GigaScience, 2017, 1–8

doi: xx.xxxx/xxxx
Manuscript in Preparation
Technical note

TECHNICAL NOTE

Smash++: finding rearrangements

Morteza Hosseini[1,*], Diogo Pratas[2,*] and Armando J. Pinho[2,*]

[1,2]IEETA/DETI, University of Aveiro, Portugal

[*]seyedmorteza@ua.pt; pratas@ua.pt; ap@ua.pt

Abstract

The Abstract (250 words maximum) should be structured to include the following details: **Background**, the context and purpose of the study; **Results**, the main findings; **Conclusions**, brief summary and potential implications. Please minimize the use of abbreviations and do not cite references in the abstract.

Key words: Keyword1; keyword 2; keyword 3 (Three to ten keywords representing the main content of the article)

Findings

Background

With ever-increasing development of high-throughput sequencing (HTS) technologies, a massive amount of genomic information is produced at much higher speed, better quality and lower cost than was possible before [?]. Analyses of such information has led to the advancement of our understanding of biology and disease, over the past decade [? ?]. 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 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, and mostly occur as an accident in the sperm or egg cell, and hence are present in every cell of the body [? ?].

Study on chromosomal aberrations, which underlie many genetic diseases and cancer, is crucial for diagnostics, prognostics and targeted therapeutics [?]. Examples of such diseases are Wolf–Hirschhorn syndrome (WHS), that is caused by a partial deletion from human chromosome location 4p16.3 [?], 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 [?], and acute myeloid leukemia (AML), that may be caused by translocations between human chromosome 8 and 21 [?].

We present an alignment-free method 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 [?]. The proposed method is implemented, under the name of Smash++, and is capable of finding and visualizing as SVG images informationally similar regions in two sequences.

Smash++ is an improved version of Smash [?], featuring (1) improved accuracy, that is using multiple context models to find fine-grained along with coarse-grained chromosomal rearrangements, (2) presenting complexity (redundancy) and relative redundancy of informationally similar regions in DNA sequences, (3) improved user interface (UI), both in command line and resulting SVG image, and (4) improved performance, in terms of time and memory usage.

In the following sections we first describe the proposed method in detail, including data modelling, finding similar regions and computing complexity. Then, we describe the datasets that are used in different experiments and how they can be accessed. Next, we provide details of the experiments carried out on the datasets along with findings based on their analysis. Source codes are publicly available under GNU GPL v3 license and also, the datasets are available on public repositories for download and use. Finally, we provide conclusions and discussion.

Compiled on: September 20, 2019.
Draft manuscript prepared by the author.

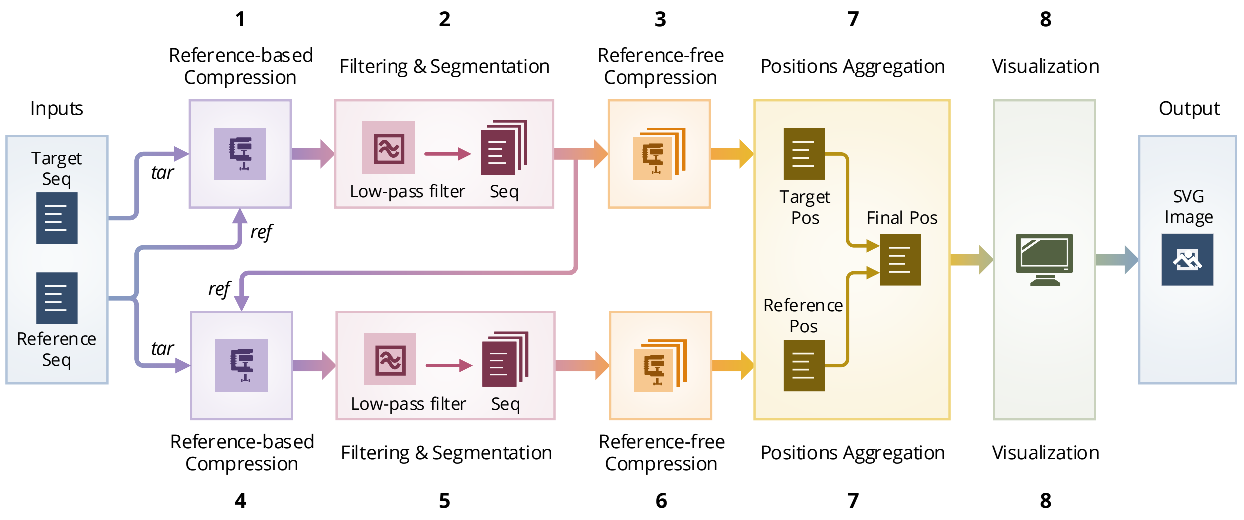


Figure 1. The schema of Smash++. The process of finding similar regions in reference and target sequences and also, computing redundancy in each region includes eight stages. Finally, Smash++ outputs a *.pos file that includes the positions of the similar regions, and can be then visualized, resulting in an SVG image.

Methods

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

Data modeling

Smash++ works on the basis of cooperation between finite-context models (FCMs) and substitutional tolerant Markov models (STMMs). Applying these models on various contexts provides probability and weight values, illustrated in Figure 2a, which are then mixed (by multiplication and addition, shown in Figure 2b) to provide the final probability (P) of occurring an input symbol. The following subsections describe FCMs and STMMs in detail.

Finite-context model (FCM)

A finite-context model considers Markov property to estimate the probability of the next symbol in an information source, based on the past k symbols (a context of size k) [? ? ?]. Denoting the context as $c_{k,i} = s_{i-k}s_{i-k+1}\ldots s_{i-2}s_{i-1}$, the probability of the next symbol s_i in an information source S, which is posed at i, can be estimated as

$$P_m(s_i|c_{k,i}) = \frac{N(s_i|c_{k,i}) + \alpha}{N(c_{k,i}) + \alpha|\Theta|}, \qquad (1)$$

in which m stands for model (FCM in this case), $N(s_i|c_{k,i})$ shows the number of times that the information source has generated symbol s_i in the past, $|\Theta|$ denotes size of the alphabet Θ, $N(c_{k,i}) = \sum_{b\in\Theta} N(b|c_{k,i})$ represents the total number of events occurred for the context $c_{k,i}$ and α allows to keep a balance

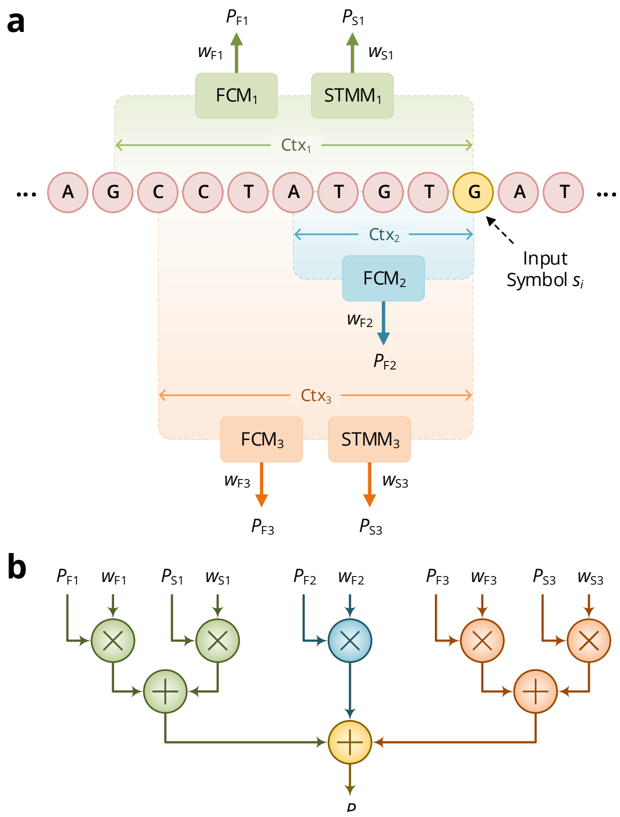


Figure 2. Data modelling by Smash++. (a) cooperation between finite-context models (FCMs) and substitutional-tolerant Markov models (STMMs). Note that each STMM needs to be associated with an FCM. (b) probability of an input symbol is estimated by employing the probability and weight values that have been obtained from processing previous symbols.

between the maximum likelihood estimator and the uniform distribution. Eq. 1 turns to the Laplace estimator, for $\alpha = 1$, and also behaves as a maximum likelihood estimator, for large number of events i [?].

Substitutional tolerant Markov model (STMM)

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

$$P_m(s_i|c'_{k,\,i}) = \frac{N(s_i|c'_{k,\,i}) + \alpha}{N(c'_{k,\,i}) + \alpha|\Theta|}, \qquad (2)$$

where N represents the number of occurrences of symbols, that is saved in memory, and $c'_{k,\,i}$ is a copy of the context $c_{k,\,i}$ which is modified as

$$c'_{k,\,i} = \underset{\forall b \in \Theta}{\arg\max}\, P_m(b|c'_{k,\,i}). \qquad (3)$$

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

This example shows the distinction between a finite-context model and a substitutional tolerant Markov model. Assume, the current context at position i is $c_{11,\,i}$ = GGCTAACGTAC, and the number of occurrences of symbols saved in memory is A = 10, C = 12, G = 13 and T = 11. Also, the symbol to appear in the sequence is T. An FCM would consider the next context as $c_{11,\,i+1}$ = GCTAACGTACT, while an STMM would consider it as $c'_{11,\,i+1}$ = GCTAACGTACG, since the base G is the most probable symbol, based on the number of occurrences stored in memory.

Cooperation of FCMs and STMMs

When FCMs and STMMs are in cooperation, the probability of the next symbol s_i in an information source S, at position i, can be estimated as

$$P(s_i) = \sum_{m \in M_F} P_m(s_i|c_{k,\,i})\, w_{m,\,i} + \sum_{m \in M_S} P_m(s_i|c'_{k,\,i})\, w'_{m,\,i},$$
$$\forall s_i \in S,\ 1 \le i \le |S|,\ 1 \le k \le i-1, \qquad (4)$$

in which M_F and M_S denote sets of FCMs and STMMs, respectively, $P_m(s_i|c_{k,\,i})$ shows the probability of the next symbol estimated by the FCM, $P_m(s_i|c'_{k,\,i})$ represents this probability estimated by the STMM, and $w_{m\,i}$ and $w'_{m,\,i}$ are weights assigned to each model based on its performance. We have

$$\forall m \in M_F: \quad w_{m,\,i} \propto (w_{m,\,i-1})^{\gamma_m} P_m(s_i|c_{k+1,\,i-1}),$$
$$\forall m \in M_S: \quad w'_{m,\,i} \propto (w'_{m,\,i-1})^{\gamma'_m} P_m(s_i|c'_{k+1,\,i-1}), \qquad (5)$$

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

$$\sum_{m \in M_F} w_{m,\,i} + \sum_{m \in M_S} w'_{m,\,i} = 1. \qquad (6)$$

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

Storing models in memory

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

- table of 64 bit counters, for $1 \le k \le 11$,
- 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 $k \ge 15$.

The table of 64 bit counters, that is shown in Figure 3a, simply saves number of events for each context. The table of 32 bit counters saves in each position the number of times that the associated context is observed. When a counter reaches to the maximum value $2^{32} - 1 = 4294967295$, all the counts will be renormalized by dividing by two, as shown in Figure 3b.

The approximate counting is a method that employs probabilistic techniques to count large number of events, while using small amount of memory [?]. Figure 4 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 3c 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 it in memory as $\log_2 8 = 3$.

The Count-Min-Log Sketch (CMLS) is a probabilistic data structure to save frequency of events in a table by means of a family of independent hash functions [?]. The algorithm for updating and querying the counter is shown in Figure 5. 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 3d.

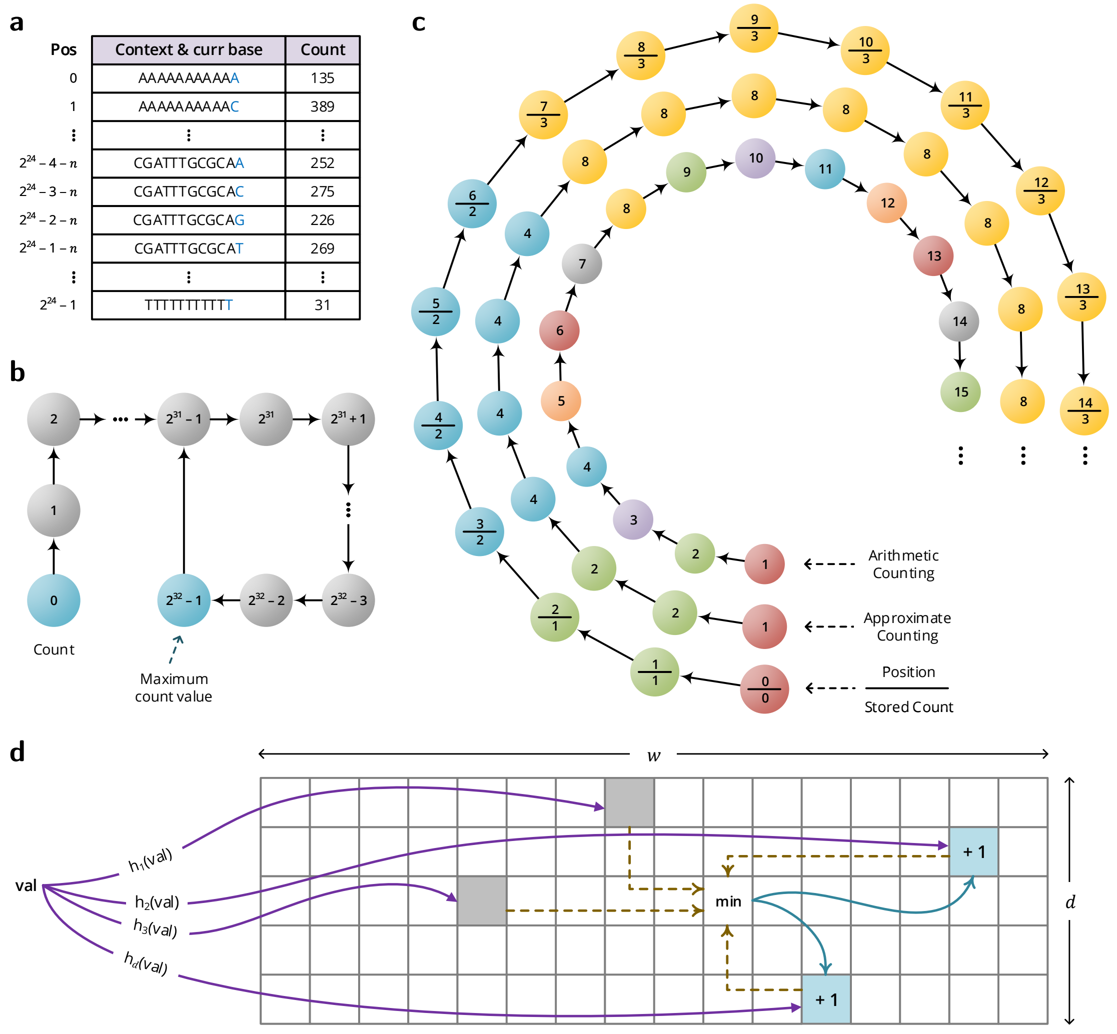
The CMLS requires a family of pairwise independent hash functions $H = \{h : U \to [m]\}$, in which each function h maps some universe U to m bins. To have this family, we use universal hashing by randomly selecting a hash function from a universal family in which $\forall x, y \in U,\ x \neq y:\ P_{h\in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

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

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

Finding similar regions

To find similar regions in reference and target files, a quantity is required for measuring the similarity. We use "per symbol information content" for this purpose, which can be calculated



Pos	Context & curr base	Count
0	AAAAAAAAAAA	135
1	AAAAAAAAAAC	389
⋮	⋮	⋮
$2^{24}-4-n$	CGATTTGCGCAA	252
$2^{24}-3-n$	CGATTTGCGCAC	275
$2^{24}-2-n$	CGATTTGCGCAG	226
$2^{24}-1-n$	CGATTTGCGCAT	269
⋮	⋮	⋮
$2^{24}-1$	TTTTTTTTTTT	31

Figure 3. 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:     $c \leftarrow$ table[$x$]
 6:     if IncreaseDecision($c$) = True then
 7:         table[$x$] $\leftarrow c + 1$
 8:     end if
 9: end function

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

Figure 4. Approximate counting update and query.

as

$$I(s_i) = -\log_2 P(s_i) \quad \text{bpb}, \quad \forall s_i \in S,\ 1 \leq i \leq |S| \qquad (8)$$

where $P(s_i)$ denotes the probability of the next symbol s_i in the information source S, obtained by Equation 4, and also "bpb" stands for bit per base.

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 [?], 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 6, 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 Hann window [?], 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 6a). The described phase repeats for all of the target regions found, in the way that

```
Input: 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 $a \not\equiv 0$

 1: function Hash($k, x$)                ▷ Universal hash family
 2:     return $((a_k x + b_k) \mod p) \mod m$
 3: end function

 4: function MinCount($x$)
 5:     minimum ← 15                      ▷ Biggest 4 bit number
 6:     for $k$ ← 1 to $d$ do
 7:         $h$ ← Hash($k, x$)
 8:         if sketch[$k$][$h$] < minimum then
 9:             minimum ← 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:     $c$ ← MinCount($x$)
19:     if IncreaseDecision($c$) = True then
20:         for $k$ ← 1 to $d$ do
21:             $h$ ← Hash($k, x$)
22:             if sketch[$k$][$h$] = $c$ then
23:                 sketch[$k$][$h$] ← $c + 1$
24:             end if
25:         end for
26:     end if
27: end function

28: function Query($x$)
29:     $c$ ← MinCount($x$)
30:     return $2^c - 1$
31: end function
```

Figure 5. Count-Min-Log Sketch update and query.

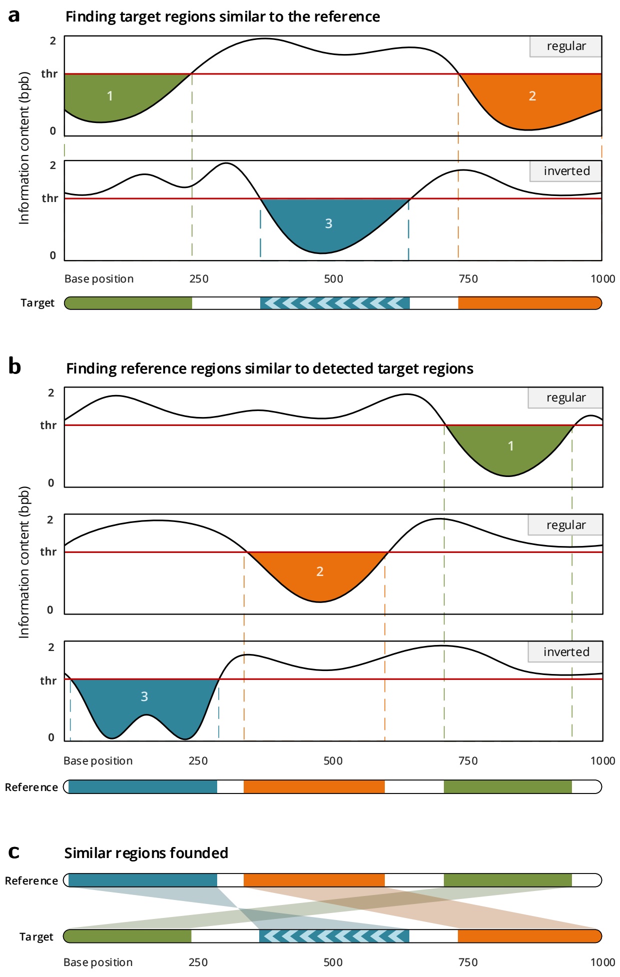


Figure 6. Finding similar regions in reference and target sequences. Smash++ finds, first, the regions in the target that are similar to the reference, and then, finds the regions in the reference that are similar to the detected target regions. This procedure is performance for both regular and inverted homologies.

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 6b). The final result would have the form of Figure 6c.

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 [?]. 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 [? ?].

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 [? ? ?]. 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.

Experiments setup

Datasets

Results

Smash++ and several other methods have been carried out on a collection of synthetic and real sequences. The machine used for the tests had a 4-core 3.40 GHz Intel® Core™ i7-6700 CPU and 32 GB of RAM.

Figure 9 conforms to [?]

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

Table 1. Dataset used in the experiments.

Sequence	Length (base)	Accession	Group	Description
Real data				
PXO99A	5,238,555	CP000967	Bacteria	*Xanthomonas oryzae* pv. *oryzae* causes the major disease of bacterial blight of rice (*Oryza sativa L.*). *Xanthomonas oryzae* pv. *oryzae* PXO99A strain is virulent toward a large number of rice varieties representing diverse genetic sources of resistance [?].
MAFF 311018	4,940,217	AP008229	Bacteria	*Xanthomonas oryzae* pv. *oryzae* MAFF 311018 is a Japanese race 1 strain [?].
GGA18	11,373,140	CM000110	Aves	*Gallus gallus* chromosome 18.
MGA20	10,730,484	CM000981	Aves	*Meleagris gallopavo* isolate NT-WF06-2002-E0010 breed Aviagen turkey brand Nicholas breeding stock chromosome 20.
GGA14	16,219,308	CM000106	Aves	*Gallus gallus* chromosome 14.
MGA16	14,878,991	CM000977	Aves	*Meleagris gallopavo* isolate NT-WF06-2002-E0010 breed Aviagen turkey brand Nicholas breeding stock chromosome 16.
ScVII	1,090,940	NC_001139	Fungi	*Saccharomyces cerevisiae* S288C chromosome VII.
SpVII	1,105,967	CP020299	Fungi	*Saccharomyces paradoxus* strain UFRJ50816 chromosome VII.
Synthetic data				
RefS	1,500	–	–	It consists of three segments of 500 base size.
TarS	1,500	–	–	To build TarS, segment I is mutated 2%, II is inversely repeated and III is duplicated.
RefM	100,000	–	–	It has four segments of 25 kilobase size.
TarM	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	5,000,000	–	–	It includes two segments, 2,500,000 bases each.
TarL	5,000,000	–	–	Segment I is inversely repeated and II is mutated 2% for building TarL.
RefXL	100,000,000	–	–	It is made of four segments, 25,000,000 bases each.
TarXL	100,000,000	–	–	Segment I is mutated 1%, segments II and III are inversely repeated and segment IV is duplicated to make TarXL.
RefMut	60,000	–	–	It includes 60 segments of 1 kilobase size.
TarMut	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	1,000,000	–	–	It consists of 10 segments of 100 kilobases.
TarComp	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.

- Other requirements: C++ 11 or higher
- License: GNU GPL v3

Availability of supporting data and materials

GigaScience requires authors to deposit the data set(s) supporting the results reported in submitted manuscripts in a publicly-accessible data repository such as *GigaDB* (see *Giga*DB database terms of use for complete details). This section should be included when supporting data are available and must include the name of the repository and the permanent identifier or accession number and persistent hyperlinks for the data sets (if appropriate). The following format is recommended:

"The data set(s) supporting the results of this article is(are) available in the [repository name] repository, [cite unique persistent identifier]."

Following the Joint Declaration of Data Citation Principles, where appropriate we ask that the data sets be cited where it is first mentioned in the manuscript, and included in the reference list. If a DOI has been issued to a dataset please always cite it using the DOI rather than the less stable URL the DOI resolves to (e.g. http://dx.doi.org/10.5524/100044 rather than http://gigadb.org/dataset/100044). For more see:

Data Citation Synthesis Group: Joint Declaration of Data Citation Principles. Martone M. (ed.) San Diego CA: FORCE11; 2014 [https://www.force11.org/datacitation]

A list of available scientific research data repositories can be found in res3data and BioSharing.

Declarations

List of abbreviations

AML: acute myeloid leukemia; CMT: Charcot-Marie-Tooth; CMLS: Count-Min-Log Sketch; CPU: central processing unit; FCM: finite-context model; GB: gigabyte; GGA: *Gallus gallus*; GHz: gigahertz; HTS: high-throughput sequencing; KB: kilobyte; MB: megabyte; MGA: *Meleagris gallopavo*; RAM: random access memory; PMP22: peripheral myelin protein 22; PRNG: pseudo-random number generator; STMM: substitutional tolerant Markov model; SVG: Scalable Vector Graphics; UI: user interface; WHS: Wolf-Hirschhorn syndrome.

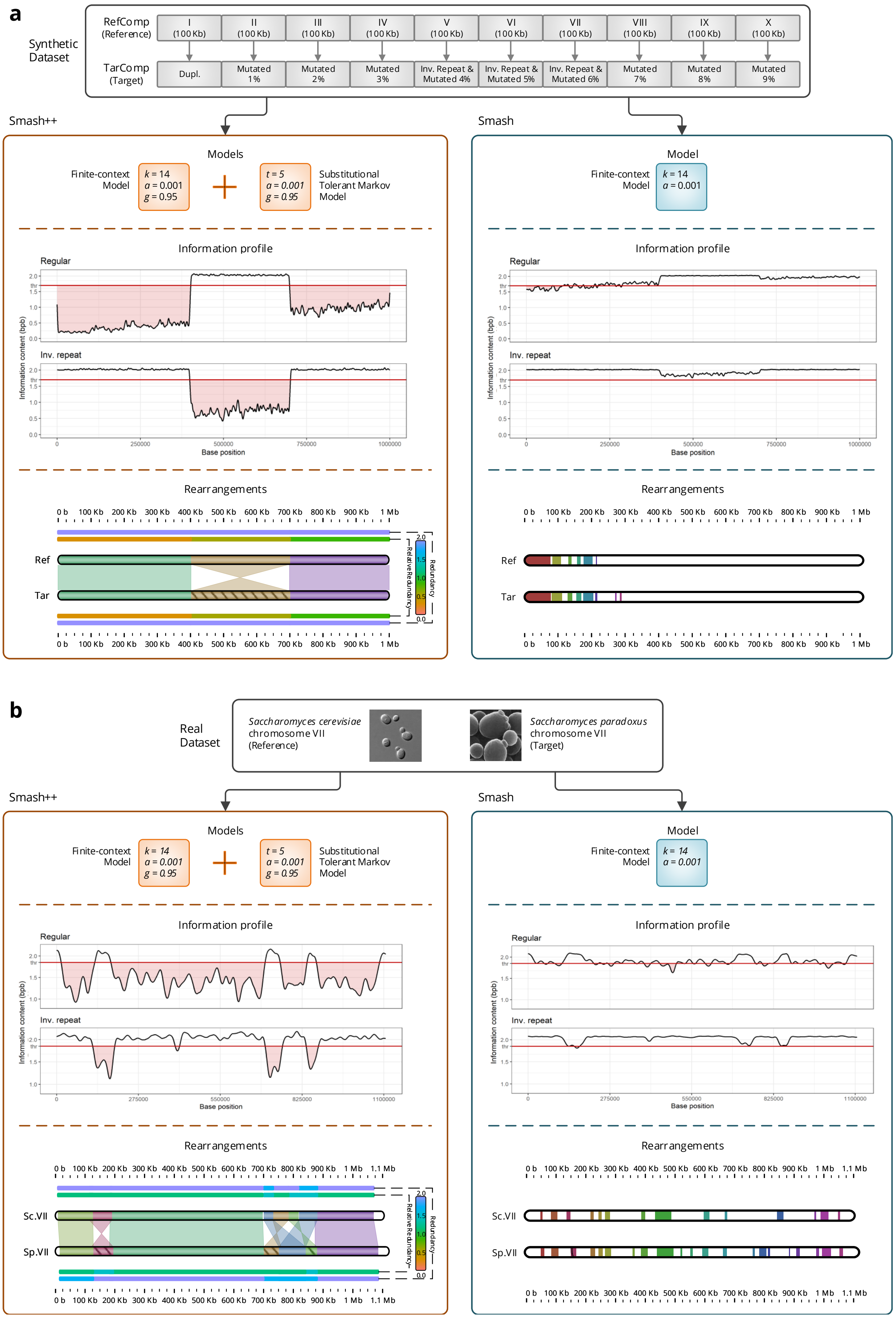


Figure 7. Compare Smash.

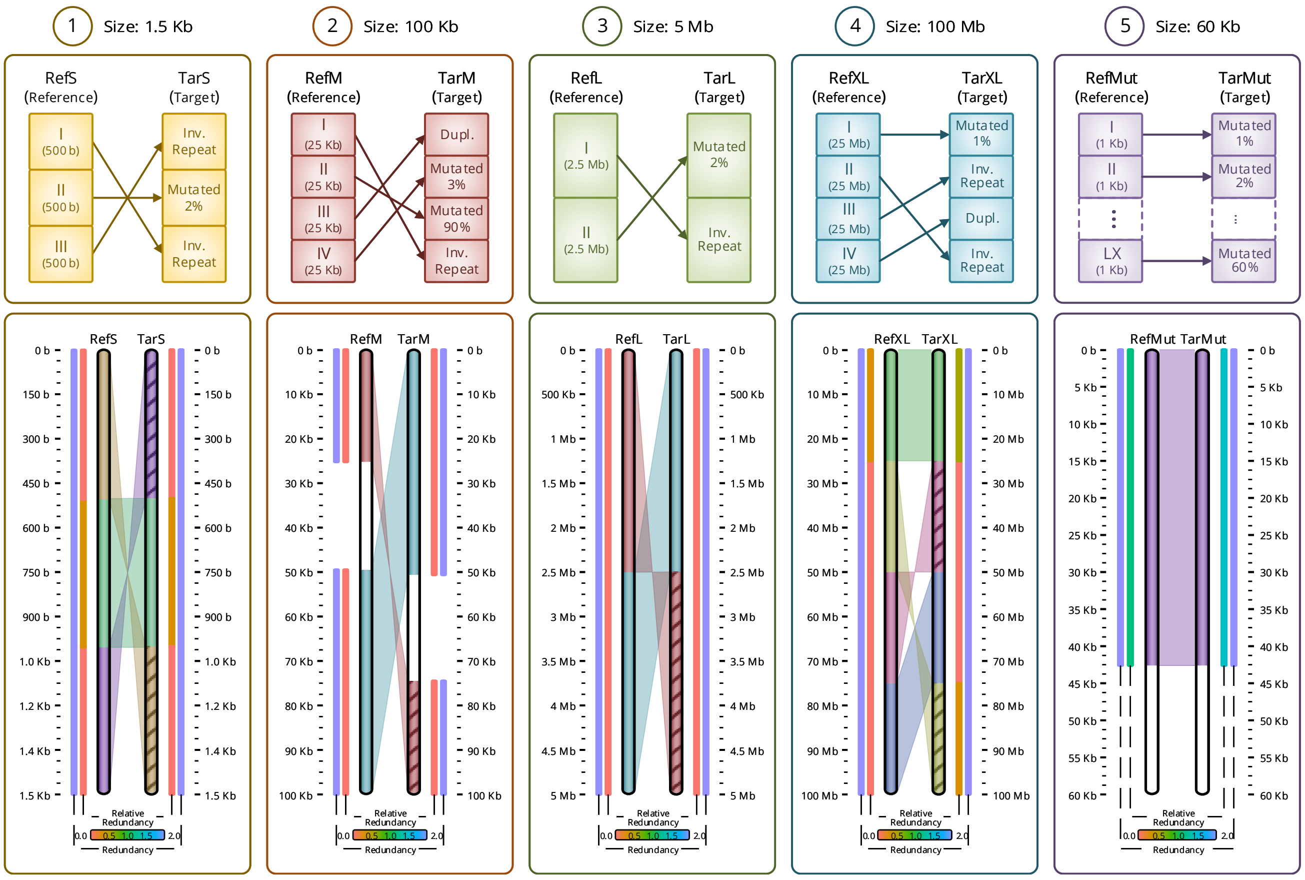


Figure 8. synthetic.

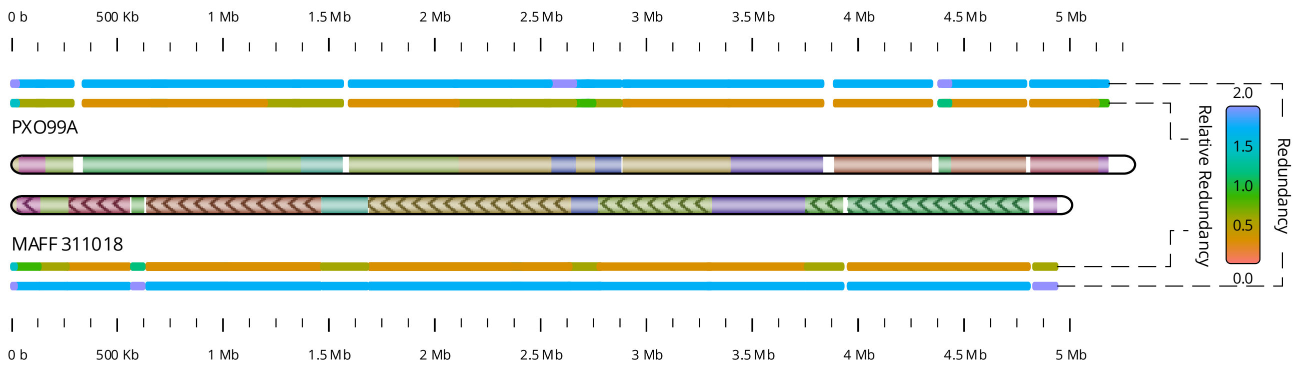


Figure 9. PXO99A - MAFF 311018.

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/00127/2013, PTCD/EEI-SII/6608/2014] and the grant [PD/BD/113969/2015].

Author’s Contributions

The individual contributions of authors to the manuscript should be specified in this section. Guidance and criteria for authorship can be found in our editorial policies. We would recommend you follow some kind of standardised taxonomy like the CASRAI CRediT (Contributor Roles Taxonomy).