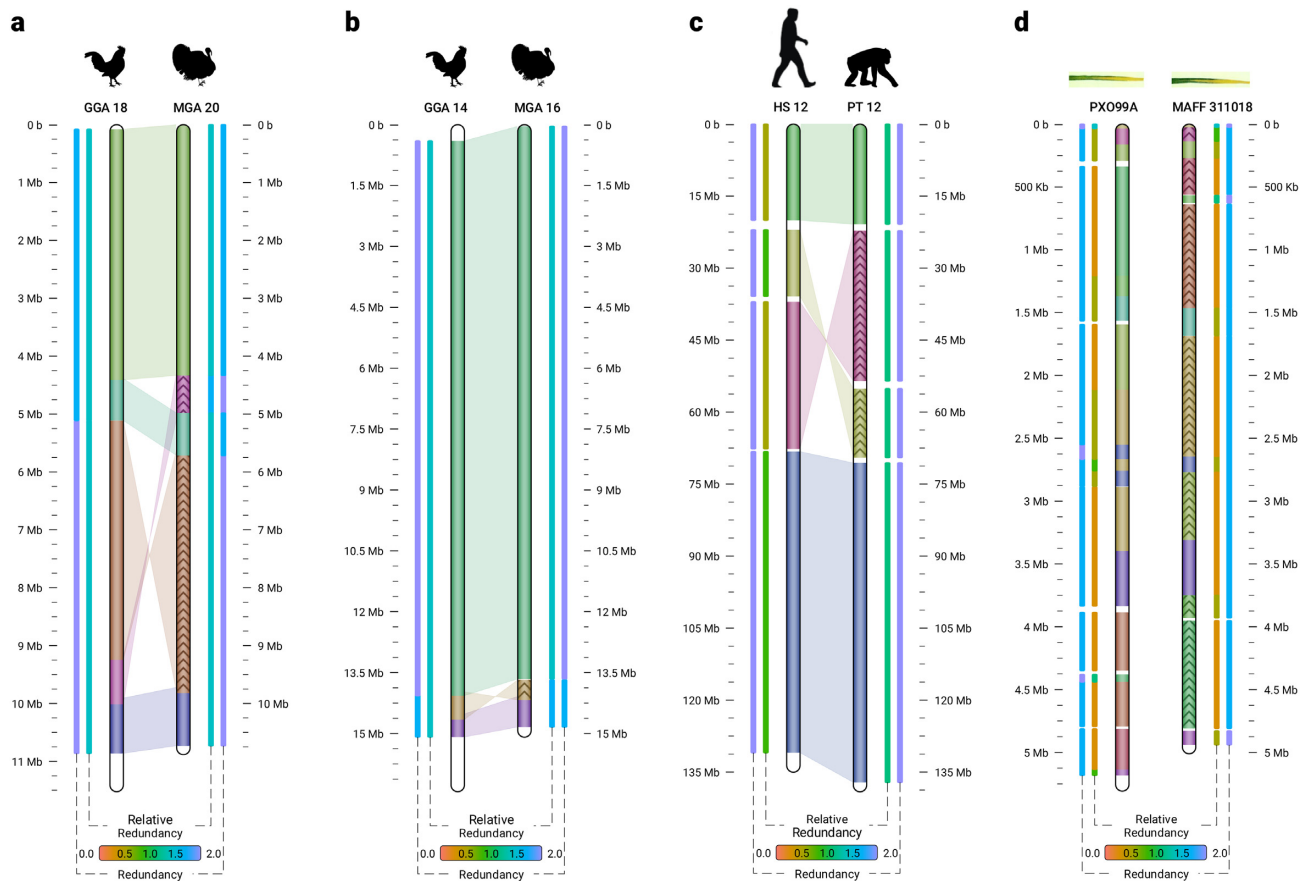


Figure 2: Similarities in a real dataset, detected by Smash++. (a) *G. gallus* (chicken) chr. 18 and *M. gallopavo* (turkey) chr. 20. The parameters were k -mer size = 14, No. substitutions in STMM = 5, threshold = 1.9, and minimum block size (m) = 500,000; i.e., regions smaller than 500,000 bp were 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 was obtained by setting k = 13, threshold = 1.55, and m = 10,000.

domly permuted block sizes. As can be seen, the same 3 target regions are detected even when the reference is fragmented to 100,000 blocks of 30 bp. This capability might be of interest in the case of non-assembled sequences or in the presence of assembly errors; note that this approach cannot be considered as an alternative to assembly.

Benchmarking

Fig. 4 illustrates the performance of the proposed tool in terms of memory and time usage for all datasets (see supplementary Table S1 for more details). Size of the datasets and number of detected similar regions between each pair of sequences (“# Rearr”) are shown at the bottom of the figure. The pair dataset “Perm30” and the pair “Perm1000” are, as outliers, not shown. Fig. 4a shows the peak memory in gigabytes used by Smash++ on all synthetic and real datasets. As can be seen, it is ~1 GB for all datasets. The maximum peak memory, ~1.08 GB, was used when the proposed tool was run on human and chimpanzee chromosomes 12. It should be mentioned that the memory usage of Smash++ is related to the k -mer size that is used to model the data because different data structures are used for different k -mer sizes (see Methods). Sizes 13 and 14 were used to perform the experiments. The maximum memory usage of ~1 GB enables Smash++ to run on any computer nowadays.

Fig. 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 greater the number of regions and/or the greater the dataset size, the more time will be taken. Note that it is not a linear relation. As an example, the pair dataset “Large” and the pair “PXO99A_MAFF311018” have approximately the same total size of 10 Mb; but in the former case that 2 similarities is detected, Smash++ takes ~16 seconds, and in the latter case with 23 similarities (~12 times more than the former case), the proposed tool takes ~48 seconds (3 times more) to run. As another example, carrying out Smash++ on the pair “Large” with 50 times larger size than the pair “Medium” leads to detection of the same number of rearrangements, i.e., 2, while it takes only ~3 times more time. Regarding the pair “Perm30” with 11,565 similarities detected (Supplementary Table S1), we note that it has a massively fragmented reference sequence with 10,000 fragments of 30 bp; 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 11,565 (number of similar regions) arises from the fact that a number of the reference chunks are similar to >1 target region and vice versa.

A major advantage of Smash++ over Smash is using a combination of FCMs and STMMs to better model the data; however, to have an idea about how the performance of these 2

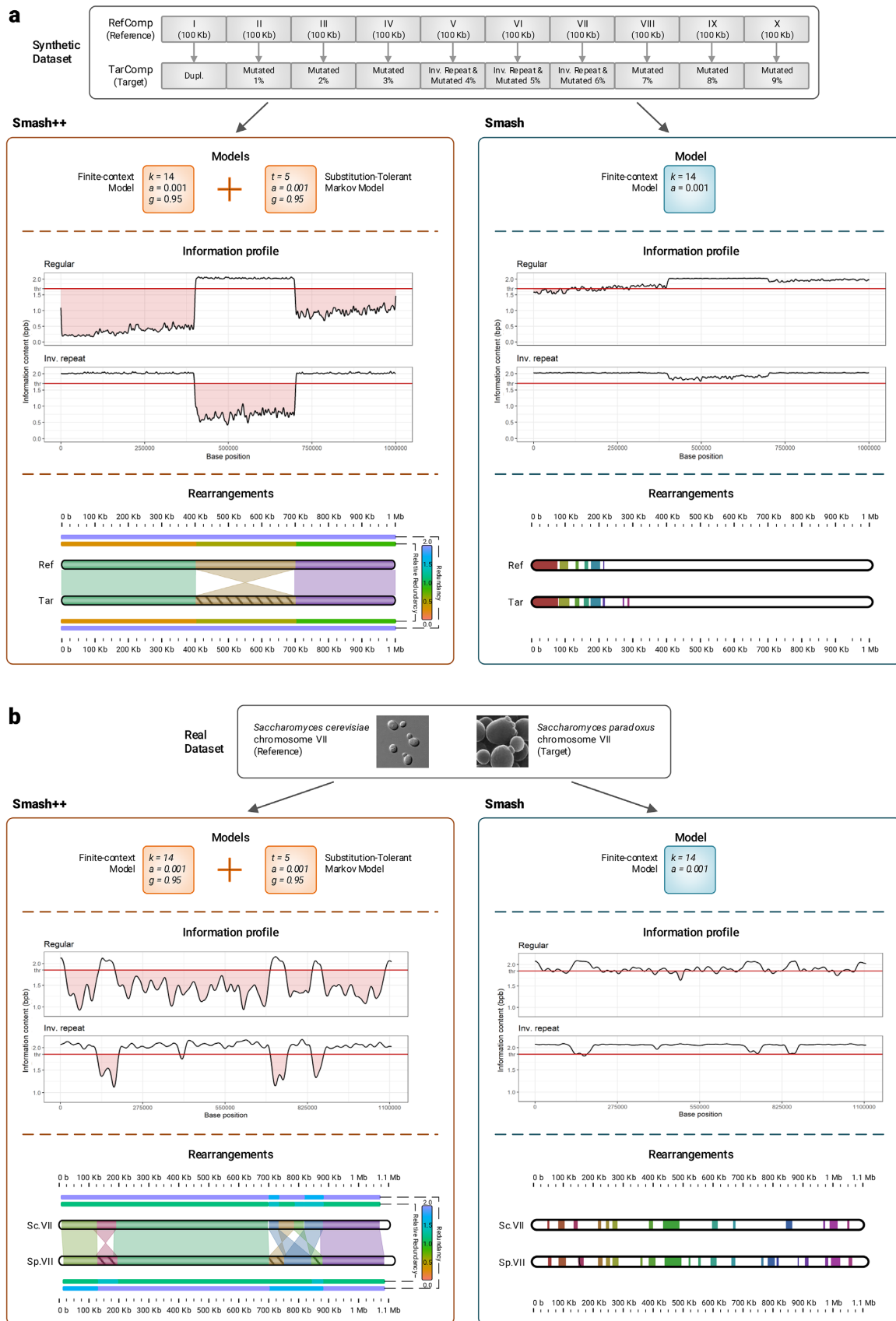


Figure 3: Comparison of Smash++ and Smash. (a) Performance on a synthetic dataset. Using the combination of an FCM and an STMM (in Smash++) produces more accurate results than using a single FCM (in Smash). (b) Performance on a real dataset, including *S. cerevisiae* chr. VII and *S. paradoxus* chr.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

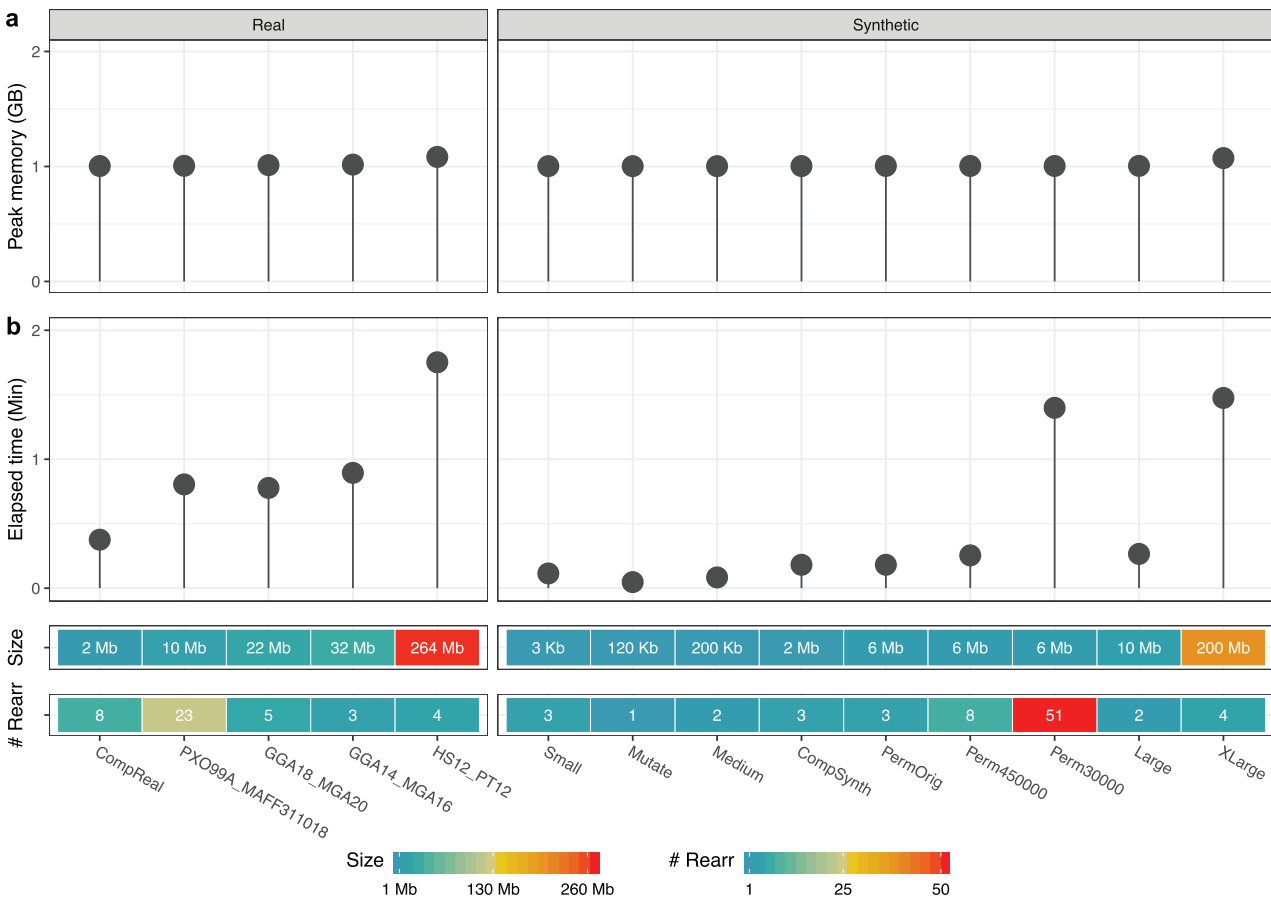


Figure 4: (a) The peak memory consumption, in gigabytes; and (b) the elapsed (wall clock) time usage, in minutes, of Smash++ obtained by running on all synthetic and real datasets described in Table 1.

tools can be compared, we let Smash++ run with only 1 FCM on the dataset described in Table 1. We also did not compute self-complexity, similarly to Smash. Fig. 5 and supplementary Table S2 show the results. In Fig. 5a, the range of peak memory usages (from minimum to maximum usage) are compared, while running Smash and Smash++ on different real and synthetic datasets. The diamond symbol shows the mean value. The results show that the maximum peak memory usage by Smash++ is 1.9 times less than that by Smash. In Fig. 5b, the range of wall clock times is compared for these 2 tools. As mentioned in the figure, Smash++ runs 5.4 times faster than Smash on the tested datasets. It is worth mentioning that due to the absence of another tool that provides relative compression in addition to detecting rearrangements, we cannot have a fair quantitative comparison to other tools, in terms of time and memory usage; therefore, we have only included the results obtained by Smash++ and Smash.

Conclusions

Finding genomic rearrangements is crucial because they play an important role in genetic disorders, cancer, and chromosomal evolution. We presented Smash++, an AF tool that accurately finds small- and large-scale genomic rearrangements between pairs of DNA sequences, by using 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 show that the detected rearrangements were in accordance with previous studies, which used alignment-based methods or performed FISH analysis. Smash++ consumed a maximum of ~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 the accuracy of diagnostic and genetic counselling and also to guide future investigations into the development of personalized therapeutics.

Methods

The schema of the proposed method is illustrated in Fig. 6. Smash++ takes as inputs a reference and a target sequence and produces as output a position file, including local similarities of the 2 sequences, which can then be used by the Smash++ visualizer to produce an SVG image illustrating the similarities. This process has 8 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

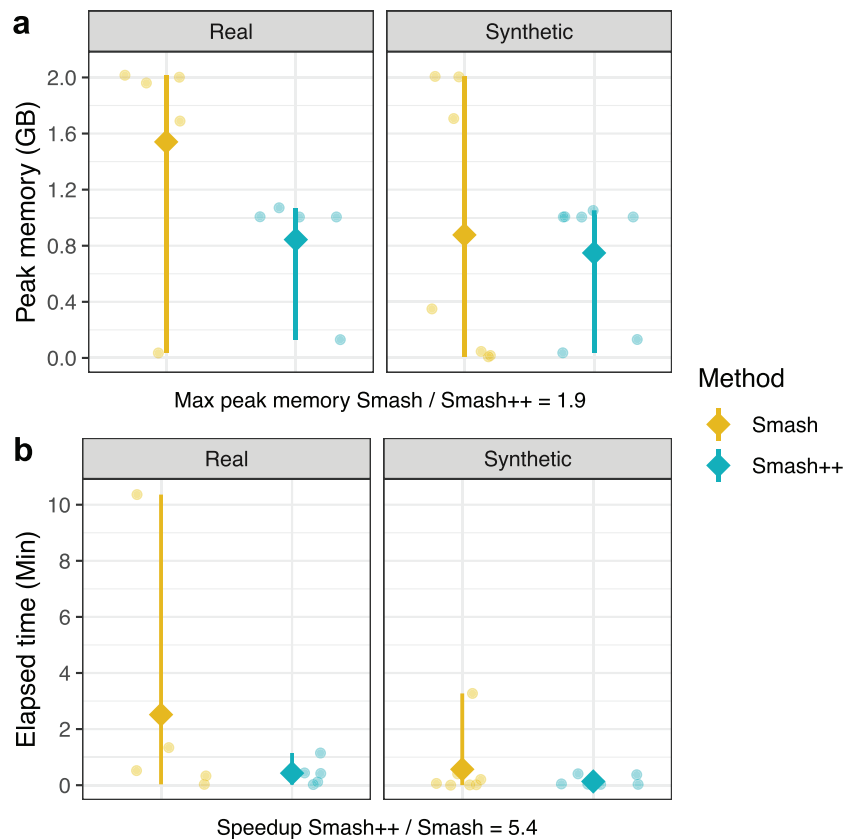


Figure 5: Comparison of Smash++ and Smash, in terms of (a) memory usage; and (b) time usage, running on real and synthetic data described in Table 1. To have a fair comparison, only 1 model (FCM) is used by Smash++, and also self-complexity is not computed. Diamonds indicate the mean, and bars, the ranges from minimum to maximum values.

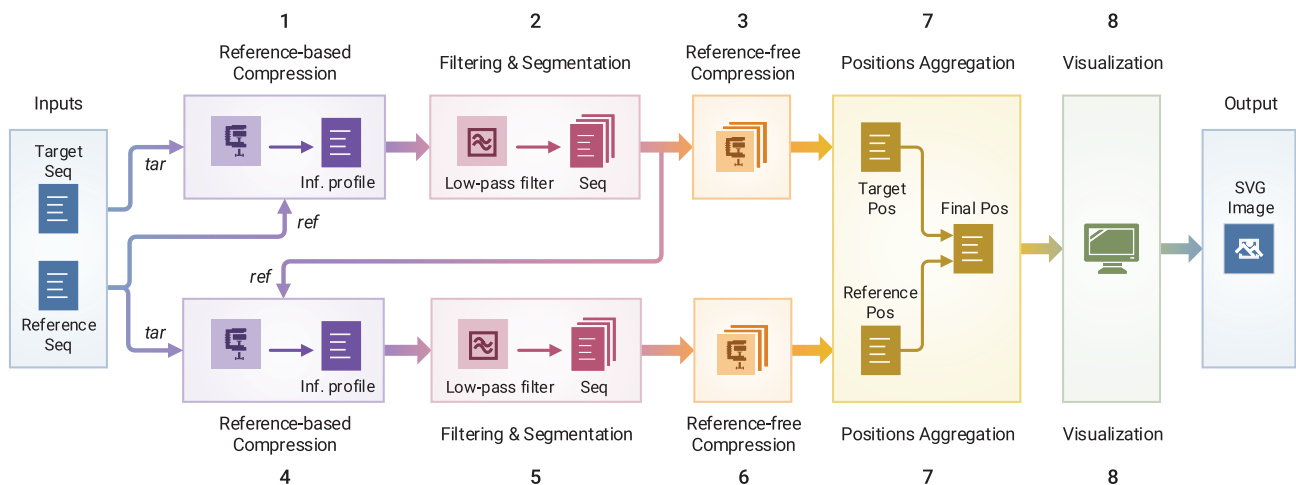


Figure 6: The schema of Smash++. The process of finding similar regions in reference and target sequences and computing the redundancy in each region includes 8 stages. Smash++ outputs a *.pos file that includes the positions of the similar regions, and can be then visualized, resulting in an SVG image.

by stages 3 and 6; and (8) visualizing the positions. The following sections describe the process in detail.

Data modeling

We consider sequences over the nucleotide alphabet $\Theta = \{A, C, G, T\}$; our goal is to measure the degree of local similarity be-

tween 2 such sequences. More specifically, we consider a reference sequence $S = s_1, \dots, s_N$ over Θ , and we want to measure the local information content of a target sequence, given this reference sequence. To this end, we use a combination of finite-context models and substitution-tolerant Markov models to derive different probability measures for observing a nucleotide x in a sequence, given the context of the previous k nucleotides

(Fig. 7a); these probabilities are then mixed (by multiplications and additions shown in Fig. 7b) to provide the final probability (P) of observing the nucleotide x . The following subsections describe the models we use in detail.

Finite-context model

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 [23, 34, 35], we then define

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

where “4” is the size of alphabet Θ and α 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 on the state of the preceding k -mer.

Substitution-tolerant Markov model

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

$$s'_i = \begin{cases} A & \text{if } i < 1 \\ \arg \max_{x \in \Theta} P_{FCM}(x|s'_{-k}, \dots, s'_{i-1}) & \text{if } i \geq 1. \end{cases} \tag{2}$$

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

$$P_{STMM}(x|Q) = \frac{N'(x|Q) + \alpha}{N'(Q) + 4 \cdot \alpha}. \tag{3}$$

STMMs, which are probabilistic-algorithmic models [23, 36], 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 an 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 T, respectively; also, the next symbol to appear in the sequence is T. An FCM considers the next context as GACGTACT, while an STMM considers it as GACGTACG because 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

$$P(x) = \sum_{i=1}^m P_{FCM_i}(x|Q) w_i + \sum_{j=1}^n P_{STMM_j}(x|Q) w_j, \quad \forall x \in S. \tag{4}$$

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

$$\begin{aligned} w_{i_p} &\propto (w_{i_{p-1}})^{\gamma_i} P_{FCM}(x|Q_{p-1}), & 1 \leq i \leq m, \\ w_{j_p} &\propto (w_{j_{p-1}})^{\gamma_j} P_{STMM}(x|Q_{p-1}), & 1 \leq j \leq n. \end{aligned} \tag{5}$$

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

$$\sum_{i=1}^m w_i + \sum_{j=1}^n w_j = 1. \tag{6}$$

By experimenting with different forgetting factors and context-order sizes for models, we have found that the factors are directly related to the context sizes and reciprocally related to the complexity (see Fig. S6 of the supplementary material).

Storing models in memory

The FCMs and STMMs include, in fact, count values that need to be saved in memory. For this purpose, 4 different data structures have been used 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 (CMLS) of 4-bit counters, for $k \geq 15$.

The table of 64-bit counters (Fig. 8a) 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 $2^{32} - 1 = 4,294,967,295$, all the counts will be renormalized by dividing by 2, as shown in Fig. 8b.

Approximate counting is a method that use probabilistic techniques to count large number of events while using a small amount of memory [37]. Fig. 9 shows the algorithm for 2 major functions associated with this method, Update and Query. To update the counter, a pseudo-random number generator 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. Fig. 8c shows the difference between arithmetic and approximate counting, and also the values that are actually stored in memory. Note that because 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 $\log_2 8 = 3$.

CMLS is a probabilistic data structure to save the frequency of events in a table by means of a family of independent hash functions [38]. The algorithm for updating and querying the counter is shown in Fig. 10. To update the counter, its current value is hashed with d independent hash functions. Then, a coin is

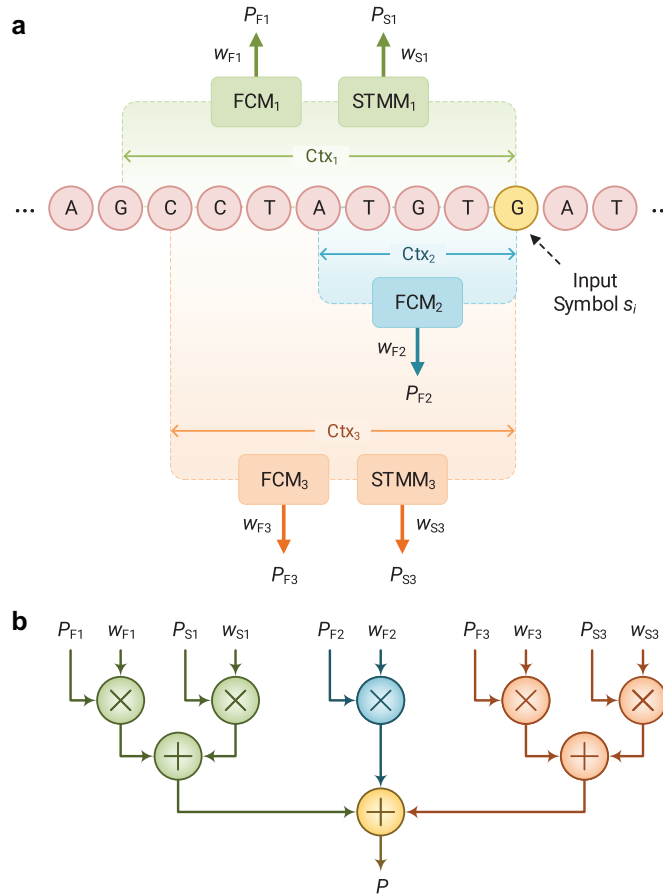


Figure 7: Data model used by Smash++. (a) Cooperation between finite-context models (FCMs) and substitution-tolerant Markov models (STMMs). Note that each STMM needs to be associated with an FCM. (b) Probability of an input symbol is estimated by using the probability and weight values that have been obtained from processing previous symbols.

flipped the number of times of the counter's current value, using 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. 8d.

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

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m, \quad (7)$$

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 2, $m = 2^M$, the multiply-add-shift scheme [39] can be used to avoid modular arithmetic. A hash function in this scheme can be obtained by:

$$h_{a,b}(x) = ((ax + b) \bmod 2^w) \text{div } 2^{w-M}, \quad (8)$$

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

C++ language by

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

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 (bits per base), which can be calculated as

$$I(x) = -\log_2 P(x), \quad \forall x \in S, \quad (9)$$

where $P(x)$ denotes the probability of observing a nucleotide x in the sequence S , obtained by Equation 4.

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 2 regions, the more information is shared between them, and, therefore, the more similar are the 2 regions. Note that a version of this measure has been introduced by Pratas et al. [21], who used a single FCM to calculate the probabilities. In this article, however, we exploit a cooperation between multiple FCMs and STMMs for highly accurate calculation of such probabilities.

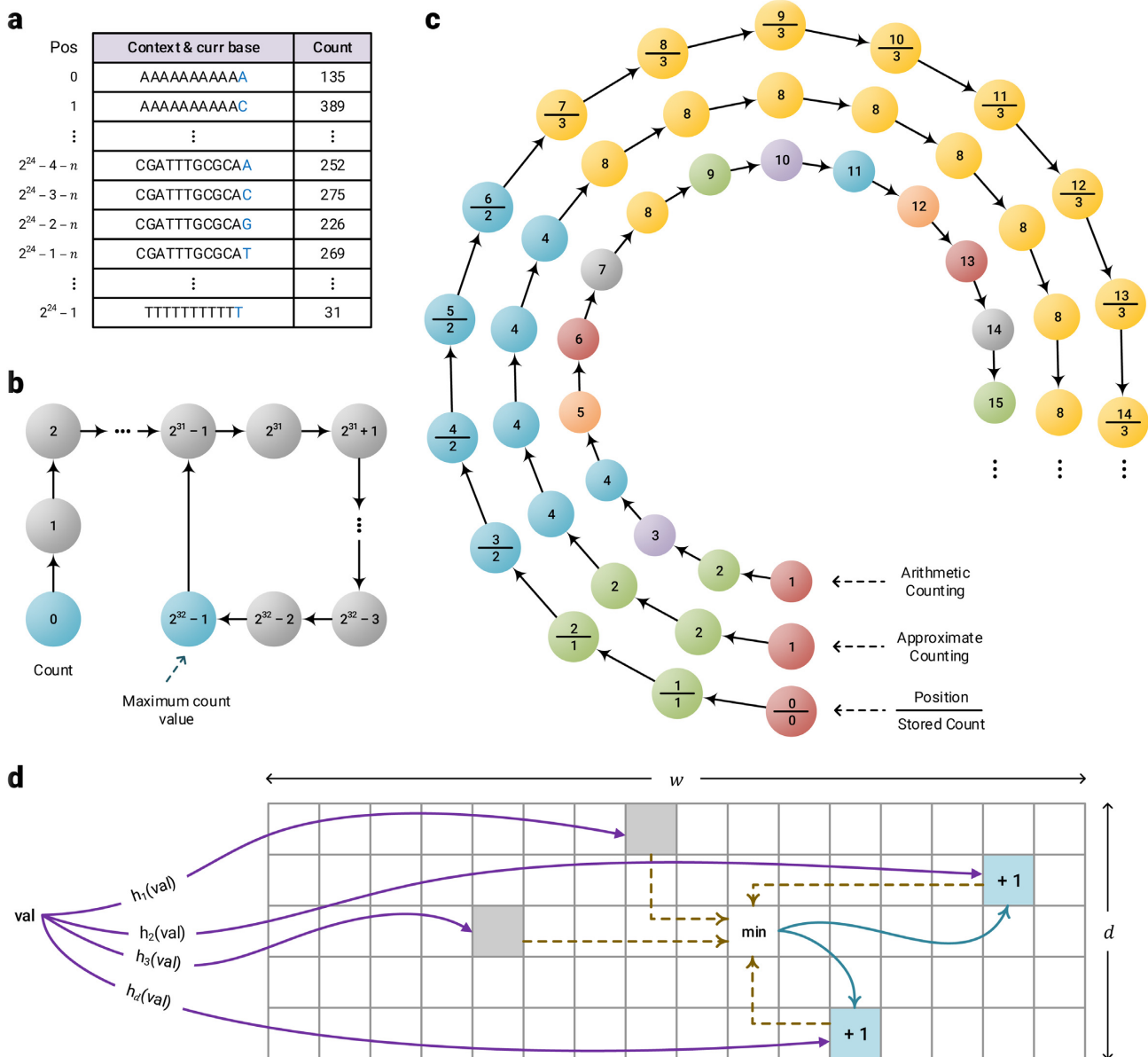


Figure 8: The data structures used by Smash++ to store the models in memory. (a) Table of 64-bit counters that uses up to 128 MB of memory, (b) table of 32 bit counters that consumes at most 960 MB of memory, (c) table of 8 bit approximate counters with memory usage of up to 1 GB, and (d) Count-Min-Log 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.

The procedure of finding similar regions in a reference and a target sequence, illustrated in Fig. 11, 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 [40], which is a discrete window function given by $w[n] = 0.5 - 0.5 \cos(2\pi n/N)$, 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 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 (Fig. 11a). The described phase repeats for all of the target regions found, in such a way that after creating the model for each region, the whole reference se-

quence is compressed to find those regions in the reference that are similar to each of the target regions (Fig. 11b). The final result would have the form of Fig. 11c.

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 [41]. The Kolmogorov complexity, K , of a binary string s , of finite length, is the length of a shortest 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 [42, 43].

```

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 9: 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:      $h \leftarrow \text{HASH}(k, x)$ 
8:     if  $\text{sketch}[k][h] < \text{minimum}$  then
9:       minimum  $\leftarrow \text{sketch}[k][h]$ 
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:    $c \leftarrow \text{MINCOUNT}(x)$ 
19:   if INCREASEDECISION( $c$ ) = True then
20:     for  $k \leftarrow 1$  to  $d$  do
21:        $h \leftarrow \text{HASH}(k, x)$ 
22:       if  $\text{sketch}[k][h] = c$  then
23:          $\text{sketch}[k][h] \leftarrow c + 1$ 
24:       end if
25:     end for
26:   end if
27: end function

28: function QUERY( $x$ )
29:    $c \leftarrow \text{MINCOUNT}(x)$ 
30:   return  $2^c - 1$ 
31: end function

```

Figure 10: Count-Min-Log Sketch update and query.

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 used for this purpose [44–46]. In this article, we use a reference-free compressor to approximate the complexity and, conse-

quently, the redundancy of the found similar regions in the reference and the target sequences. This compressor works on the basis of cooperation of FCMs and STMMs, which has been previously described in detail. Note that the difference between a reference-based and reference-free version of such a compressor is that, in the former mode, a model is first created for the reference sequence and then the target sequence is compressed on the basis of 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

RRID: SCR_018307

Availability of Supporting Data and Materials

The datasets supporting the results of this article are available in the Smash++ Github repository, <https://github.com/smortezah/smashpp/tree/master/experiment/dataset>. Snapshots of our code and other supporting data are available in the GigaScience repository, GigaDB [47].

Additional Files

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* MAF 311018.

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

Supplementary Figure S6. Relation between context-order sizes, forgetting factors, and complexity (information content).

Supplementary Table S1. Performance of Smash++, in terms of memory and time usage, running on different synthetic and real datasets.

Supplementary Table S2. Comparison of the performance of Smash++ and Smash, running on different synthetic and real datasets.

Supplementary Note S1. Software manual for Smash++.

Abbreviations

AF: alignment-free; bp: base pairs; CMLS: Count-Min-Log Sketch; CPU: central processing unit; FCM: finite-context model; FISH: fluorescence in situ hybridization; GGA: *Galus gallus*; HS: *Homo sapiens*; HSV: hue, saturation, value; HTS: high-throughput sequencing; Mb: megabase pairs; MGA: *Meleagris gallopavo*; MYA: million years ago; NCBI: National Center for Biotechnology Information; PMP22: periph-

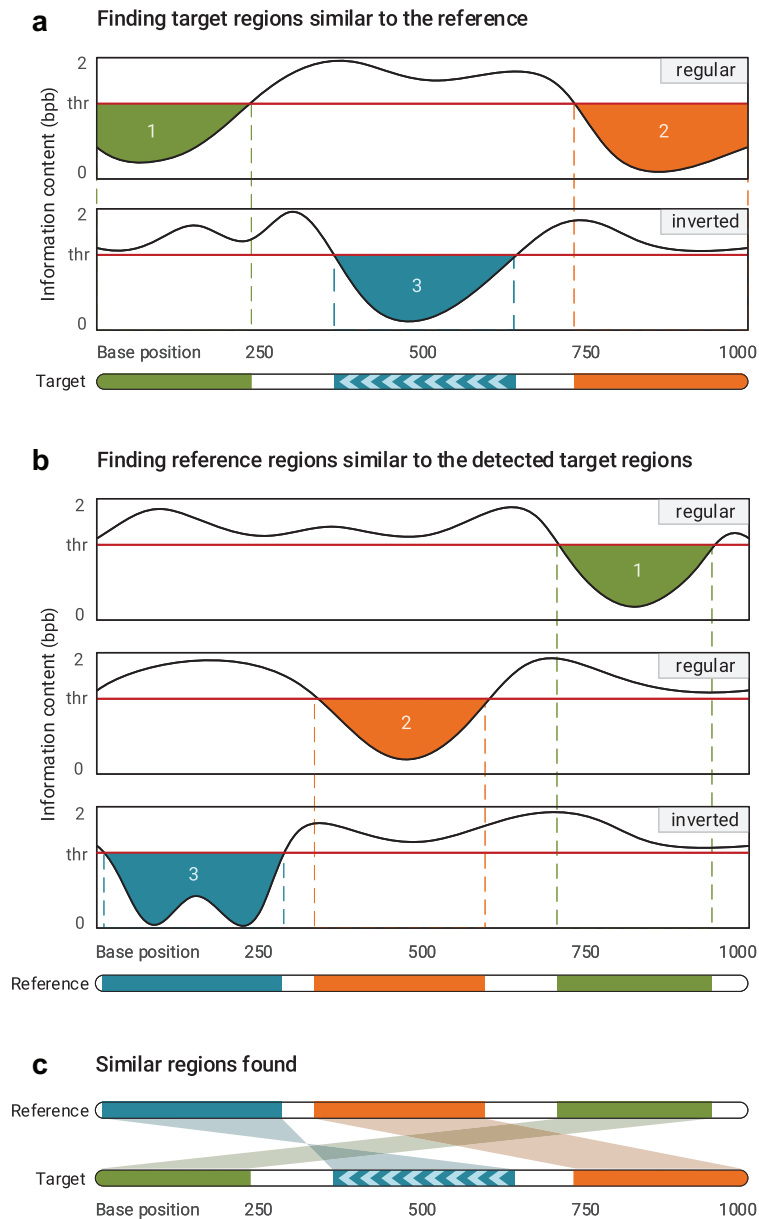


Figure 11: Finding similar regions in reference and target sequences. Smash++ first finds the regions in the target that are similar to the reference and then finds the regions in the reference that are similar to the detected target regions. This procedure is performance for both regular and inverted homologies.

eral myelin protein 22; PT: *Pan troglodytes*; RAM: random access memory; Sc: *Saccharomyces cerevisiae*; Sp: *Saccharomyces paradoxus*; STMM: substitution-tolerant Markov model; SVG: Scalable Vector Graphics; UCSC: University of California, Santa Cruz.

Competing Interests

The authors declare that they have no competing interests.

Funding

M.H. was supported by PhD MAP-i grant (PD/BD/113969/2015) from Foundation for Science and Technology (FCT) in Portugal. D.P. was funded by Scientific Employment Stimulus Program (CI-CTTI-94-ARH/2019) from FCT. M.H., D.P., and A.J.P.

were supported by Operational Program of Competitiveness and Internationalization (COMPETE) (UID/CEC/00127/2019 and UIDB/00127/2020) from FCT.

Authors' 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 review the final manuscript and provide critical comments.

Acknowledgements

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

References

1. Reuter J, Spacek DV, Snyder M. High-throughput sequencing technologies. *Mol Cell* 2015;**58**(4):586–97.
2. Villamor DEV, Ho T, Al Rwahnih M, et al. High throughput sequencing for plant virus detection and discovery. *Phytopathology* 2019;**109**(5):716–25.
3. Rego SM, Snyder MP. High throughput sequencing and assessing disease risk. *Cold Spring Harb Perspect Med* 2019;**9**(1), doi:10.1101/cshperspect.a026849.
4. Hartmann T, Middendorf M, Bernt M. Genome rearrangement analysis: cut and join genome rearrangements and gene cluster preserving approaches. In: Setubal J, Stoye J, Stadler P, eds. *Comparative Genomics*. New York: Humana; 2018:261–89.
5. Gardner R, Gardner RM, Amor DJ. *Gardner and Sutherland's Chromosome Abnormalities and Genetic Counseling*. No. 70. Oxford University Press; 2018.
6. Theisen A, Shaffer LG. Disorders caused by chromosome abnormalities. *Appl Clin Genet* 2010;**3**:159.
7. Damas J, Samuels DC, Carneiro J, et al. Mitochondrial DNA rearrangements in health and disease—a comprehensive study. *Hum Mutat* 2014;**35**(1):1–14.
8. Dufke A, Seidel J, Schöning M, et al. Microdeletion 4p16.3 in three unrelated patients with Wolf-Hirschhorn syndrome. *Cytogenet Genome Res* 2000;**91**(1-4):81–4.
9. Timmerman V, Nelis E, Van Hul W, et al. The peripheral myelin protein gene PMP-22 is contained within the Charcot-Marie-Tooth disease type 1A duplication. *Nat Genet* 1992;**1**(3):171.
10. Huang L, Abruzzo LV, Valbuena JR, et al. 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. *Am J Clin Pathol* 2006;**125**(2):267–72.
11. Darling AE, Mau B, Perna NT. progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement. *PLoS One* 2010;**5**(6):e11147.
12. Brudno M, Malde S, Poliakov A, et al. Glocal alignment: finding rearrangements during alignment. *Bioinformatics* 2003;**19**(suppl 1):i54–62.
13. Pham SK, Pevzner PA. DRIMM-Synteny: decomposing genomes into evolutionary conserved segments. *Bioinformatics* 2010;**26**(20):2509–16.
14. Pevzner P, Tesler G. Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome Res* 2003;**13**(1):37–45.
15. Lee J, Hong WY, Cho M, et al. SyntenY Portal: a web-based application portal for syntenY block analysis. *Nucleic Acids Res* 2016;**44**(W1):W35–40.
16. Sinha AU, Meller J. Cinteny: flexible analysis and visualization of syntenY and genome rearrangements in multiple organisms. *BMC Bioinformatics* 2007;**8**(1):82.
17. Zielezinski A, Girgis HZ, Bernard G, et al. Benchmarking of alignment-free sequence comparison methods. *Genome Biol* 2019;**20**(1):144.
18. Frith MC, Khan S. A survey of localized sequence rearrangements in human DNA. *Nucleic Acids Res* 2017;**46**(4):1661–73.
19. Zielezinski A, Vinga S, Almeida J, et al. Alignment-free sequence comparison: benefits, applications, and tools. *Genome Biol* 2017;**18**(1):186.
20. Vinga S. Information theory applications for biological sequence analysis. *Brief Bioinform* 2013;**15**(3):376–89.
21. Pratas D, Silva RM, Pinho AJ, et al. An alignment-free method to find and visualise rearrangements between pairs of DNA sequences. *Sci Rep* 2015;**5**:10203.
22. Nielsen CB, Cantor M, Dubchak I, et al. Visualizing genomes: techniques and challenges. *Nat Methods* 2010;**7**(3s):S5.
23. Hosseini M, Pratas D, Pinho AJ. AC: a compression tool for amino acid sequences. *Interdiscip Sci* 2019;**11**(1):68–76.
24. Goose GitHub Repository <https://github.com/pratas/goose>. Accessed 1st March 2020
25. Salzberg SL, Sommer DD, Schatz MC, et al. Genome sequence and rapid evolution of the rice pathogen *Xanthomonas oryzae* pv. *oryzae* PXO99A. *BMC Genomics* 2008;**9**(1):204.
26. Ochiai H, Inoue Y, Takeya M, et al. Genome sequence of *Xanthomonas oryzae* pv. *oryzae* suggests contribution of large numbers of effector genes and insertion sequences to its race diversity. *Japan Agric Res Q* 2005;**39**(4):275–87.
27. Kumar S, Stecher G, Suleski M, et al. TimeTree: a resource for timelines, timetrees, and divergence times. *Mol Biol Evol* 2017;**34**(7):1812–9.
28. Dalloul RA, Long JA, Zimin AV, et al. Multi-platform next-generation sequencing of the domestic turkey (*Meleagris gallopavo*): genome assembly and analysis. *PLoS Biol* 2010;**8**(9):e1000475.
29. Zhang Y, Zhang X, O'Hare TH, et al. A comparative physical map reveals the pattern of chromosomal evolution between the turkey (*Meleagris gallopavo*) and chicken (*Gallus gallus*) genomes. *BMC Genomics* 2011;**12**(1):447.
30. Cabanettes F, Klopp C. D-GENIES: dot plot large genomes in an interactive, efficient and simple way. *PeerJ* 2018;**6**:e4958.
31. Li H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 2018;**34**(18):3094–100.
32. Fischer G, Rocha EP, Brunet F, et al. Highly variable rates of genome rearrangements between hemiascomycetous yeast lineages. *PLoS Genet* 2006;**2**(3):e32.
33. Charron G, Leducq JB, Bertin C, et al. Exploring the northern limit of the distribution of *Saccharomyces cerevisiae* and *Saccharomyces paradoxus* in North America. *FEMS Yeast Res* 2014;**14**(2):281–8.
34. Sayood K. *Introduction to Data Compression*. Morgan Kaufmann; 2017.
35. Pinho AJ, Pratas D. MFCOMPRESS: a compression tool for FASTA and multi-FASTA data. *Bioinformatics* 2013;**30**(1):117–8.
36. Pratas D, Hosseini M, Pinho AJ. Substitutional tolerant Markov models for relative compression of DNA sequences. In: Fdez-Riverola F, Mohamad M, Rocha M, et al., eds. *International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB)*. Cham: Springer; 2017:265–72.
37. Morris R. Counting large numbers of events in small registers. *Commun ACM* 1978;**21**(10):840–2.
38. Pitel G, Fouquier G. Count-min-log sketch: approximately counting with approximate counters. In: *International Symposium on Web Algorithms*, Deauville, France; 2015.
39. Woelfel P. Efficient strongly universal and optimally universal hashing. In: Kutylowski M, Pacholski L, Wierzbicki T, eds. *International Symposium on Mathematical Foundations of Computer Science*. Springer; 1999:262–72.
40. Blackman R, Tukey J. Particular pairs of windows. In: *The Measurement of Power Spectra, from the Point of View of Communications Engineering*. New York: Dover; 1959:95–101.
41. Hosseini M, Pratas D, Pinho AJ. Cryfa: a secure encryption tool for genomic data. *Bioinformatics* 2018;**35**(1):146–8.

42. Turing A. On computable numbers, with an application to the Entscheidungsproblem. *Proc Lond Math Soc* 1936;**42**(2):230–65.
43. Li M, Vitányi P. *An Introduction to Kolmogorov Complexity and Its Applications*. 3rd ed. Springer; 2009.
44. Zenil H, Soler-Toscano F, Delahaye JP, et al. Two-dimensional Kolmogorov complexity and an empirical validation of the Coding theorem method by compressibility. *PeerJ Comput Sci* 2015;**1**:e23.
45. Antão R, Mota A, Machado JAT. Kolmogorov complexity as a data similarity metric: application in mitochondrial DNA. *Nonlinear Dyn* 2018;**93**(3):1059–71.
46. Faloutsos C, Megalooikonomou V. On data mining, compression, and Kolmogorov complexity. *Data Min Knowl Discov* 2007;**15**(1):3–20.
47. Hosseini M, Pratas D, Morgenstern B, et al. Supporting data for “Smash++: an alignment-free and memory-efficient tool to find genomic rearrangements.” *GigaScience Database* 2020. <http://dx.doi.org/10.5524/100741>.