Bioinformatics, 34, 2018, i706–i714
doi: 10.1093/bioinformatics/bty586
ECCB 2018

OXFORD

Fast characterization of segmental duplications in genome assemblies

Ibrahim Numanagić[1,2], Alim S. Gökkaya[3], Lillian Zhang[1], Bonnie Berger[1,2], Can Alkan[3,*] and Faraz Hach[4,5,*]

[1]Computer Science and Artificial Intelligence Laboratory and [2]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, [3]Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey, [4]Vancouver Prostate Centre, Vancouver, BC V6H 3Z9, Canada and [5]Department of Urologic Sciences, University of British Columbia, Vancouver, BC V5Z 1M9, Canada

*To whom correspondence should be addressed.

Abstract

Motivation: Segmental duplications (SDs) or low-copy repeats, are segments of DNA $>$ 1 Kbp with high sequence identity that are copied to other regions of the genome. SDs are among the most important sources of evolution, a common cause of genomic structural variation and several are associated with diseases of genomic origin including schizophrenia and autism. Despite their functional importance, SDs present one of the major hurdles for *de novo* genome assembly due to the ambiguity they cause in building and traversing both state-of-the-art overlap-layout-consensus and de Bruijn graphs. This causes SD regions to be misassembled, collapsed into a unique representation, or completely missing from assembled reference genomes for various organisms. In turn, this missing or incorrect information limits our ability to fully understand the evolution and the architecture of the genomes. Despite the essential need to accurately characterize SDs in assemblies, there has been only one tool that was developed for this purpose, called Whole-Genome Assembly Comparison (WGAC); its primary goal is SD detection. WGAC is comprised of several steps that employ different tools and custom scripts, which makes this strategy difficult and time consuming to use. Thus there is still a need for algorithms to characterize within-assembly SDs quickly, accurately, and in a user friendly manner.

Results: Here we introduce SEgmental Duplication Evaluation Framework (SEDEF) to rapidly detect SDs through sophisticated filtering strategies based on Jaccard similarity and local chaining. We show that SEDEF accurately detects SDs while maintaining substantial speed up over WGAC that translates into practical run times of minutes instead of weeks. Notably, our algorithm captures up to 25% 'pairwise error' between segments, whereas previous studies focused on only 10%, allowing us to more deeply track the evolutionary history of the genome.

Availability and implementation: SEDEF is available at https://github.com/vpc-ccg/sedef.

Contact: calkan@cs.bilkent.edu.tr or faraz.hach@ubc.ca

1 Introduction

Segmental duplications (SDs) are defined as genomic segments of size >1 Kbp that are repeated within the genome with at least 90% sequence identity (Bailey *et al.*, 2001) in either tandem or interspersed organization. Almost all genomes harbor large SDs; e.g. the build 37 release of the human reference genome (GRCh37) contains a total of 159 Mbp gene-rich duplicated sequence, which corresponds to ∼5.5% of its entire length (http://humanparalogy.gs.washington.edu/build37/build37.htm). It is known that SDs played a major role in evolution (Marques-Bonet *et al.*, 2009; Prado-Martinez *et al.*, 2013; Sudmant *et al.*, 2013), and are one of the most important factors that contribute to human disease either directly (Gonzalez *et al.*, 2005; Hollox *et al.*, 2008; Yang *et al.*, 2007), or through leading to other forms of structural variation (SV) (Alkan *et al.*, 2011a; Mills *et al.*, 2011). Furthermore, human populations show SD diversity that may be used as markers for population genetics studies (Alkan *et al.*, 2009; Sudmant *et al.*, 2010).

Despite their functional importance, SDs are poorly characterized due to the difficulties they impose on constructing accurate genome assemblies (Alkan *et al.*, 2011b; Chaisson *et al.*, 2015;

Steinberg *et al.*, 2017), as well as the ambiguities in read mapping (Treangen and Salzberg, 2011; Firtina and Alkan, 2016). Building *inaccurate* assemblies due to SDs is an important problem, since it may lead to potentially incorrect conclusions about the evolution of the species of interest as in the case of the giant panda assembly (Li *et al.*, 2010). In the giant panda genome analysis, the authors concluded that the panda genome included substantially less repeats and duplications compared with other mammalian genomes; however, it is shown that this result is likely incorrect due to misassembly of these regions. Additionally, duplications give rise to gaps in the assembly, which likely contain genes and other functionally active regions (Alkan *et al.*, 2011b; Bailey *et al.*, 2001, 2002).

Accurate assembly of duplicated regions remains a difficult and unsolved problem, which may be ameliorated through the use of ultra-long reads generated by the Oxford Nanopore platform (Jain *et al.*, 2018) or Linked-Read sequencing (e.g. 10x Genomics; Mostovoy *et al.*, 2016; Yeo *et al.*, 2018) if the duplicated segment is shorter than the read or linked-read length, respectively. However, characterization of the SD content in existing assemblies is still important for two reasons: (i) to evaluate the 'completeness' of these genome assemblies and (ii) understand genome evolution for comparative genomics studies.

SD content in assemblies can be assessed using two strategies, and the overlap between the results of the two methods determines the completeness of the assembly in terms of duplications. The first method, called Whole-Genome Assembly Comparison (WGAC), relies on the alignment of the entire genome to itself to identify repeating segments (Bailey *et al.*, 2001) within the assembly, except for common repeats which are filtered out. The second strategy is called Whole-genome Shotgun Sequence Detection (WSSD) which relies on the read depth sequence signature (Bailey *et al.*, 2002). Briefly, the WSSD method aligns the original reads back to the assembled genome, and looks for regions of read depth significantly higher than the average, which signals a putative duplication (Bailey *et al.*, 2002). Regions that are marked as SDs by WSSD, but not by WGAC, are then classified as likely collapsed duplications (Alkan *et al.*, 2011b).

Although the 'optimal' alignment of the entire genome to itself can be theoretically computed via standard dynamic programing (e.g. Smith-Waterman algorithm), such an approach remains impractical due to quadratic time and memory complexity, and is likely to remain so (Backurs and Indyk, 2015). Furthermore, high edit distance between the SD paralogs disqualifies the use of most of the available edit distance approximations with theoretical guarantees (Andoni *et al.*, 2010; Hanada *et al.*, 2017), as well as the majority of the sequence search tools that operate under the assumption that the similarity between regions is high. Standard whole-genome or long-read aligners [e.g. MUMmer (Marçais *et al.*, 2018) or Minimap2 (Li, 2018)] are not able to efficiently capture SDs with low similarity rates (i.e. <80%), and also confuse SDs with other repeated elements in the genome (e.g. stretches of short tandem repeats). For these reasons, the WGAC method is composed of a number of heuristics that include several tools and scripts (Bailey *et al.*, 2001). First, the common repeats are removed from the assembly in a step called *fuguization*. Remaining regions are partitioned into 400 Kb chunks (due to memory limitations when WGAC was developed), all pairwise alignments are computed using BLAST (Altschul *et al.*, 1990), and significant alignments are kept as putative duplications. A modified version of BLASTZ (Schwartz *et al.*, 2003) is also used to find within-chunk (i.e. 400 Kb segments) duplications. Next, common repeats are inserted back, spurious alignments at the end of the sequences are trimmed, and final global alignments are calculated. Note that another method, SDquest (Pu *et al.*, 2018), was published while this article was under review.

The original WGAC implementation as outlined above is difficult to run, and it is time consuming as it relies on several general purpose tools (such as BLAST) and custom Perl scripts. In its current form, the only way to accelerate the WGAC analysis is using a compute cluster to parallelize BLAST alignments. Interestingly, another problem is the modified version of BLASTZ for self alignments: the source code is not available, and only a binary compiled for the Sun Solaris operating system has been released (http://humanparalogy.gs.washington.edu/code/WGAC_HOWTO.pdf), rendering the tool unusable for most other researchers. We note that this self-alignment step might be replaced with another tool such as LASTZ (Harris, 2007); however, the parameter settings for the current release is not yet optimized for alternative aligners.

Here we introduce a new algorithm to characterize SDs in genome assemblies. Although we follow a strategy akin to WGAC in aligning a whole genome to itself; we do so in a more efficient way by introducing sophisticated optimizations to both the putative SD detection and global alignment steps. We leverage our knowledge from the biology of human and other genomes that different mutation events contribute unequally to the total value of the error rate (quantification of differences) between segments, in order to better optimize our SD detection algorithms. A key conceptual advance of our work that helps model such events in the genome is to separately consider germline mutation rates (denoted as *small mutations*), and larger-scale *de novo* SV rates. This formulation enables us to speed up SD detection and better capture evolutionary events. We implement our algorithms in C++ and provide a single package called SEDEF (SEgmental Duplication Evaluation Framework). In contrast to WGAC, which requires several weeks to complete even on a compute cluster, SEDEF can characterize SDs in the human genome in 10 CPU h. We believe SEDEF will be a powerful tool to characterize SDs for both genome assembly evaluation and comparative genomics studies.

2 Preliminaries

SDs are generated by large-scale copy events that have occurred during the evolution of the genome. After such a copy event, both sites involved in SD may have undergone a number of changes during the evolutionary history of the genome. Formally, consider a genomic sequence $G = g_1 g_2 g_3 \dots g_{|G|}$ of length $|G|$, where $g_i \in \{A, C, G, T\}$ for any i. Let $G_{i:i+n} = g_i, \dots, g_{i+n-1}$ be a substring in G of length n that starts at position i. Furthermore, let X_i be the set of all k-mers in the substring $G_{i:i+n}$. We assume that k is predefined and fixed.

The Levenshtein (Levenshtein, 1966) *edit distance* is defined as: $\mathrm{ed}(G_{i:i+n}, G_{j:j+m})$ of two substrings $G_{i:i+n}$ and $G_{j:j+m}$ (further simplified as G_i and G_j) to be a minimal number of edit operations (i.e. single nucleotide substitutions, insertions and deletions) that are needed to convert the string G_i into G_j. The length of the alignment between G_i and G_j is denoted as l, and clearly $l \geq \max(m, n)$. Let us define the notion of *edit error* (further referred to as just *error*) between two strings G_i and G_j as $\mathrm{err}(G_i, G_j) = \mathrm{ed}(G_i, G_j)/l$—the edit distance normalized over alignment length. Intuitively, this is the average number of the edits needed to turn G_i into G_j. Clearly, two strings are identical if $\mathrm{err}(G_i, G_j) = 0$. We consider G_i and G_j as a *SD* of length l with error δ if the following *SD conditions* are met:

- $l \geq 1000$ where l is the length of alignment between G_i and G_j,
- $\mathrm{err}(G_i, G_j) \leq \delta$.

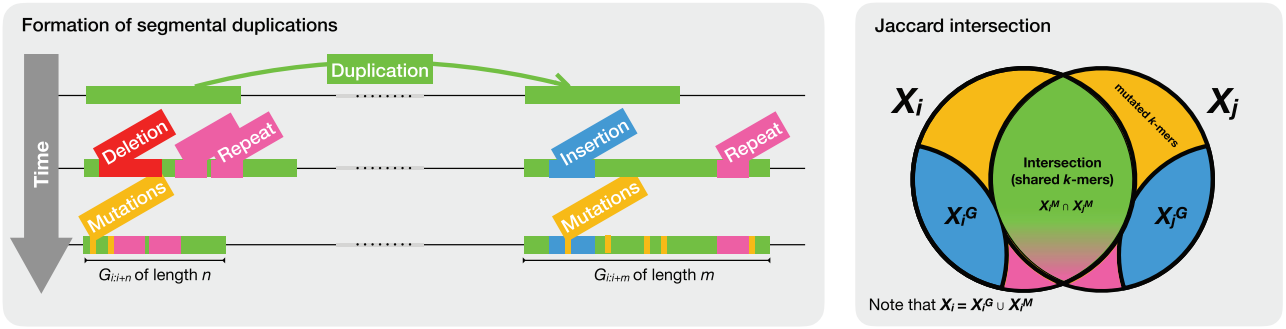

Fig. 1. (Left) Simplified representation of a SD lifetime. Initially, a large-scale duplication forms an SD, at which point both the original region and the copy are identical. Then, both the original region and copy undergo various independent changes, such as large-scale deletions (in red), insertions (in blue), and small repeat insertions (in fuchsia). Finally, various germline mutations (in yellow) affect both regions. The resulting SD as seen today, defined as the pair $(G_{i:i+n}, G_{j:j+m})$, is shown in the third row. **(Right)** Shows the idealized Jaccard similarity between the *k*-mer sets X_i and X_j corresponding to the G_i and G_j, respectively. Note that some repeats also increase the proportion of shared *k*-mers. Colors denote same as on Left

We also assume that the overlap between the G_i and G_j in the genome is at most $\delta \cdot n$.

2.1 Edit distance model

Each SD is generated by a past SV event that copied a substring of length n in G from locus i to locus j. This copy was initially perfect, meaning that corresponding strings G_i and G_j (initially both of length n) were identical. However, various changes during the evolutionary history of the genome—such as point mutations, small indels, and other structural variants—have altered both the original and duplicate strings independently. Thus it is necessary to take such changes into consideration when identifying the potential SDs. Although previous SD studies focused only on SDs with pairwise error at most 10%, here we aim to focus on SDs whose error rate can go up to 25% (in another words, $\delta \leq 1/4$). Higher δ allows us to track the evolutionary history of the human genome to earlier periods. However, it also significantly renders the SD detection problem more difficult since the majority of the known filtering techniques that operate on the edit distance metric space assume much lower values of δ (Andoni *et al.*, 2010). We address this challenge by leveraging our knowledge from the biology of human and other genomes that different mutation events contribute unequally to the total value of δ in order to better optimize our SD detection algorithms.

A key conceptual advance of our work that helps model such events in the human genome is to separately consider germline mutation rates (denoted as *small mutations*), and *de novo* SV rates. It is estimated that the substitution rate in the human genome is roughly 0.5×10^{-9} per basepair per year (Scally, 2016), and we may assume a similar rate for other mammalian genomes. The evolutionary split of the human and chimpanzee species is estimated to have occurred ~7 million years ago (Hedges and Kumar, 2009). Thus, we expect that the probability of a basepair being mutated since the split is roughly 3.5×10^{-3}. If we also account for small indels [with an even smaller mutation rate than those of substitutions (Montgomery *et al.*, 2013)], the edit error between any two paralogs of an SD that have occurred after the evolutionary split is not >0.1%. Even if we consider SDs that occurred much earlier in history (e.g. after the lowest common ancestor of human and mouse roughly 90 million years ago), the total edit error will not be >10%. However, the edit error between two paralogs of an SD can be much larger due to large SVs. One such example is insertion of transposons within an SD. These events can be visualized as large gaps within an edit distance string of two SDs which contribute a large share towards the total edit error (Fig. 1) (We note that inversion and translocation events contribute more to sequence divergence due to incorrect alignments, but such events are rare). Thus we assume that small mutations contribute at most $\delta_M \leq 0.15$ towards the total edit error δ (both paralogs can be mutated up to 7.5%); this default setting is higher than the estimated rates (above) for human and mouse genomes in order to be able to handle older species as well. Analogously, large-scale events (subsequently referred to as *gaps*) contribute the remaining $\delta_G = \delta - \delta_M$ of the edit error. We assume that the probability of a large gap occurring at any basepair in the genome is not >0.005 (as estimated by analysis of existing human SDs; similar value can be derived for other species). Note that the gap penalty is typically calculated via affine gap model, where gap openings are heavily penalized while the gap extensions are either ignored or assigned a very low penalty. Many human SDs have $\delta_G \gg 0.15$ if calculated by standard Levenshtein distance metric. SEDEF uses standard (Levenshtein) gap distance metric while locating seed SDs (meaning all seed SDs have $\delta_G \leq 0.15$); however, this restriction is lifted in a later step where we switch to the affine gap penalty.

Furthermore, we assume that the mutations within SD paralogs follow a Poisson error model (Jain *et al.*, 2017; Fan *et al.*, 2015), and that those mutations occur independently of each other. It follows that any *k*-mer in X_j (the set of *k*-mers of G_j) has accumulated on average $k \cdot \delta_M$ mutations compared with the originating *k*-mer in X_i provided that such a *k*-mer was part of the original copy. By setting a Poisson parameter $\lambda = k \cdot \delta_M$, we obtain the probability of an event in which a *k*-mer is preserved in both paralogs of an SD (i.e. that it is error-free): $\Pr(\#\text{mutations} = 0|\lambda) = e^{-k\delta_M}$.

We call this error model *SD error model*, and assume that any SD of interest satisfies the error constraints mentioned above.

2.2 Jaccard similarity

Suppose that we ask whether two substrings G_i and G_j are similar to each other, where the length of both strings is n. One way to measure the similarity of those substrings is to analyze their respective *k*-mer sets X_i and X_j and to count the number of shared *k*-mers between them. This metric is known as *Jaccard similarity* of sets X_i and X_j, and is formally defined as

$$J(X_i, X_j) = \frac{|X_i \cap X_j|}{|X_i \cup X_j|}.$$

Clearly, higher similarity of strings G_i and G_j implies a larger value of $J(X_i, X_j)$.

The calculation of Jaccard similarity between two sets can be approximated via the MinHash technique developed by

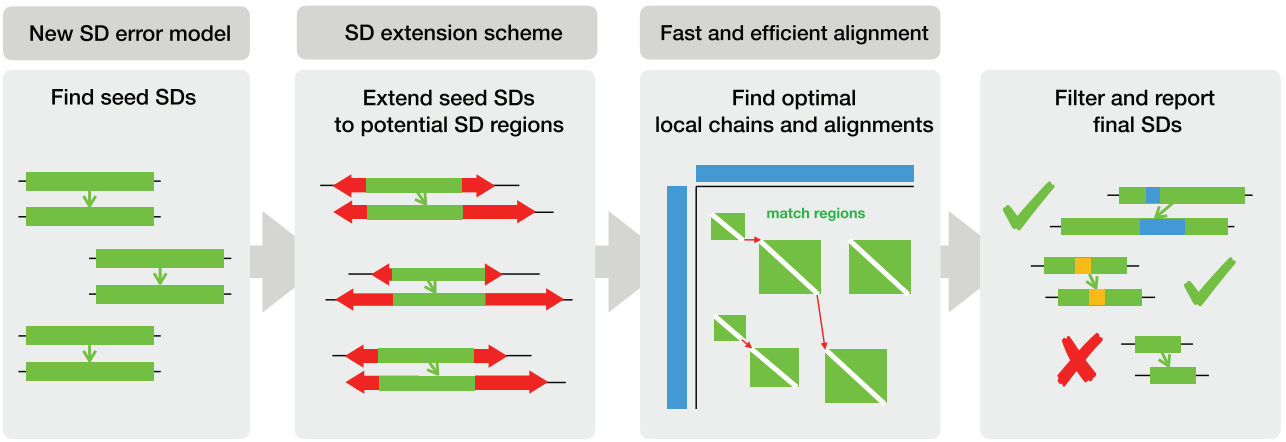

Fig. 2. Step-by-step depiction of the SEDEF framework. Our contribution is highlighted above the steps in the dark gray boxes

Broder (1997), who proved that given a universe of all k-mers U and a random permutation h on U (typically a hash function with no collisions), it follows that $J(X_i, X_j) = \Pr[h_{\min}(X_i) = h_{\min}(X_j)]$ where $h_{\min}(T)$ is the minimal member of T with respect to h. Furthermore, $J(X_i, Y_j)$ can be estimated and efficiently computed by calculating

$$\frac{|S(X_i \cup X_j) \cap S(X_i) \cap S(X_j)|}{|S(X_i \cup X_j)|}$$

where $S(X_i)$ is the *sketch of* X_i and stands for a subset of s elements from X_i whose hash values are minimal with respect to the hash function h (such elements are called *minimizers* of set X_i). This estimate is unbiased as long as h is random, and its accuracy depends on the sketch size s. Since in practice s is much smaller than $|X_i|$, calculating a MinHash estimate is substantially faster compared with the calculation of $J(X_i, X_j)$.

The performance of the MinHash technique can be further improved in the context of large strings, as shown by Jain *et al.* (2018). Instead of computing $S(X_i)$, it is possible to compute $S(W(X_i))$, where $W(X_i)$ is a *winnowing fingerprint* of the corresponding string G_i. $W(X_i)$ is calculated by sliding a window of size w through G_i and by taking in each window a k-mer of minimal hash value (in case of a tie, the rightmost k-mer is selected). The expected size of $W(A)$ for a random sequence A is $2|A|/(w+1)$ (Schleimer *et al.*, 2003). The main benefit of winnowing, aside from speeding up the construction of sketch $S(A)$, is the fact that winnow $W(A)$ can be computed efficiently in linear time and $O(w)$ space in a streaming fashion with appropriate data structures (Carruthers-Smith, 2013).

Moreover, it has been empirically shown that $J(X_i, X_j)$ can be efficiently estimated by calculating a *winnowed MinHash* score of sets X_i and X_j (Jain *et al.*, 2017):

$$\frac{|S[W(X_i) \cup W(X_j)] \cap S[W(X_i)] \cap S[W(X_j)]|}{|S[W(X_i) \cup W(X_j)]|}.$$

Furthermore, given the minimal desired value τ of Jaccard similarity between the two sets, it follows that $|W(X_i) \cap W(X_j)| \geq s \cdot \tau$ (Jain *et al.*, 2017). This estimate can be used to efficiently filter out any sets X_i and X_j whose Jaccard similarity is below a given threshold with high confidence.

3 Materials and methods

The SD detection problem can be formulated as follows: find all pairs of loci (i, j) inside a genome G with $l \geq 1000$ such that (i) the edit error (i.e. divergence) between G_i and G_j is at most δ; and (ii) the corresponding alignment between G_i and G_j is of size at least l and is not contained within a larger alignment satisfying the SD criteria. In other words, for any pair (i, j) we aim to find a maximal valid region alignment of size l between G_i and G_j that satisfies the criteria and error model of SDs.

A naïve method for locating SDs within any genomic sequence G consists of locally aligning G onto itself, followed by the analysis of all acceptable paths within a local alignment matrix. However, this strategy is impractical for large $|G|$ since the best known algorithms for optimal local alignment require $O(|G|^2)$ time and space. Another possible approach, which we take, is to solve this problem by iterating through each pair of indices (i, j) within G and testing whether the matching G_i and G_j satisfy the SD criteria through global alignment (given a fixed size n of G_i and G_j). Although this method, if implemented naïvely, is still too slow for larger genomes and requires quadratic space, it can be significantly accelerated by filtering out any pair (i, j) that is unlikely to form an SD. This iterative approach is the cornerstone of our SD detection framework, SEDEF, which consists of a novel seed and extend algorithm (Fig. 2):

- **SD seeding:** Initially, we aim to find all pairs of strings (G_i, G_j)—called *seed SD*—such that the length of both strings is $n \geq 1000 \cdot (1 - \delta) = 750$, and such that G_i and G_j are believed to satisfy the SD criteria. We achieve this by iterating through the genome, and for each locus i in the genome rapidly enumerating all feasible pairs j for which winnowed MinHash Jaccard similarity between G_i and G_j goes over a pre-defined threshold τ.
- **SD extension:** Here we relax the condition that both G_i and G_j have the same size n, and keep expanding both seed regions G_i and G_j until the winnowed MinHash estimate drops below τ. These enlarged seed SDs are called *potential SD regions*. We terminate this extension when we either reach the maximal allowed value of SD, or if the extension causes G_i and G_j to significantly overlap.
- **SD chaining:** Finally, we locate all 'true' SDs within any potential SD region and calculate their alignments by locally aligning potential SD regions via local chaining and sparse dynamic programing. Afterwards, we filter out any spurious hits and report the remaining SDs.

3.1 Identifying seed SDs

In order to verify if the strings G_i and G_j have edit error $\leq \delta$ under the SD error model, we will calculate the Jaccard similarity of their corresponding k-mer sets X_i and X_j and check if it is $\geq \tau$. For the

sake of explanation, we will assume that $n = |G_i| = |G_j|$ (analogous reasoning holds if this is not the case). If c is $|X_i \cap X_j|$ (number of shared k-mers), and $t = n - k + 1$ (the size of sets X_i and X_j), we can express the Jaccard similarity of those sets as $J(X_i, X_j) = c/(2t - c)$. We will also assume that no k-mer occurs twice in these sets; this assumption is sufficient for the calculation of lower bound below.

Simply using error δ to calculate the expected lower bound of Jaccard similarity τ is infeasible in practice due to the large value of δ in our setting. However, as noted earlier, differences between duplicated regions are not chosen uniformly at random, and thus we can separate the error δ into the $\delta_M + \delta_G$, where δ_M is the error rate of the small mutations and δ_G is the error rate of large indels, as defined in the preliminaries. This separation of the two error rates is one of the novel contributions of our work.

If there exists a valid SD spanning G_i and G_j, set X_j can be considered as a union of two disjoint sets X_j^G and X_j^M, where X_j^M represents the k-mers initially copied by the SD event and might have undergone small mutations, while X_j^G contains all 'new' k-mers introduced by subsequent large events such as SVs and large indels (analogous separation applies to X_i as well). In this way we can separate the effects of the small mutations and large-scale events. Ideally X_j^G shares no k-mers with X_i, while $X_j^M \cap X_i^M \neq \emptyset$ since we expect some shared k-mers to remain error-free after small mutations (see Fig. 1 for visualization). Now let us also express t as $t_M + t_G$, where $t_M = |X_j^M|$ and $t_G = |X_j^G|$ (note that $|X_i| = |X_j|$ implies $|X_i^G| \approx |X_j^G|$ because the small mutations keep strings that generate $|X_i^M|$ and $|X_j^M|$ similar in size; thus we assume w.l.o.g that $|X_i^G| = |X_j^G|$).

Let c/t_M be the ratio of k-mers that are not mutated in both X_i^M and X_j^M (we assume that X_i^G and X_j^G share no common k-mers, which is a valid assumption for a lower bound calculation). Its expected value, provided a Poisson error model introduced above, is $\mathbb{E}[c/t_M] = e^{-k\delta_M}$ (Jain *et al.*, 2017).

Now we proceed to estimate the minimal required Jaccard similarity $J(X_i, X_j)$ of X_i and X_j. Note that so far:

1. $|X_i \cap X_j| = |X_i^M \cap X_j^M|$;
2. $t_G/(t_M + t_G) \leq \delta_G \Rightarrow t_G \leq t_M \cdot \delta_G/(1 - \delta_G)$; and
3. $|X_i^G \cup X_j^G| \leq 2|X_j^G| = 2t_G$ because $|X_i^G| = |X_j^G|$ (equality holds for the ideal condition where $|X_i^G \cap X_j^G| = \emptyset$).

It follows that:

$$J(X_i, X_j) = \frac{|X_i \cap X_j|}{|X_i \cup X_j|} = \frac{|X_i^M \cap X_j^M|}{|X_i^M \cup X_j^M| + |X_i^G \cup X_j^G|} \quad \text{(by 1)}$$

$$\geq \frac{|X_i^M \cap X_j^M|}{|X_i^M \cup X_j^M| + 2t_G} \quad \text{(by 2)}$$

$$\geq \frac{|X_i^M \cap X_j^M|}{|X_i^M \cup X_j^M| + \frac{2\delta_G}{1-\delta_G}|X_i^M \cup X_j^M|} \quad \text{(by 3)}$$

$$= \frac{1-\delta_G}{1+\delta_G}\frac{|X_i^M \cap X_j^M|}{|X_i^M \cup X_j^M|} = \frac{1-\delta_G}{1+\delta_G} J(X_i^M, X_j^M).$$

Since $J(X_i^M, X_j^M) = c/(2t_M - c)$ and the expected value of c/t_M is $e^{-k\delta_M}$, it clearly follows that the minimum required expectation of Jaccard similarity τ is at least:

$$\tau = \mathbb{E}[J(X_i, X_j)] \geq \frac{1-\delta_G}{1+\delta_G} \cdot \frac{1}{2e^{k\delta_M} - 1}.$$

To find the seed SDs, we follow a similar strategy as described in (Jain *et al.*, 2017), where our G_i and G_j correspond to the long reads and the genomic hits. We start by indexing a genome G and constructing an index I_G of genome G that is a sorted list of unique pairs (i, x) where x is a k-mer in the winnow $W(G)$ and i is a starting position of x in G. We also construct a reverse index I_G^{-1}: it provides for any input k-mer x a list of all positions i in G such that $(i, x) \in I_G$. These two tables are computationally inexpensive to calculate and allow us to quickly calculate winnow $W(X_i)$ of any substring G_i in G. For any locus i within G, we enumerate a list of all pairs $C = \{(j, x) \in I_G : x \in W(G_i)\}$. By using the winnowed MinHash lemma, we know that substring G_j starting at some locus j is a potential SD match for G_i if $W(G_i)$ and $W(G_j)$ share at least $\tau \cdot s$ k-mers, where the sketch size s is set to $|W(G_i)|$. Since C is sorted by index, we can use this lemma to efficiently select all candidate locations $j \in [j_a, j_b]$ for which $J(G_i, G_j) \geq \tau$ by 'rolling' a MinHash calculation as follows (Jain *et al.*, 2017). We start by setting $j = j_a$, and then construct an ordered set

$$L = \{(y, b) : y \in W(X_i) \cup W(X_j) \text{ and } b = 1 \text{ if } y \in W(X_i) \cap W(X_j)\},$$

where each element y is assigned 1 if it belongs to the intersection of $W(X_i) \cap W(X_j)$ and zero otherwise (Such a set can be efficiently implemented with a balanced binary tree where any update operation costs only $O(\log |L|)$.). Then we keep 'rolling' G_j by increasing j: this corresponds to checking the similarity between $G_{i:i+n}$ and $G_{j+1:j+n+1}$, wherein we remove any minimizer from L which occurred at position j and add any minimizer that occurs at the position $j + n + 1$. Note that any such step costs at most $O(\log s)$ operations (where s is the sketch size). With the appropriate auxiliary structures, we can calculate the winnowed MinHash estimate of $W(X_i)$ and $W(X_{j+1})$ in $O(1)$ time. Once we find a j for which the corresponding MinHash estimate is maximal and above τ, we add the pair (i, j) to the list of found SD seeds.

3.2 Finding potential SD regions

So far, we have assumed that the value of n is fixed and that $n = |G_i| = |G_j|$. Now we lift this restriction and attempt to extend any seed SD as much as possible in both directions in order to ensure that we can find the boundaries of 'true' SDs. This can be done by iteratively increasing the values of n and m by one [each step takes $O(\log s)$ time], which essentially keeps expanding the sets $W(X_i)$ and $W(X_j)$: any minimizer which occurs at loci $i + n + 1$ and $j + m + 1$ within G is added to the ordered set L. Here we utilize the same structures as in the previous step (see Section 3.4), and keep extending SD region until the value of the winnowed MinHash estimate goes below τ. We also terminate extension if both n and m become too large (we limit SEDEF to find potential SDs of at most 1 Mbp in length, as per WGAC). Note that the term $|S(W(X_i) \cup W(X_j))|$ keeps growing while $|S(W(X_i) \cap W(X_j))|$ stays the same if two regions stop being similar after some time, which iteratively lowers the Jaccard estimate. We also interrupt the extension if the strings G_i and G_j begin to overlap. Note that we can perform this extension in the reverse fashion, by slowly decreasing the values i and j and applying the same techniques as described above. Finally, we report the largest G_i and G_j whose corresponding MinHash estimates are above τ.

For each potential SD, we also apply a q-gram filter (Jokinen and Ukkonen, 1991) in order to further reduce the rate of false positives as follows. Define the q-gram similarity $Q(G_i, G_j)$ of strings G_i and G_j to be the total number of q-mers shared by both G_i and G_j. We

Table 1. Running time performance of SEDEF in single-core mode and multi-core mode on 80 Intel Xeon E7-4860 v2 cores at 2.60 GHz

	Human (hg19)			Mouse (mm8)		
	Total	Seeding and extending	Chaining and aligning	Total	Seeding and extending	Chaining and aligning
1 core	10 h 30 min	7 h 33 min	2 h 57 m	13 h 7 min	7 h 53 min	5 h 14 min
80 cores	0 h 14 min	0 h 10 min	0 h 04 m	0 h 30 min	0 h 10 min	0 h 20 min

adapt the well-known q-gram lemma for our problem as follows: any G_i and G_j whose edit error is below δ and satisfies the SD error model will share at least $n(1-\delta_G-q\delta_M)-(np_G+1)\cdot(q-1)$ q-grams, where we assume that $n \leq m$ and where p_G is the expected number of gaps per basepair in the genome. This modification allows us to losslessly reject any pair of substrings G_i and G_j that do not satisfy the SD error model for the given value of p_G.

The aforementioned algorithm performed on the whole human genome produces >500 million potential SD regions due to the presence of various small repeats in the genome. In order to alleviate this problem, we only use k-mers that contain at least one non-repeat-masked nucleotide during the detection of seed SDs. In order to allow the case of repeats being inserted in the SD during the evolutionary process, the SD extension step uses any available k-mer to extend seed SDs. Finally, we pad each potential SD region with a pre-defined number of bases (which is a function of the size of the potential SD region) in order to further increase the probability of locating large SDs within the potential regions.

3.3 Detecting final SDs

After finding the potential SD regions, we enumerate all local alignments of size 1000 within those regions that satisfy the SD criteria. In order to do this efficiently, SEDEF employs a two-tiered local chaining algorithm similar to those in (Abouelhoda and Ohlebusch, 2003; Myers and Miller, 1995). In the first part, we use a seed-and-extend method to construct the list of matching seed locations (of size 11 and higher), and proceed by finding the longest chains formed by those seeds via an $O(n\log n)$ sparse dynamic programing algorithm as described in Abouelhoda and Ohlebusch (2003) and Myers and Miller (1995). In this step, we restrict the maximum gap size between the seeds to $l\cdot\delta_G$ in order to cluster the seeds within a chain as 'close' as possible. After finding these *initial* chains (which might span <1000 bp), we *refine* them by further chaining them into the large final chains by allowing larger gaps. In order to retain compatibility with WGAC, which allows arbitrary large gaps within the SD (since it does not penalize the gap extension), we use the affine gap penalty during the construction of SD chains; however, we limit gaps to no longer than 10 000 bp in order to avoid low-quality alignments. Chaining is accompanied by the global alignments which are done with the KSW2 library, which utilizes 'single instruction, multiple data' (SIMD) parallelization through Streaming SIMD Extensions (SSE) instructions to speed up the global sequence alignment (Li, 2017). Importantly, we report all our alignments in standard BEDPE format, together with corresponding edit strings in CIGAR format (Li *et al.*, 2009) and various other useful metrics similar to WGAC such as Kimura two parameter genetic distance (Kimura and Ohta, 1972) and Jukes-Cantor distance (Jukes and Cantor, 1969).

In our experiments, we used $k=12$ for the seed SD stage and $k=11$ for chaining step (note that this parameter is configurable by user). While lower values of k may improve the sensitivity, we found that any such improvement is rather negligible and not worth the increase in the running time. On the other hand, higher values of k improve the running time while lowering the sensitivity.

4 Results

We evaluated SEDEF using the human reference genome (UCSC hg19) and mouse reference genome (UCSC mm8), and compared its calls to WGAC calls. Note that as mentioned in the Introduction it is not possible to run WGAC without Sun Solaris operating system; therefore, we were not able to benchmark it ourselves. WGAC calls were obtained from http://humanparalogy.gs.washington.edu and http://mouseparalogy.gs.washington.edu. WGAC calls are the current gold (and only) standard of SDs in both human and mouse genomes, and are used as SD annotations by UCSC Genome Browser.

In case of human genome, the entire process took around 10 CPU h with the peak RAM usage of 7 GB in single-CPU mode. SEDEF is also highly parallelizable, and it took only 14 min for the whole process to finalize on 80 CPU cores. This is a significant improvement over WGAC, which takes several weeks to complete (private communication). Similar running times were observed in mouse genome, despite the fact that mouse genome contains significantly more repeats than human genome and thus necessitates longer running times (She *et al.* 2008). Run times on a single CPU and 80 CPU cores when ran in parallel via GNU Parallel (Tange, 2011) are given in Table 1.

SEDEF initially detected around 2 250 000 seed SD regions in human genome. After the chaining process, the final number of SDs was reduced to ≈186 400. Finally, after filtering out the common repeats and other spurious hits, we report 67 882 final SD pairs that cover 219 Mbp of the human genome. This is a significant increase over WGAC data, which reports 24 477 SD pairs that cover 159 Mbp of the genome. Of this 60 Mbp increase in the duplication content, 30 Mbp belongs to regions in the genome without common repeats. Figure 3 shows the genome coverage, together with size and error distribution of SDs found by SEDEF and WGAC. The majority of SEDEF SDs have cumulative error δ (with affine gap penalty) around 15%. As for the mouse genome, SEDEF found 352 991 final SDs which cover 259 Mbp of the genome, as compared with 140 Mbp covered by 117 213 WGAC SDs. Of the additional 120 Mbp found by SEDEF, 45 Mbp belongs to non common repeat regions.

4.1 Filter and alignment accuracy

4.1.1 Simulations

We also evaluated the accuracy of the seeding and chaining process based on total error rate δ. For this purpose, we generated 1000 random sequences of sizes 1–100 Kbp for each $\delta \in \{0.01, 0.02, \ldots, 0.30\}$ (i.e. up to 30%), and for each such sequence generated a random SD according to the SD criteria defined above (where δ_M and δ_G are randomly chosen such that they are both less than min $\{0.15, \delta\}$). All sequences and mutations were randomly generated with uniform distribution. These two sequences (original one and the randomly mutated one) were fed to SEDEF, and then we checked whether SEDEF finds a match between these two sequences, and whether this match covers the original SDs (a match covers SD if >95% of the SD bases are included in the

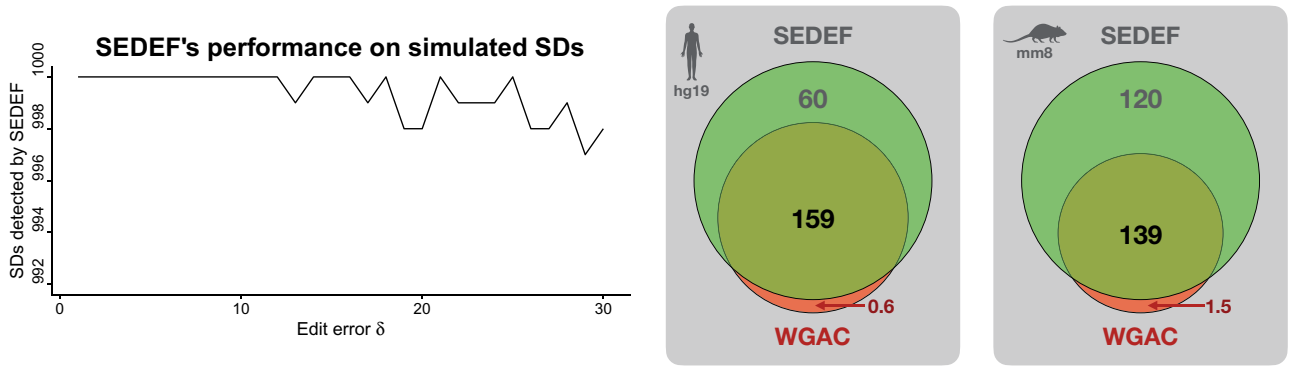

Fig. 3. (Left) Performance of SEDEF's algorithm on simulated SDs. *x*-axis is the total simulated SD error rate δ, whereas *y* axis is the number of correctly detected SDs (total 1000 for each δ). Since SEDEF successfully detects more than 995 simulated SDs for any δ, the plot area is cropped. **(Right)** Venn diagram depicts the SD coverage of the human and mouse genome (in Mbp) as calculated by SEDEF and WGAC. Intersected region stands for the bases covered by both SEDEF and WGAC

match). As shown in Figure 3, SEDEF's overall sensitivity is 99.94%, and the sensitivity drops slowly as δ increases. However, even for $\delta = 0.30$, sensitivity remains above 99%.

We performed a similar experiment on chromosome 1, where we randomly fetched 10 000 sequences (uniform distribution) of various lengths and introduced random mutations to simulate a SD event. In this experiment, SEDEF had only a 0.15% false negative rate (i.e. undetected SDs), where all missed duplications were very small SDs of lengths ≈1000.

4.1.2 WGAC coverage

It is worth mentioning that SEDEF-detected human SDs completely cover ≈98% of the previously reported SD intervals (≈99.6% in basepairs) by WGAC. SEDEF entirely misses <0.3% (70) of SD intervals reported in WGAC results, and for the 1.4% of WGAC SDs, SEDEF reports partial overlap (i.e. <80% reciprocal overlap). All together, SEDEF misses about 0.6 Mbp out of 159 Mbp as reported by WGAC (≈0.4%), where 0.5 Mbp contained short common repeats. We note that several WGAC SDs are in fact common repeats, and that several WGAC alignments contain long gaps. This is likely due to the dependency of WGAC on common repeat annotations, which may not be comprehensive. Additionally, WGAC employs several heuristics to reinsert common repeats to fuguized putative duplications that might 'glue' very short non common repeat segments into larger segments with high repeat content that show similar alignment properties to a SD. This effect is much more present in mouse genome, where SEDEF misses 16 471 WGAC SDs (14.1%), and partially covers 2.2% of such SDs. However, in terms of basepairs SEDEF only misses 1.5 Mbp (0.1 Mbp non common repeat elements). After extra validation, we found that most of missed WGAC SDs (≈14 470) are in fact common repeats incorrectly reported as SDs; thus SEDEF misses only 1.7% of the correct WGAC SD calls.

4.2 Comparison to other methods

We also evaluated the SD discovery accuracy of whole-genome aligners Minimap2 (Li, 2018) and MUMmer/nucmer (Marçais *et al.*, 2018) on the human genome assembly (UCSC hg19). These tools do not support SD detection out of the box; however, a self assembly-to-assembly comparison can be performed in order to identify the repetitive regions in the genome. These regions can be refined into SDs after applying further processing with SDDetector (Dallery *et al.*, 2017) and filtering out candidate SDs which consist solely of common short repeats. When compared with these tools, SEDEF is an integrated pipeline for identifying SDs from scratch on a given assembly. Note that other similar tools, such as DupMasker (Jiang *et al.*, 2008), are developed to annotate SDs and require already existing SD database from similar genomes to be able to mark SDs in a given genome (Table 2).

Table 2. SD coverage of the human genome (hg19) as reported by different tools

Tool	Covers	Misses	Extra	Time (h: m)
WGAC (gold standard)	159.5	0.0	0.0	weeks
SEDEF	218.8	0.6	60.0	0:36
Minimap2	53.3	107.3	1.1	1:30
MUMmer/nucmer	142.6	30.8	13.9	≥20:00
SDDetector	30.1	130.8	1.5	≥1:00[a]

Note: Misses and Extra are calculated with respect to the WGAC SD calls, which are currently the gold standard of SD calls. Note that we have filtered out all calls where at least one mate is composed solely of common short repeats (Minimap2, MUMmer/nucmer and SDDetector) as we did on SEDEF. All running times were adjusted for 20 CPU cores (all tools which support parallelization were run on 20 cores).

[a]Adjusted running time for 20 cores; in reality, SDDetector spends ≥8 hours in the single threaded pre-processing stage. Furthermore, the reported running time only includes post-processing and does not include initial BLAST alignment calculations.

We ran these tools on 20 CPU cores using the GNU Parallel (Tange, 2011) by aligning all pairs of chromosomes in hg19. Minimap2-based approach identified only 29% of the SD intervals reported by WGAC, which spanned 33% of the duplicated basepairs (53 Mbp out of 159 Mbp). MUMmer/nucmer approached better SD coverage performance, which identified 98.8% of WGAC regions that spanned 89% of duplicated basepairs (143 out of 159 Mbp), but the analysis was much slower and completed in 20 h in the same compute setting. Minimap2 required 1.5 h of run times using 20 CPU cores (in comparison, SEDEF takes only 36 min on 20 cores).

Overall, analysis misses a significant amount of duplications in a self-comparison task when using the recommended parameters of intra-species assembly-to-assembly comparison. Meanwhile MUMmer/nucmer-based approach covers the SD regions more consistently with those reported by WGAC; however it still misses many WGAC calls which are found by SEDEF. Finally, SEDEF is able to find more calls compared with the other tools in much shorter amount of time, as shown in Table 2.

5 Conclusion

SDs are among the most important forms of genomic rearrangements that drive genome evolution. However, their accurate identification is lacking due to the unavailability of necessary computational tools. In this article we presented SEDEF to help fill this gap in methodology.

In future work, we aim to characterize the effect of various edit distance embeddings and techniques such as gapped *q*-grams (Bar-Yossef *et al.*, 2004; Burkhardt and Kärkkäinen, 2002). Although many of these techniques have been previously implemented (Hanada *et al.*, 2017), our initial experiments did not show that any such embeddings or techniques are beneficial for strings with large edit distances.

SEDEF is designed as a fast, accurate, and user friendly tool to discover duplicated segments in genome assemblies. Therefore it aims to help researchers easily identify duplicated segments in genomes from several organisms, enabling them to extend their ability to perform comparative genomic studies in complex regions of the genome. We aim to extend it with an A-Bruijn graph based analysis (Jiang *et al.*, 2007) to provide a full view of the evolution of SDs. Armed with the extensions as we mention earlier, we will then use SEDEF to fully analyze reference genome assemblies from various genomes to both evaluate the assembly accuracy, and to better understand the role of SDs in organism evolution.

Acknowledgements

We thank Evan E. Eichler for early discussions on formulating the problem, and Ashwin Narayan for helpful suggestions.

Funding

This work was supported in part by National Science and Engineering Research Council Discovery Grant to F.H., EMBO Installation Grant [IG-2521 to C.A.] and National Institutes of Health [grant GM108348 to B.B.].

Conflict of Interest: none declared.

References

Abouelhoda,M.I. and Ohlebusch,E. (2003) A local chaining algorithm and its applications in comparative genomics. In: Benson, G. and Page, R.D.M. (eds.) *Algorithms in Bioinformatics*. Springer, Berlin Heidelberg, pp. 1–16.

Alkan,C. *et al.* (2009) Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat. Genet.*, **41**, 1061–1067.

Alkan,C. *et al.* (2011a) Genome structural variation discovery and genotyping. *Nat Rev. Genet.*, **12**, 363–376.

Alkan,C. *et al.* (2011b) Limitations of next-generation genome sequence assembly. *Nat. Methods*, **8**, 61–65.

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Andoni,A. *et al.* (2010) Polylogarithmic approximation for edit distance and the asymmetric query complexity. In: *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS '10*, IEEE Computer Society, Las Vegas, Nevada, USA, pp. 377–386.

Backurs,A. and Indyk,P. (2015) Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15. ACM, New York, NY, USA, pp. 51–58.

Bailey,J.A. *et al.* (2001) Segmental duplications: organization and impact within the current human genome project assembly. *Genome Res.*, **11**, 1005–1017.

Bailey,J.A. *et al.* (2002) Recent segmental duplications in the human genome. *Science*, **297**, 1003–1007.

Bar-Yossef,Z. *et al.* (2004) Approximating edit distance efficiently. In: *Proceedings of the 45th Annual IEEE Symp. Foundations of Computer Science*, pp. 550–559.

Broder,A.Z. (1997) On the resemblance and containment of documents. In: *Proceedings of the Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pp. 21–29.

Burkhardt,S., and Kärkkäinen,J. (2002) One-gapped q-gram filters for Levenshtein distance. In: *Annual Symposium on Combinatorial Pattern Matching*, pp. 225–234. Springer.

Carruthers-Smith,K. (2013) *Sliding window minimum implementations*. https://people.cs.uct.ac.za/~ksmith/articles/sliding_window_minimum.html (28 January 2018, date last accessed).

Chaisson,M.J.P. *et al.* (2015) Genetic variation and the de novo assembly of human genomes. *Nat. Rev. Genet.*, **16**, 627–640.

Dallery,J.-F. *et al.* (2017) Gapless genome assembly of colletotrichum higginsianum reveals chromosome structure and association of transposable elements with secondary metabolite gene clusters. *BMC Genomics*, **18**, 667.

Fan,H. *et al.* (2015) An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data. *BMC Genomics*, **16**, 522.

Firtina,C., and Alkan,C. (2016) On genomic repeats and reproducibility. *Bioinformatics*, **32**, 2243–2247.

Gonzalez,E. *et al.* (2005) The influence of CCL3L1 gene-containing segmental duplications on HIV-1/AIDS susceptibility. *Science*, **307**, 1434–1440.

Hanada,H. *et al.* (2011) A practical comparison of edit distance approximation algorithms. In: *Proceedingss of 2011 IEEE International Conference on Granular Computing, GrC-2011*, IEEE Computer Society, Kaohsiung, Taiwan, pp. 231–236.

Harris,R.S. (2007) *Improved pairwise alignment of genomic DNA*. PhD Thesis, Pennsylvania State University, University Park, PA, USA. AAI3299002.

Hedges,S.B. and Kumar,S. (2009) *The Timetree of Life*. OUP, Oxford.

Hollox,E.J. *et al.* (2008) Psoriasis is associated with increased beta-defensin genomic copy number. *Nat. Genet.*, **40**, 23–25.

Jain,C. *et al.* (2017) A fast approximate algorithm for mapping long reads to large reference databases. In: Sahinalp, S.C. (ed.), *Proceedings of 21st Annual International Conference on Research in Computational Molecular Biology (RECOMB 2017)*, Vol. 10229, Springer International Publishing, Cham., pp. 66–81.

Jain,M. *et al.* (2018) Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat. Biotechnol.*, **36**, 338–345.

Jiang,Z. *et al.* (2007) Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. *Nat. Genet.*, **39**, 1361–1368.

Jiang,Z. *et al.* (2008) Dupmasker: a tool for annotating primate segmental duplications. *Genome Res.*, **18**, 1362–1368.

Jokinen,P. and Ukkonen,E. (1991) *Two Algorithms for Approxmate String Matching in Static Texts*. Springer, Berlin, Heidelberg, pp. 240–248.

Jukes,T.H. and Cantor,C.R. (1969) Evolution of protein molecules. In: Munro,H. (ed.) *Mammalian Protein Metabolism, III*. Academic Press, New York, pp. 21–132.

Kimura,M. and Ohta,T. (1972) On the stochastic model for estimation of mutational distance between homologous proteins. *J. Mol. Evol.*, **2**, 87–90.

Levenshtein,V. (1966) Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Doklady*, **10**, 707–710.

Li,H. (2017) KSW2: global alignment and alignment extension. https://github.com/lh3/ksw2.

Li,H. (2018) Minimap2: fast pairwise alignment for long dna sequences. *Bioinformatics*. doi: 10.1093/bioinformatics/bty191.

Li,H. *et al.* (2009) The sequence alignment/map format and samtools. *Bioinformatics*, **25**, 2078–2079.

Li,R. *et al.* (2010) The sequence and de novo assembly of the giant panda genome. *Nature*, **463**, 311–317.

Marçais,G. *et al.* (2018) Mummer4: a fast and versatile genome alignment system. *PLoS Comput. Biol.*, **14**, e1005944.

Marques-Bonet,T. *et al.* (2009) A burst of segmental duplications in the genome of the African great ape ancestor. *Nature*, **457**, 877–881.

Mills,R.E. *et al.* (2011) Mapping copy number variation by population-scale genome sequencing. *Nature*, **470**, 59–65.

Montgomery,S.B. *et al*. (2013) The origin, evolution, and functional impact of short insertion-deletion variants identified in 179 human genomes. *Genome Res.*, **23**, 749–761.

Mostovoy,Y. *et al*. (2016) A hybrid approach for de novo human genome sequence assembly and phasing. *Nat Methods*, **13**, 587–590.

Myers,G., and Miller,W. (1995) Chaining multiple-alignment fragments in sub-quadratic time. In: *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 38–47.

Prado-Martinez,J. *et al*. (2013) Great ape genetic diversity and population history. *Nature*, **499**, 471–475.

Pu,L. *et al*. (2018) Detection and analysis of ancient segmental duplications in mammalian genomes. *Genome Res.*, **28**, 901–909.

Scally,A. (2016) The mutation rate in human evolution and demographic inference. *Curr. Opin. Genet. Dev.*, **41**, 36–43.

Schleimer,S. *et al*. (2003) Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 76–85. ACM.

Schwartz,S. *et al*. (2003) Human-mouse alignments with BLASTZ. *Genome Res.*, **13**, 103–107.

She,X. *et al*. (2008) Mouse segmental duplication and copy number variation. *Nat. Genet.*, **40**, 909.

Steinberg,K.M. *et al*. (2017) Building and improving reference genome assemblies. *Proc. IEEE*, **105**, 422–435.

Sudmant,P.H. *et al*. (2010) Diversity of human copy number variation and multicopy genes. *Science*, **330**, 641–646.

Sudmant,P.H. *et al*. (2013) Evolution and diversity of copy number variation in the great ape lineage. *Genome Res*, **23**, 1373–1382.

Tange,O. (2011) Gnu parallel - the command-line power tool. *Login USENIX Magazine*, **36**, 42–47.

Treangen,T.J., and Salzberg,S.L. (2011) Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nat. Rev. Genet.*, **13**, 36–46.

Yang,Y. *et al*. (2007) Gene copy-number variation and associated polymorphisms of complement component C4 in human systemic lupus erythematosus (SLE): low copy number is a risk factor for and high copy number is a protective factor against SLE susceptibility in European Americans. *Am. J. Hum. Genet.*, **80**, 1037–1054.

Yeo,S. *et al*. (2018) ARCS: scaffolding genome drafts with linked reads. *Bioinformatics*, **34**, 725–731.