Accurate detection of complex structural variations using single molecule sequencing

Fritz J. Sedlazeck*, Philipp Rescheneder*, Moritz Smolka, Han Fang, Maria Nattestad, Arndt von Haeseler, Michael C. Schatz

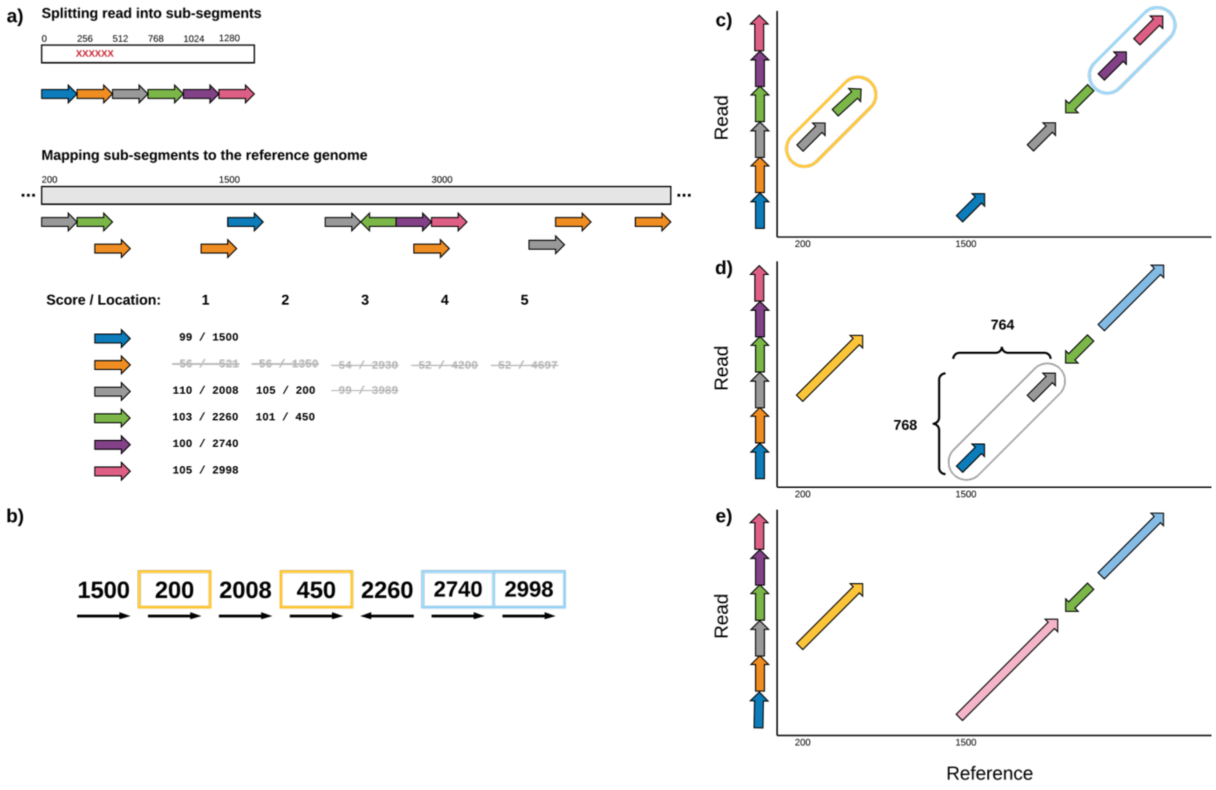
Supplemental Materials

1. NGMLR Alignment Algorithm

NGMLR is designed to accurately map long single molecule sequencing reads from either Pacific Biosciences or Oxford Nanopore to a reference genome with the goal of enabling precise structural variation calls. See Figure 1 in the main text for an overview of the overall alignment algorithm. In the following, we describe the details of alignment computation in NGMLR.

1.1 Detection of linear mapping pairs

1.1.1 Sub-segment alignment



Supplemental Figure 1.1. NGMLR workflow for detecting linear mapping pairs (LMPs). (a) Reads are split into sub-segments and aligned to the reference genome (a). A modified longest increasing subsequence algorithm detects sub-segments that map co-linearly to the reference sequence (b and c). LMPs located on the same diagonal in the alignment matrix are merged (d) to form the final set of LMPs (e).

To identify local similarities between a long read and the reference genome, NGMLR splits each read into non-overlapping 256 bp sub-segments and maps them to the reference genome independently of each other using the seed and vote approach described by Sedlazeck, et al. [1] (**Supplementary Figure 1.1a**). Briefly, a sub-segment is decomposed into all overlapping k-mers (13-mers per default). For each k-mer, the location(s) for that k-mer on the reference genome are retrieved from a hash table index data structure. All regions of the reference genome that exceed a certain number of k-mer matches are considered candidate mapping regions (CMR) for the sub-segment. Next, a pairwise local alignment score for all CMRs and the sequence of the sub-segment is computed. NGMLR sorts all CMRs based on their alignment score and retrieves the highest score. All CMRs with a score lower than 75% of the highest score are discarded. We call all remaining CMRs “anchors” between the sub-segment and the reference genome. An anchor is described by its starting position on the long read, its mapping position on the reference genome and the respective alignment score.

Sub-segments that map to highly repetitive regions with more than 1000 (default) anchors are discarded, as they are not informative for finding local similarities between the read and the reference. **Supplementary Figure 1c** shows the result of this step: A set of high quality anchors that consist of their position of origin on the read and one or more mapping locations on the reference sequence including alignment scores. The length and number of LMPs found for a single read will depend on the quality of the read, whether it spans a structural variation or not, and on how repetitive its sequence is.

1.1.2 Building linear mappings segments

Next, NGMLR identifies all segments of the read that are not interrupted by a structural variation and can therefore be represented by a single linear alignment to the reference genome. To this end, NGMLR identifies the largest set of sub-segments that map co-linearly to the reference genome. In other words, NGMLR looks for sub-segment mappings that are located on the same diagonal in a hypothetical dot plot of the read and the reference genome. The search for sub-segments that map in the same order to the read and the reference can be implemented by sorting the sub-segment mappings based on their position on the read and searching for the longest increasing subsequence (LIS) of their respective reference coordinates [2]. To enforce co-linearity between sub-segment mappings, we extended the basic LIS algorithm to include the following restrictions. Two sub-segment mappings can only be included in the LIS if they are on the same strand and if the distance between their starting positions on the read and distance between there mapping location on the reference genome deviates by only 25 % of the sub-segment length. This ensures that the two sub-segments are not separated by a structural variation. To avoid merging of two unrelated sub-segments, we further require the distance of two sub-segment mappings on the reference genome and on the read, be less than two times the length of the sub-segment. This constrained longest increasing subsequence algorithm allows us to identify the largest set of co-linear sub-segment mappings. Joining this set gives us the longest linear mapping pair (LMP) of the read. As a read that spans a structural variation might generate more than one LMP, NGMLR repeats this step until it is unable to find any more LMPs with support from at least two sub-segment mappings.

1.1.3 Merging compatible linear mapping pairs

So far, NGMLR has identified a set of segments of the read that do not span structural variations and therefore align linearly to the reference genome. However, for a sufficiently long read, insertions and deletions shorter than the read length can be part of a linear local alignment. To identify the minimal number of linear local alignments needed to correctly map a read NGMLR next looks for pairs of LMPs that are separated by short indels or were falsely split because of sequencing error and merge them. To this end, NGMLR divides all LMPs into subsets such that within a subset all LMPs are in a per default 8000 bp wide corridor in a hypothetical dot-plot between the read and the reference sequence. Next, NGMLR sorts the LMPs within each subset by their start location on the read and iteratively attempts to merge adjacent LMPs.

The relative location of two adjacent LMPs can indicate whether they are separated by a structural variation or not. If the distance between two LMPs on the read and on the reference is the same, meaning they align collinearly, this indicates that they are not separated by a structural variation (**Supplementary Figure 1.1d**). Thus, NGMLR will join them. In contrast, a larger distance on the reference indicates a deletion, while a larger distance on the read indicates an insertion. In this case, NGMLR joins two LMPs only if the size of the insertion or deletion is smaller than both LMPs. LMPs on opposite strands are never joined (**Supplementary Figure 1.1d**).

1.1.4 Extending linear mapping pairs

LMPs frequently do not contain the first and the last few base pairs of a long high error rate read. Therefore, NGMLR extends all LMPs by two times the sub-segment length. Furthermore, NGMLR closes all gaps between two adjacent LMPs that are within an alignment corridor but were not merged (e.g. because of different strands).

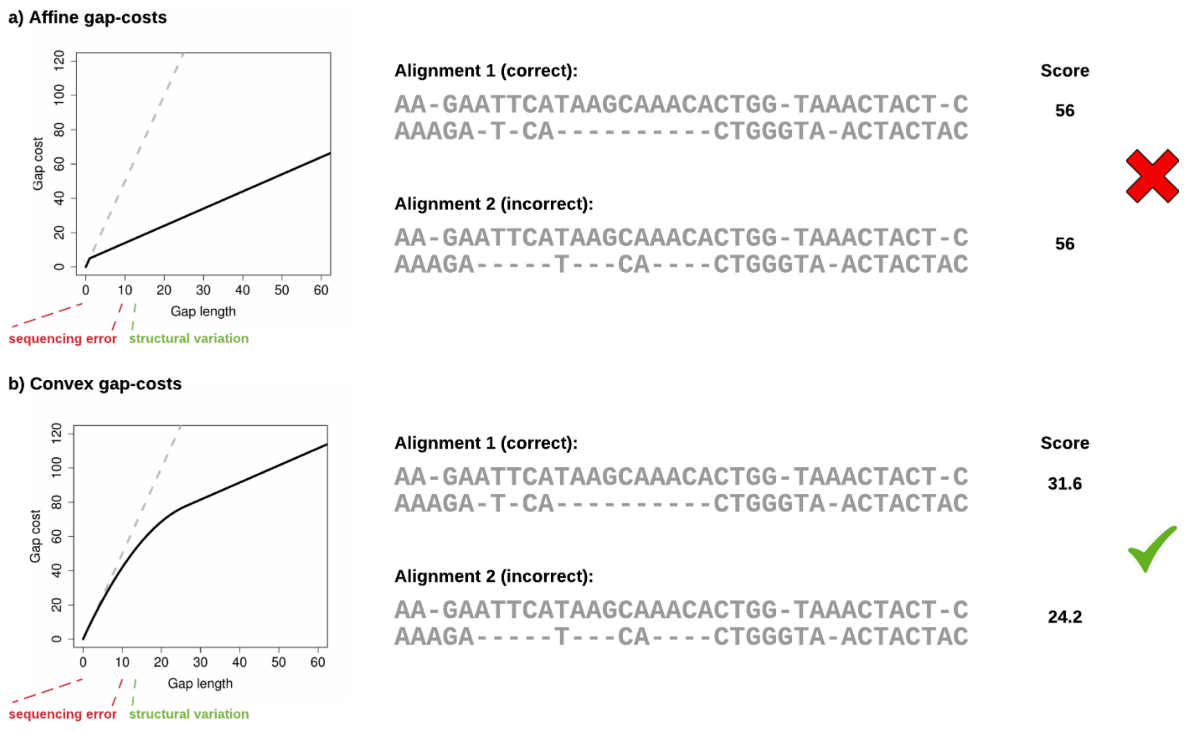
1.2 Computing pairwise alignments with convex gap costs

1.2.1 Convex gap-cost model

Each linear mapping pair (LMP) is recorded by the start and end position on the read and a start and end position on the reference genome but not the specific sequence alignment. Therefore, in the next step, for each LMP, NGMLR extracts the read sequence and the reference sequence and uses a Smith-Waterman-like dynamic programming algorithm to compute the pairwise sequence alignment. When aligning long-reads it is crucial to choose an appropriate gap model as there are two distinct sources of insertions and deletions (indels): Sequencing error predominantly causes very short randomly distributed indels (1-5bp), while biological structural variations cause longer indels (20bp+). Furthermore, for indels caused by structural variations it is more likely to find one large indel than two smaller indels in close proximity.

Currently, two gap models are mainly used: linear and affine gap cost models. A linear gap cost model – where the cost of a gap with length *L* equals the cost of *L* gaps with length 1 - appropriately models indels originating from sequencing error. However, they favor shorter gaps and therefore cause long indels stemming from SV to be falsely split into several smaller indels. Affine gap costs more realistically model indels from SVs by introducing a separate penalty for gap opening and gap extension. However, for long-reads the effect of the higher gap-open penalty is outweighed by the cost of the gaps from the many sequencing errors, causing them to be falsely clustered. Therefore, the affine gap cost only has a small effect on longer indels, especially when indels are located in regions of low sequence complexity. **Supplementary Figure 1.2a** shows two different pairwise alignments of the same sequences.

Alignment 1 is the correct alignment showing one long deletion stemming from a SV and six 1bp indels stemming from sequencing error. In Alignment 2 the deletion is split into three mid-sized deletions and only four 1bp indels are reported. However, the number of gap openings and gap extensions is the same between both alignments. Therefore, with an affine gap cost model the alignment score of both alignments is the same.



Supplemental Figure 1.2. Two different alignments for the same sequences with affine gap-costs (a) and convex gap-costs (b). Only with convex gap-costs, the correct alignment shows a higher score than the incorrect alignment.

To account for sequencing error and real SVs at the same time, NGMLR uses convex gap-costs for aligning long-reads. Appropriately parameterized, Convex gap costs mimic linear gap costs for short indels (e.g. sequencing errors) while at the same time favoring longer gaps for indels stemming from structural variations (**Supplementary Figure 1.2b**).

We define the cost of a gap G of length i to be:

$$G(i) = \begin{cases} g_O, & i = 0 \\ G(i-1) + min\begin{cases} g_M \\ g_E + g_D * (i-1) \end{cases}, & i > 0 \end{cases}$$

Like the affine gap cost model, g_O (default: -5) only applies to the first gap character while g (default: -5) is used for any additional gap character. In addition, we introduce a gap decay parameter g_D (default: 0.15) that reduces the cost of adding an additional gap character depending on the total length of the gap. In other words, the longer a gap is the lower the penalty of extending it. To prevent the penalty for extending a gap from going to zero, we also introduce the gap min g_M (default: -1) parameter.

For very short insertions and deletions, where g_D is much smaller than $g_E * (i-1)$, our gap model behaves similarly to linear gap penalties (**Supplementary Figure 1.2b**, gap length 0 – 10). Meaning that e.g. two gaps of length one are assigned a very similar score as a single gap of length two ($G_1 + G_1 \approx G_2$). This models indels originating from random sequencing error. For longer insertions or deletion, where $g_E * (i-1) \approx g_M$, our gap model favors one longer gap over two smaller ones as extending a longer gap becomes much cheaper than opening a new gap ($G_{25} + G_{25} \gg G_{50}$). Therefore, the alignment score of the correct alignment 1 from **Supplementary Figure 1.2b** is higher than the score of the incorrect Alignment 2.

Available algorithms capable of using this gap model for computing alignments have to scan the full row i and column j of the alignment matrix V to compute the correct score of any cell $V_{i,j}$. This increases the runtime complexity from $O(m^2)$ to $O(m^3)$ for a naïve implementation. Using such an algorithm for aligning long-reads to a reference genome is computationally infeasible. Gusfield[3] describes an improved implementation with $O(m^2 * \log(m))$. Although more favorable in terms on runtime complexity, this algorithm is complex and hard to optimize and therefore still not fast enough for mapping large data-sets in practice. Therefore, we adapted a heuristic implementation of the convex gap cost algorithm found in swalign (https://github.com/mbreese/swalign). We follow the approach of linear gap costs were the value of $V_{i,j}$ only depends on $V_{i-1,j}$, $V_{i,j-1}$ and $V_{i-1,j-1}$. However, we define a function $g(l)$ the gives us the penalty of extending an indel of length l by one as follows:

$$g(l) = \begin{cases} g_O, & l = 0 \\ min\begin{cases} g_M \\ g_E + g_D * l \end{cases}, & l > 0 \end{cases}$$

Furthermore, we introduce two additional matrices to keep track of our length estimates for insertions $I_{i,j}$ and deletions $D_{i,j}$ while computing $V_{i,j}$:

$$D_{i,j} = \begin{cases} D_{i-1,j} + 1, & V_{i,j} = V_{i-1,j} + g(D_{i-1,j}) \\ 0, & otherwise \end{cases}$$
$$I_{i,j} = \begin{cases} I_{i,j-1} + 1, & V_{i,j} = V_{i,j-1} + g(I_{i,j-1}) \\ 0, & otherwise \end{cases}$$

In other words, if the maximum score in $V_{i,j}$ was derived from $V_{i-1,j}$ we assume that the final alignment will contain a deletion at this position. Therefore, we increase the estimated length of the deletion at cell $V_{i,j}$ by one. Finally, we compute the alignment matrix as follows:

$$V_{i,j} = max\begin{cases} V_{i-1,j-1} + s(Q_i, S_j) \\ V_{i-1,j} + g(D_{i-1,j}) \\ V_{i,j-1} + g(I_{i,j-1}) \end{cases}$$

where $s(Q_i, S_j)$ is the match score or the mismatch penalty for the reference base i and the read base j. After computing V we search for the element with the highest score and use backtracking to find the full sequence alignment.

1.2.2 Speeding up alignment computation

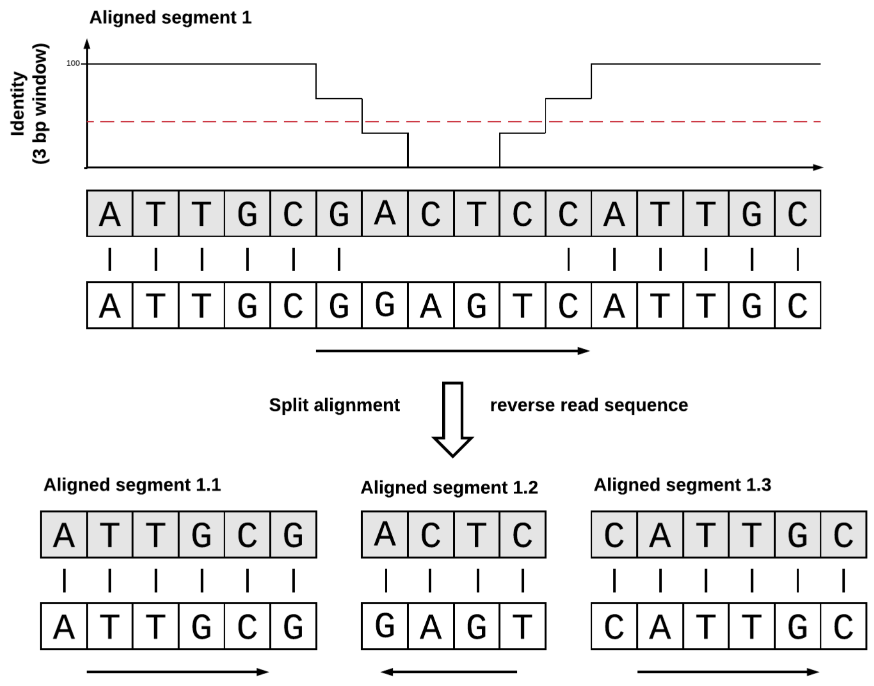
Computing the full alignment matrix V is infeasible for long read alignments [4]. Since NGMLR knows from the sub-segment alignments that the read and the reference sequence are highly similar, the optimal alignment will be close to the diagonal of the full alignment matrix. Therefore, NGMLR computes a banded alignment centered between the start and end positions of the LMP and extend its width until all sub-segment mappings the LMP is based on are contained. In cases where NGMLR underestimated the bandwidth, the computed alignment will not span the full read sequence but stop at or very close to the border of the alignment band. NGMLR detects such cases during backtracking and then recomputes the alignment with a larger band.

NGMLR further optimizes the computation of the convex alignments by applying vectorization using SSE2 instructions. SSE are a form of SIMD (single instruction, multiple data) and save computational time by processing multiple values simultaneously. The use of SSE allows NGMLR to raise the number of reads that are mapped within the same amount of time, independently of the number of CPU threads used. Computing the alignment matrix V is the bottleneck when computing convex gap cost alignments and therefore was chosen as primary target for optimization with SSE. In this step, the alignment matrix is filled with the optimal sub-alignment scores for each letter pair between the read and reference sequence. NGMLR applies vectorization to optimize the forward step, by concurrently computing the values of multiple cells in the alignment matrix. Most of the computationally expensive branching operations, e.g. the selection of the optimal scores, are further replaced with linear arithmetic operations. The concurrent step is followed up by a regular non-SIMD pass which then resolves the dependencies between the values of the cells which were computed in parallel.

1.3 Small inversion detection

The LMP identification and the alignment step account for most types of structural variations. However, short inversions and balanced translocations are difficult to detect as they often do not get sufficient sub-segment support to be identified during the LMS identification step. Therefore, two LMS that are separated by a small inversion or balanced translocations are sometimes falsely merged. Even during the subsequent alignment computation of the merged LMS these SVs keep undetected especially in long reads since the score penalty of misaligning e.g. a short inverted read segment is small compared to the overall alignment score of the rest of the read. Thus, the segment of the read covering the inversion is forced to align to its reverse complement in the reference genome. This leads to random alignments for the inverted segment and makes it impossible to detect the inversion in down-stream analysis.

The correct way to handle such an inversion is splitting the reads at the borders of the inversion and mapping it in three separate segments. To this end, NGMLR scans all aligned LMS for parts with a sequence identity smaller 65% (as expected from random alignments). If such a region is detected, NGMLR extracts the respective sequence, computes the reverse complement and realigns it to the reference genome. If the score increases compared to the original alignment and is above an empirically determined threshold, NGMLR reports the inversion.



Supplementary Figure 1.3. Detection of a 5bp inversion in a 17 bp alignment.

To detect misalignments caused by short inversion and balanced translocations, NGMLR scans all LMP alignments for parts with low sequence identity. To this end, NGMLR computes for each match or mismatch position of the alignment its local sequence identity I_i as follows:

$$I_i = \frac{m}{32}$$

Where m is the number of matches found between $i - 16$ and $i + 16$ in the alignment. Next, NGMLR scans for clusters of positions I with an identity Ii < 65 %. Briefly, NGMLR looks for neighboring positions with I_I < 65 % which are separated by not more than 20 bp. If it finds a cluster C that covers at least 40 bp, NGMLR extracts the covered read sequence and aligns this sequence and its reverse complement to the reference sequence. If the alignment score is higher for the reverse complement, the read covers an inversion. Therefore, NGMLR splits the LMP at the borders of the inversion and recomputes the alignments for the three resulting LMPs. **Supplementary Figure 1.3** shows an example of an aligned LMP. The read and the reference sequences are identical except for 4 bp inversion. For simplicity, we chose a 3 bp window to compute the local sequence identity for this example. For the positions of the alignment that cover the inversion the local sequence identity drops (**Supplementary Figure 1.3 top**). All positions below the 65 % threshold are extracted. The extracted sequence and its reverse complement are aligned to the reference genome. Since the alignment of the reversed sequence is higher, the LMP is split and the respective alignments computed (**Supplementary Figure 1.3 bottom**).

1.4 Selection of linear alignments & Mapping Quality computation

The final set of aligned LMPs contains all linear alignments required for the correct read mapping. However, due to repeats in the reference genome, segments of a read can map to more than one location. For such a segment, NGMLR would detect two independent linear alignments. However, in the final output we want every nucleotide of the read aligned to exactly one nucleotide in the reference sequence. Therefore, NGMLR must choose the best combination of linear alignments that do not overlap on read coordinates. NGMLR uses a dynamic programming algorithm that determines the non-overlapping set of linear alignments with the maximal joint score by computing the best joint score S(i) for all prefixes of the read that end in position i.

For illustration, assume a read R has a linear alignment with score s that starts on the read at position 3000 and ends at position 8000. If we know all $S(j)$ with $j < 8000$:

$$S(8000) = max\begin{cases} S(7999) \\ S(2999) + s \end{cases}$$

Similarly, $S(1)$ only depends on $S(0)$ and the scores of all linear alignments that end in position 1. Since we know the alignment scores for all linear alignments, and $S(0) = 0$, we can compute S(1) and subsequently S(i) for all prefixes of R. Finally, we use a simple backtracking procedure, to determine the set of linear alignments S was computed from.

For each linear alignment of this set, NGMLR finally computes a mapping quality value individually. We define the mapping quality of a linear alignment to be the average mapping quality of all its overlapping sub-segments. Note that this potentially underestimates mapping quality, as the mapping of an LMS can be unique with respect to the genome even if all its sub-segments have a low mapping quality.

1.5 Table of NGMLR Parameters

Parameter	Default	Explanation
-r, --reference	NA	Path to the reference genome (FASTA/Q, can be gzipped)
-q, --query	NA	Path to the read file (FASTA/Q) [/dev/stdin]
-o, --output		Path to output file [stdout]
--skip-write	false	Don't write reference index to disk
--bam-fix	false	Report reads with > 64k CIGAR operations as unmapped. Required to be compatibel to BAM format [false]
-t, --threads	1	Number of threads
-x, --presets	pacbio	Parameter presets for different sequencing technologies
-i, --min-identity	0.65	Alignments with an identity lower than this threshold will be discarded
-R, --min-residues	0.25	Alignments containing less than <int> or (<float> * read length) residues will be discarded
--no-smallinv	false	Don't detect small inversions
--no-lowqualitysplit	false	Split alignments with poor quality
--verbose	false	Debug output
--no-progress	false	Don't print progress info while mapping
--match	2	Match score
--mismatch	-5	Mismatch score
--gap-open	-5	Gap open score
--gap-extend-max	-5	Gap open extend max
--gap-extend-min	-1	Gap open extend min
--gap-decay	0.15	Gap extend decay
-k, --kmer-length	2	Number of k-mers to skip when building the lookup table from the reference
--bin-size	4	Sets the size of the grid used during candidate search
--max-segments	1	Max number of segments allowed for a read per kb
--subread-length	256	Length of fragments reads are split into
--subread-corridor	40	Length of corridor sub-segments are aligned with

2. Structural Variant Detection with Sniffles

Sniffles is a long read based structural variation (SV) caller capable of detecting deletions, tandem duplications, insertions, inversions, translocations as well as combinations (nested SV) of these five SV types. Sniffles incorporates multiple approaches for detecting small (by default: 30bp) to large (10kb+) SV. Furthermore, Sniffles implements multiple novel techniques to ensure a low false discovery rate. In the following, we give a detailed description how Sniffles works and the implemented novel mechanisms that lead to an enhancement in sensitivity and precision.

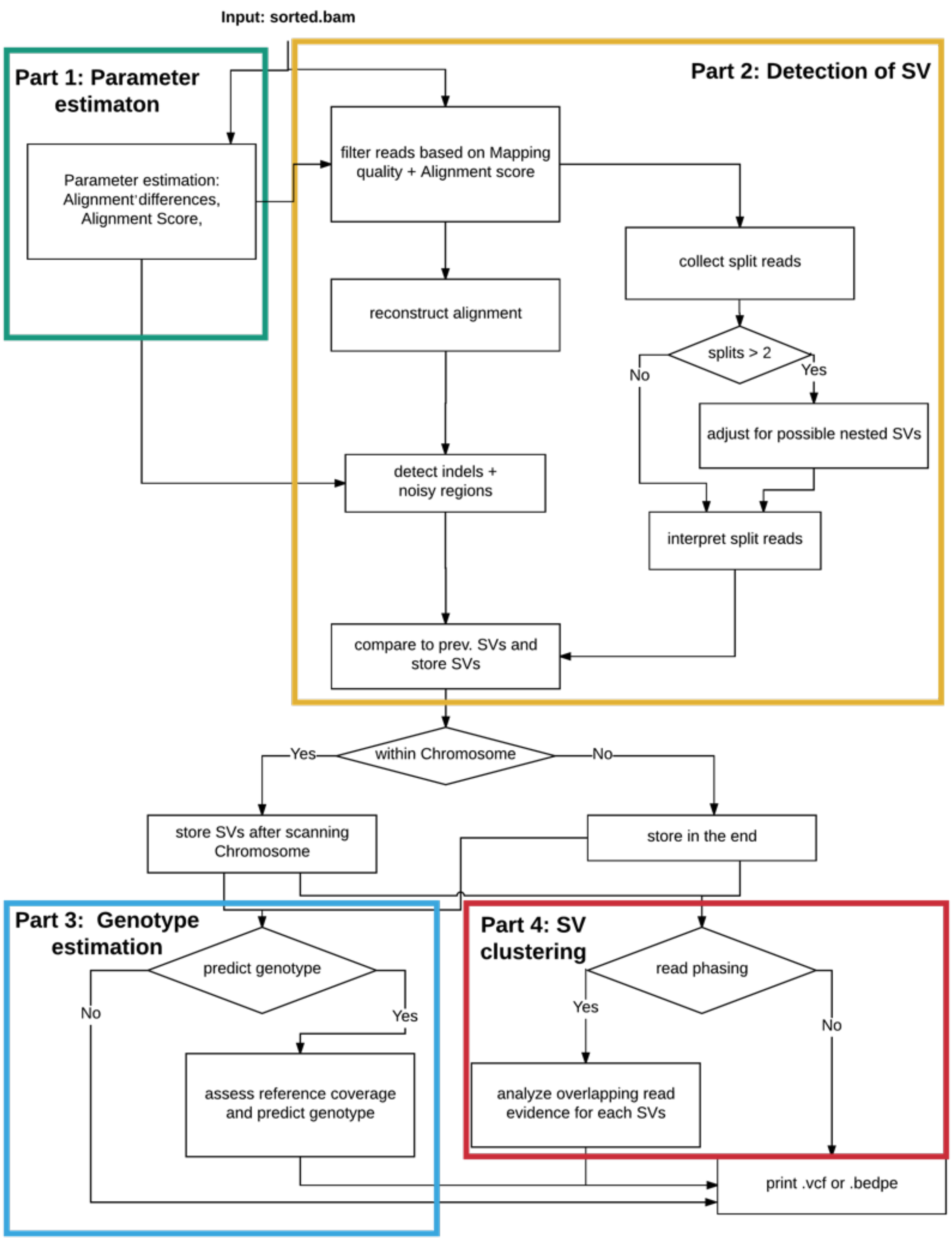
Sniffles incorporates four major phases, which are all automatically executed during runtime (**Supplementary Figure 2.1**). First, Sniffles quantifies characteristics of the input data such as the overall sequencing error rate, SNP rate, and the average alignment scores of the reads. Second, Sniffles uses within-alignment and split read information to call SVs. Finally, in steps 3 and 4 Sniffles infers genotype and phasing information from the SVs called in the previous steps. In the following we give a detailed description of the individual parts.

2.1 Preprocessing and parameter estimation

The Sniffles SV calling algorithm depends on three key parameters:

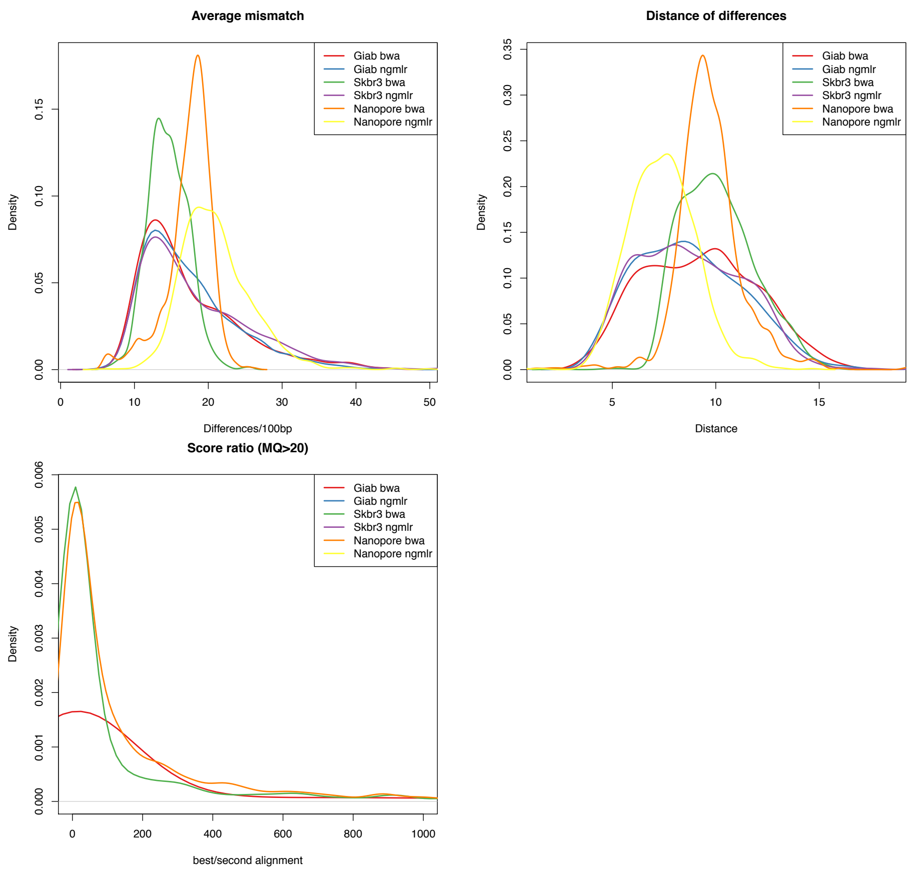
(1) The sequencing error observed of the reads, the average distance between differences (indels or mismatches) in the read alignments, and the 95th percentile of the number of mismatches and indels in a 100bp window. Sniffles uses these parameters to detect read alignments or parts of read alignments that show an increased mismatch/indel rate and therefore might overlap with a SV.
(2) The ratio between the best and second best alignment scores for each read mapping. Sniffles uses this parameter to assess the reliability of a particular read mapping.

These parameters differ between genomes and sequencing technologies. Therefore, Sniffles tries to estimate them from the input data. To this end, Sniffles scans 10,000 randomly chosen reads to obtain the distribution of these parameters. We emphasize that this is done solely to estimate these parameters and not designed to obtain a comprehensive view across the entire genome. The assumption is SVs are generally rare so that most of these reads should not overlap a SV and thus representing a solid baseline of these parameters.



Input: sorted.bam
Part 1: Parameter estimaton
Parameter estimation: Alignment differences, Alignment Score,
Part 2: Detection of SV
filter reads based on Mapping quality + Alignment score
reconstruct alignment
collect split reads
splits > 2
No
Yes
adjust for possible nested SVs
detect indels + noisy regions
interpret split reads
compare to prev. SVs and store SVs
within Chromosome
Yes
No
store SVs after scanning Chromosome
store in the end
Part 3: Genotype estimation
predict genotype
No
Yes
assess reference coverage and predict genotype
Part 4: SV clustering
read phasing
Yes
No
analyze overlapping read evidence for each SVs
print .vcf or .bedpe

Supplementary Figure 2.1. Overview of Sniffles workflow. Sniffles incorporates four major steps. (1) First, it estimates the parameters given the mapped read file. (2) Second, it identifies SVs based on split read and alignment events. (3) Third, optionally it estimates the genotype of the called SVs. (4) Fourth, optionally it attempts to cluster SVs together based on the same set of reads that are overlapping (Part 4).



Supplementary Figure 2.2. Density Plots for the parameter estimation of two different PacBio data sets (Giab, SKBR3) and one Nanopore (NA12878)

2.2 Scanning for SVs

To reduce false positive SV calls Sniffles stringently filters for spurious read mappings. A read is discarded if it has a mapping quality lower than 20 (by default) or the ratio of its best and second best alignment score is less than 2, or its alignment score ratio is smaller than the minimum alignment score ratio computed by the parameter estimation step (section 2.1). Furthermore, a read is discarded if it shows more than 7 (by default) split read alignments or if every aligned portion of the read does not exceed 1kbp (by default).

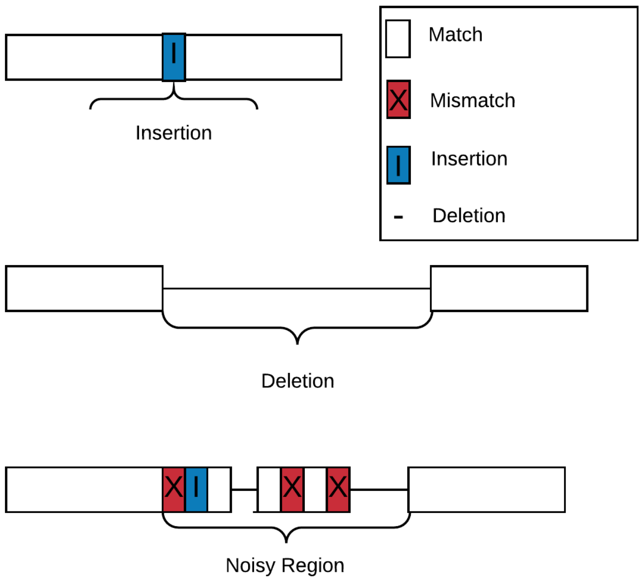
For each of the remaining reads, Sniffles performs the following four steps to detect SVs.

1. First, Sniffles scans the read alignments to detect smaller (<1kb) insertions, deletions and regions with an increased number of mismatches and very short (1-5bp) indels. These “noisy regions” often indicate incorrect read mappings caused by SV.
2. Second, Sniffles processes split read information to identify SV that cannot be represented in a single alignment (large indels, inversions, duplications, translocations). All SVs found during step 1 and 2 are stored in a self-balancing binary tree.
3. Sniffles traverses the binary tree to merge SV calls that were caused by the same SV.
4. After all reads were scanned for SVs, Sniffles identifies spurious SV calls and discards them.

2.2.1 Alignment analysis

To identify SVs within read alignments Sniffles first extracts the genomic position and the length of all mismatches and indels from the MD and CIGAR string of every read. All insertions and deletions longer then the user defined minimum SVs length (default: 30bp) are recorded as potential starting points of a SV. Next, we use a PlaneSweep algorithm to identify segments of the read that show an increased number of mismatches and small indels (noisy regions).

PlaneSweep algorithms are widely used in genomics for determining for example the read coverage for each position of a reference genome. Instead of read start and end positions, here we use the genomic position of the mismatches and indels as start and define the end location of the interval as its length plus 100bp (by default) to allow for some noise in the position of the SV event. Our PlaneSweep algorithm is modified so that it takes these intervals as input and outputs candidate SV regions if the number of mismatches and indels at a certain position exceeds the maximum number of differences per 100bp computed in the parameter estimation step (section 2.1). Thus, each of the coordinates identified by the PlaneSweep algorithm represent start locations of a segment along the read that shows an increased accumulation of mismatches and short indels and therefore might overlap a SV. Subsequently, for each of previously stored regions, Sniffles attempts to enlarge the size in both directions. This is done until the distance between two differences on the read minus the length of the event is larger than the maximum difference distance allowed, as determined in the parameter estimation (section 2.1).



Supplementary Figure 2.3. Schematic of alignment events. Sniffles distinguishes between 3 different types: insertions, deletions and noisy regions. Noisy regions often indicate that a read overlaps a deletion or inversion.

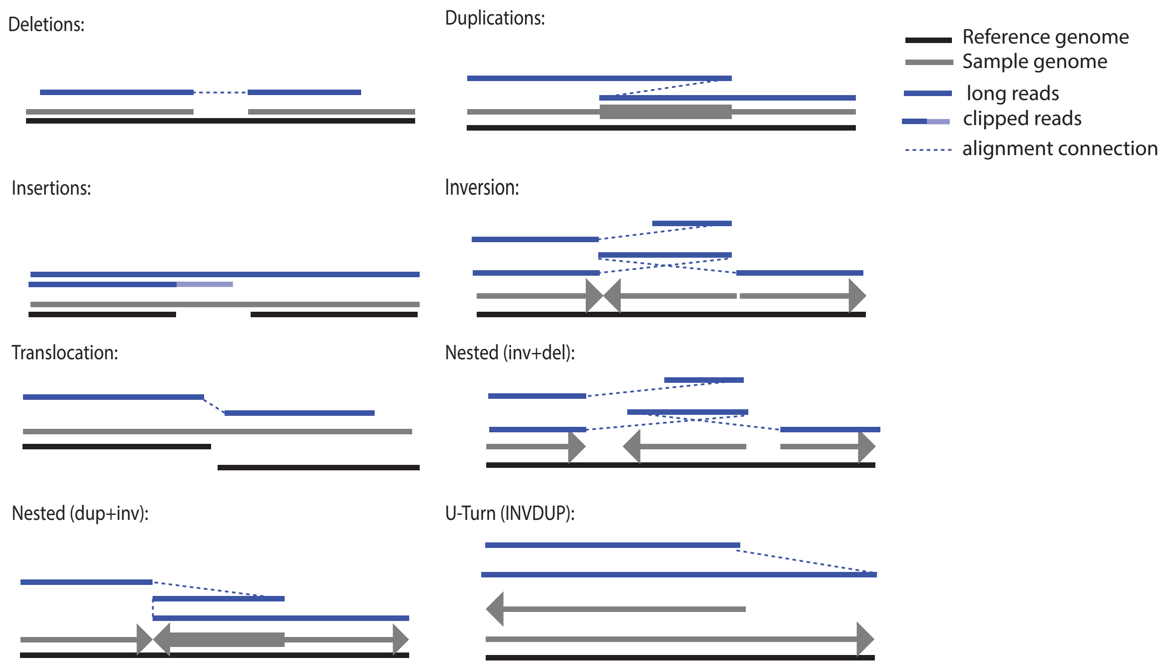
Next, Sniffles tries to determine the SV type that most likely caused the noisy regions. To this end, Sniffles uses the number mismatches (M), the maximum length of a single insertion (sl) and the maximum length of a single deletion event (dl) within the determined region. Sniffles reports an insertion if sl > minimum SVs size (default: 30bp) and the sum of all insertion lengths observed in that region is at least twice as much as sum of the length of all deletions. Deletions are identified using a reciprocal rule. We required this more complicated rule set to distinguish clear indels from noisy regions in a read. Sniffles reports the region as being noisy if it is neither a distinct insertion nor a distinct deletion, but the sum of mismatches in this area is larger than the minimum allowed SV size (default 30bp). Such noisy regions are most commonly caused by deletions or inversions, especially from aligners other than NGMLR.

In addition to these three cases (insertions, deletions and noisy regions), Sniffles also checks whether the read was clipped by more than 2kbp (by default). This often indicates that the read partly spans an insertion. Furthermore, Sniffles utilizes a flag reported by NGMLR, which indicates if a read was clipped due to N's in the reference genome or if it could not identify a region where the clipped sequence maps to. In this case we cannot determine the exact size and thus we set every insertion of this type to have a size of "NA".

At the end of this scan, Sniffles knows the number of potential insertions, deletions and noisy regions in every read. If a read contains more than 3 noisy regions it is discarded and all SV calls from this read are removed. Otherwise, all remaining SVs are inserted into our binary tree to collect all potential SVs from all valid reads (see section 2.2.3).

2.2.2 Split read analysis

Reads spanning inversions and translocations can only be mapped by splitting the read and reporting two or more separate alignments. These split read mappings are typically reported as separate SAM records (see SAM specifications)[5] in a SAM/BAM file. For each read one of the alignments (usually the longest) is marked as primary alignment. All the others are called supplementary alignments. To simplify parsing, BWA–MEM and NGMLR compute the SA tag for each SAM record. The SA tag of the primary alignment record contains all information about all supplementary alignments required for SV calling. Therefore, Sniffles only reads the primary alignment for each read from the BAM file and extracts supplementary alignment information from the SA tag.



Supplementary Figure 2.4. Schematic illustration of the different SV types Sniffles can detect and how the split reads are aligned to be able to detect these events.

Note that the SA tag does not provide a second-best alignment score, therefore Sniffles filter supplementary alignments only by MQ but not by alignment score ratio. Reads with more than N alignments (default N=7, but user definable) are typically low quality reads and are therefore ignored. Furthermore, Sniffles ignores reads where non of the sub alignments are larger then 1kbp (by default) as these often sequencing artifacts (**Supplementary Figure 2.5)**. For all other reads, Sniffles extracts starts and end positions on the read and on the reference of the primary and supplementary alignments and stores them as intervals in a list ordered by the starting position on the read.

Next, Sniffles divides the reads into two categories: reads with two alignments that most probably span simple SVs, and reads with more alignments that potentially span nested SVs. For reads consisting of two alignments Sniffles applies the following rules to detect SVs:

(1) If the two segments of the read are aligned to the same chromosome and share the same orientation then Sniffles compares their distances on the read level to their distances on the genome level. Sniffles calls an insertion if the distance on the read level minus the distance on the genome is larger than the minimum SV length (default 30bp). Sniffles calls a deletion if the distance on the genome level minus the distance on the read level is larger than the minimum SV length. If the distance on the reference is larger than the minimum SV length and the distance on the read level is smaller than the minimum SV length, Sniffles calls a duplication.

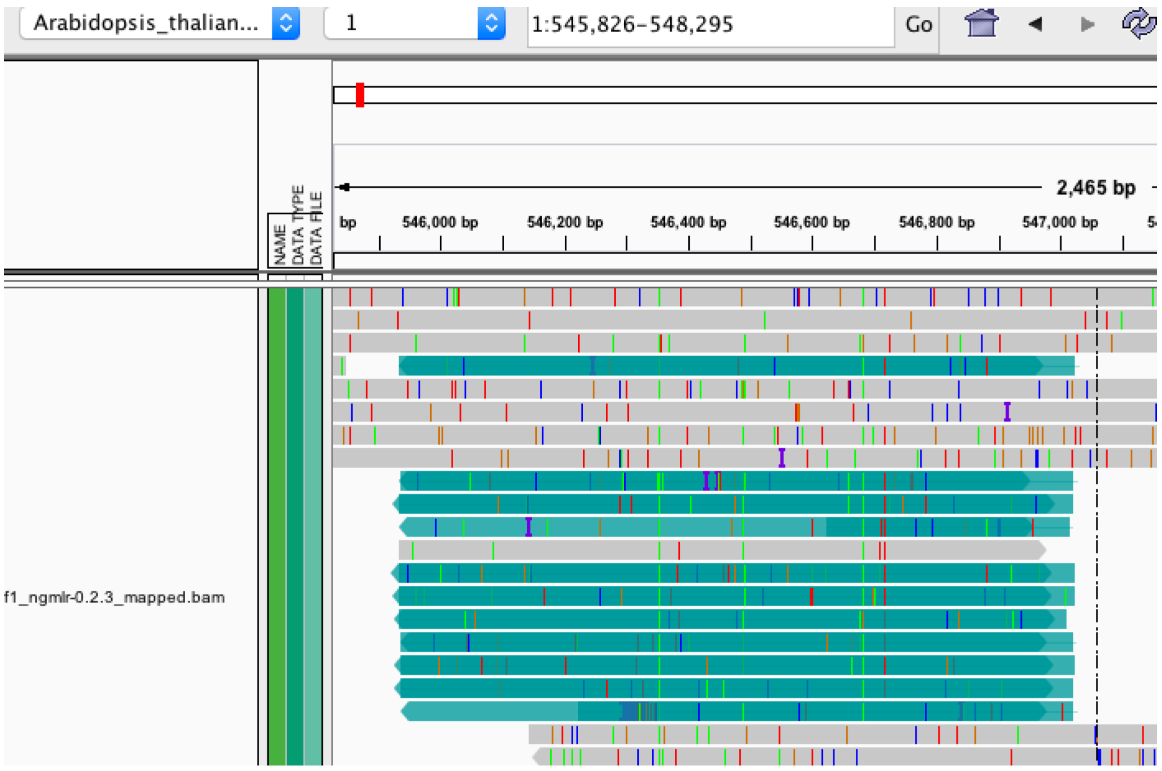
(2) If the two segments are on the same chromosome, but have different strands Sniffles calls an inversion. In case these two segments overlap by at least 200bp and at least 40% of the shorter segment length they are called an inverted duplication

("U-turn"). For inverted duplications, Sniffles requires one of the segments to be larger than 2kbp (by default). This is necessary to ignore certain base calling artifacts where a read is "folded" multiple times onto itself (**Supplemental Figure 2.5)**

(3) If the two segments are aligned to different chromosomes Sniffles calls a translocation.

Reads that consist of more than two segments with a length larger than 200 bp and map to the same chromosome potentially cover a nested SV. For such reads, Sniffles applies separate rules: If the segments are overlapping and only one segment has a different strand then the other two, an inverted duplication is called. Furthermore, to account for inversion flanked with indels we determine if one segment has a different strand then the other two flanking it and the overlap or the distance of the segments in read space vs. the reference space is more then the minimum SVs length. If this is the case, Sniffles introduces pseudo segments, which are 1bp long elements that ease the detection of such complicated inversion segments. This is necessary to generalize and distinguish this nested inversion,

Finally, all detected SVs are inserted into our binary tree of all detected SVs in all valid reads (see **Section 2.2.3**).



Supplementary Figure 2.5. Example INVDUP signal due to an error in base calling and/or subread segmentation. Note that one polymerase read (marked) is folded over 11 times on top of itself with in a 1 kbp region.

2.2.3 Storing/Clustering of SVs

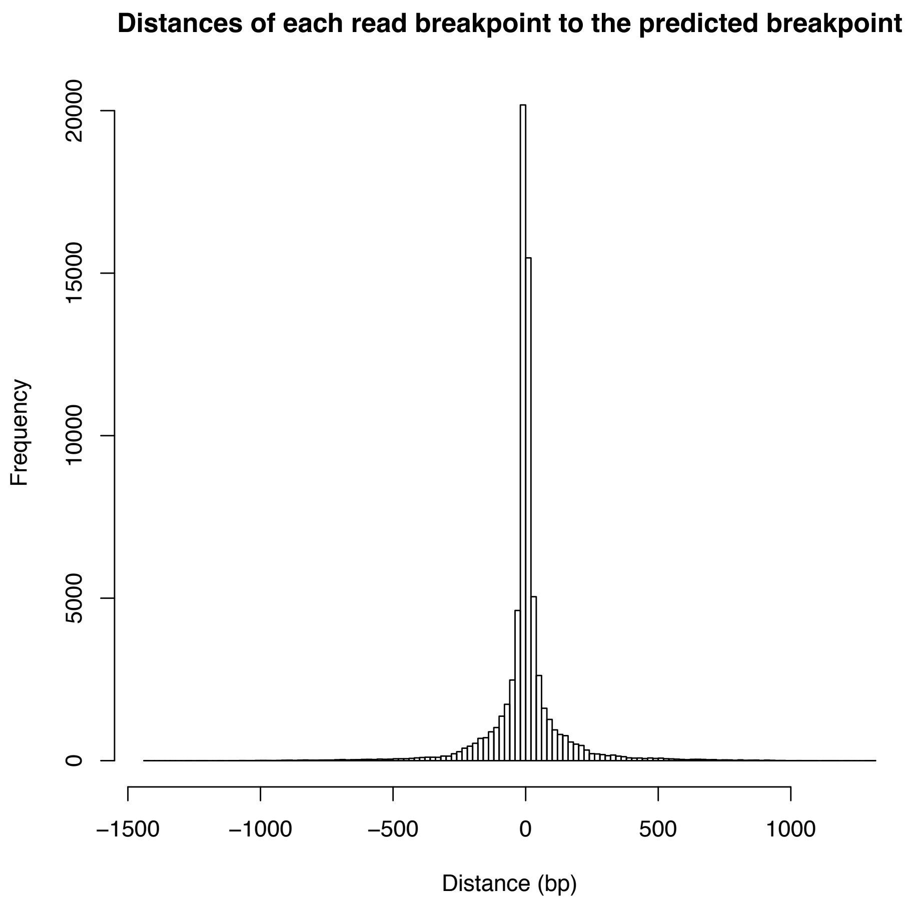
Sniffles use a self-balancing binary tree to store and merge SV calls. Each node in the tree represents a single SV. The SVs are sorted based on the start coordinate of each SV.

Each time Sniffles detects a read that supports a SV, Sniffles traverses the binary tree to see if that particular SV has been observed before. The current SV call is merged with an already known one if their types (e.g. deletion) are the same and their breakpoints are within the maximum distance D. Sniffles automatically infers sensible values for D based on SV length and type (see section 2.2.3.1). In addition, an upper bound for D can be specified by the user (by default: 1kbp). This tolerated distance (D) is necessary to account for imprecise alignments due to sequencing errors or sequence composition (e.g. microsatellites) or non-optimal score function of the mapper itself (**Supplementary Figure 2.8**).

In the tree, each SV is represented by the coordinates that it was first found at. However, the coordinates from other reads supporting the same SV are stored as well. To store the SV type Sniffles uses a set of bit flags to enable a fast comparison between different SVs. Furthermore, the bit flags allow Sniffles to assign multiple types and additional information to a single SV, especially for nested SVs. For complex types, we allow inversions or deletions to be merged with a candidate SV as long as they agree on the coordinates. Furthermore, we allow insertions and tandem duplications to be merged since a tandem duplication is an insertion of the same element next to itself.

To account for multiple overlapping SVs or SVs in close proximity, especially if the genome is polyploid in this region as commonly observed in human cancers or plant genomes, Sniffles implements a more thorough tree search to assess whether the current SV has already been observed. Here, Sniffles starts at the current parental node and walks using an in-order traversal search through the sub tree to identify an already stored SV that would match the current one. Note that this does not significantly increase the runtime, since this procedure will generally only be performed on a very small subtree.

If Sniffles does not find the current SV in the tree, it adds it as a new leaf node. Each SV is stored together with the name of the read it was observed in, the strands, the start and stop position of the genome, the start and stop position on the read, the bit-flag for the type and information about the source (split reads, alignment event, noisy region).



Supplementary Figure 2.6. Breakpoint distribution of clustered read events compared to the estimated breakpoint taken from real data in the SKBR3 data set.

2.2.3.1 Estimation of maximum distance between SVs

In the previous section, we described how Sniffles merges SVs based on their type and the distances between their breakpoints. This is necessary due to imprecise alignment breakpoints that can occur due to sequencing errors, complexity or sequence composition (e.g. microsatellites) or non-optimal score function of the mapper itself. Next Sniffles estimates the maximum distance D that should be allowed between genuine variants. If the distance is too large, it will merge distinct SV calls or spurious SVs stemming from sequencing artifacts in regions with high read coverage and falsely call a SV. Supplementary **Figure 2.8** shows examples of insertions caused by PacBio sequencing artefacts. Conversely, if the distance is too short, a genuine single SV could be called twice with slightly different breakpoints. In addition, read support for these SVs will be lower, potentially causing them to be filtered by the minimum read support threshold.

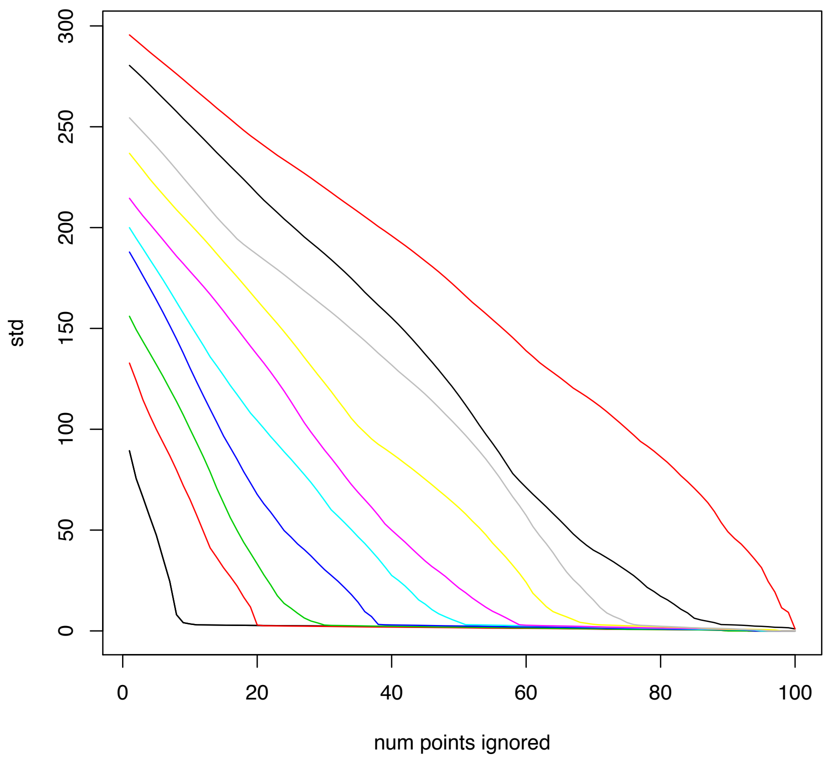
Sniffles attempts to balance these two scenarios, while letting the user decide about the maximum distance (d). We define the estimated length of an SV as length_est = max(length(SV), min_size*2), where min_size is user defined (default: 30bp). This controls for very small events that are harder to place due to e.g. sequencing errors. The allowed distance between two SV are then computed as d’=min(length_est(SV),d). This allows for an adaptive distance threshold while setting the upper bound according to the user. Translocations have an undefined size since they connect two chromosomes together. Thus, d’=d in this case. When comparing two SVs, Sniffles computes the allowed maximum distance (d’) for both SVs and takes the smaller estimate for the comparison to avoid merging two distinct SV with different sizes.

2.2.4 Filtering and summarization

When Sniffles reaches the first read of a new chromosome it triggers the filtering and processing step for all the SVs observed so far. Only translocations are omitted as they might span a chromosome not yet processed by Sniffles. All other SV types identified so far are processed and printed. All translocations are summarized and filtered after the last read of the data set was processed.

For each SV, Sniffles counts the number of supporting reads and compares this to the user defined threshold (default: 10 reads). If the number of supporting reads is at least the threshold, Sniffles computes the most likely start and stop position. This is done by counting the number of reads supporting the same start position. If at least 5 reads share the same breakpoint position, then that value is used. Otherwise, Sniffles reports the average position of the breakpoints. This is done independently for start and stop breakpoints. For insertions Sniffles further computes in the same way the supported length of the insertion.

2.2.4.1 Detection of spurious SVs calls



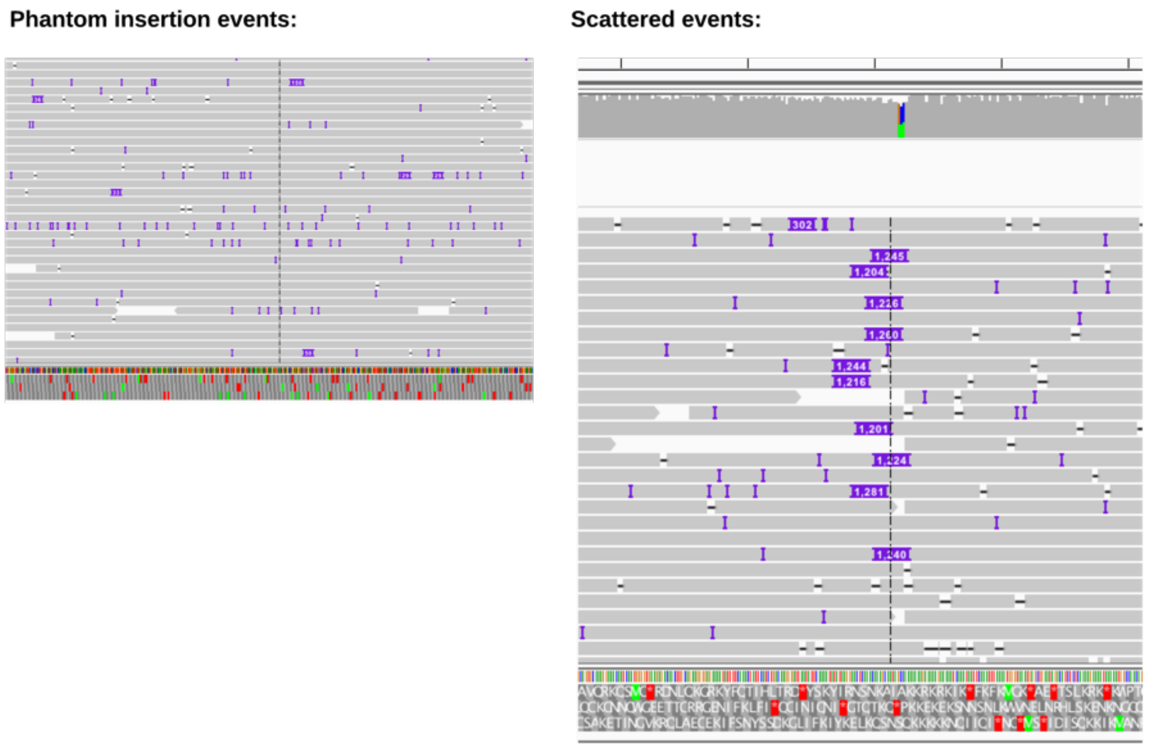
Supplementary Figure 2.7. Simulated breakpoints of a mix of a uniform distribution and a normal distribution. From left to right the concentration of uniform distributed breakpoints increases by 10%. The y axis reports the standard deviation (σ) given that we ignore the most outliers (x axis). In a typical truly identified SVs we would expect some alignments to break earlier or later depending on the sequencing error or sequence complexity at the region. However, a complete artificial signal should have a broader distribution even with a more stringed filtering.

Usually, all the read alignments that support a SV call show very similar breakpoint positions. However, high sequencing error, repeat rich sequences, or low complexity regions of the genome can cause the breakpoints from different reads supporting the same SV to be scattered. One example are short randomly occurring insertions caused by sequencing or base calling errors. Especially in regions with high read coverage, the number of these random insertions might exceed the minimum read support causing Sniffles to report them as genuine SV. To filter out these phantom insertions Sniffles scans for abnormally noisy breakpoints. For a spurious SV call we expect the positions of the read breakpoints to be uniformly randomly distributed. In contrast, for genuine SVs we expect read breakpoints to be normally distributed (with a small standard deviation) around the correct breakpoints of the SV. Thus, we need to distinguish between a uniform/random distribution of read breakpoints and normal distributed read breakpoints even when the number of spanning reads is low. To this end, Sniffles computes the standard deviation (σ) of all read breakpoints that support a SV (separately for the start end the end breakpoints) as follows:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}\ ,$$

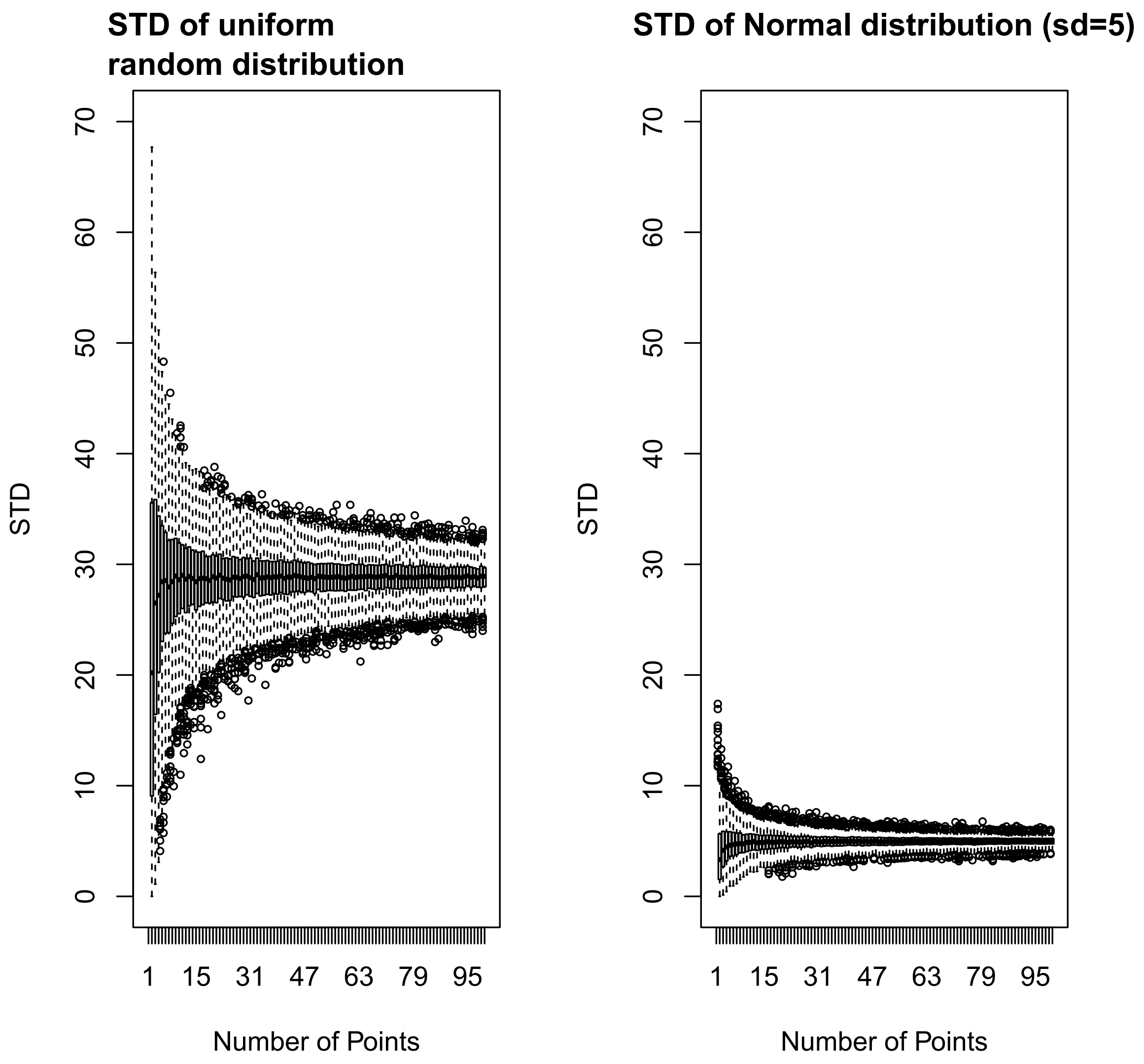
$$where\ \mu = \frac{1}{N}\ (x_1 + \cdots + x_N)\ or\ the\ most\ (n > 5)\ observed\ bp.;$$

The idea is that σ will be small for normally distributed read breakpoints stemming from genuine SVs and large for random read breakpoints. A crucial factor for this analysis is read coverage. To determine how many read breakpoints are required to confidently differentiate between real and random breakpoints, we sampled genomic locations once from a uniform distribution and once from a normal distribution (σ=5) and investigated different sample rations between those two.



Supplementary Figure 2.8. Comparison of phantom insertions (region of ~800 bp) and scattered insertions (region of ~200bp) using IGV. We define phantom insertions as random events that happen due to the base calling of PacBio. In high coverage regions these events sometimes cluster to form a signal. Sniffles detects and erases such calls. We define a scattered insertion as a real SV event, that has imprecise breakpoints. This is often due to the sequencing error occurring near the breakpoints. Sniffles cluster these events together and allows for a wobble on the breakpoints according to the size of the SV.

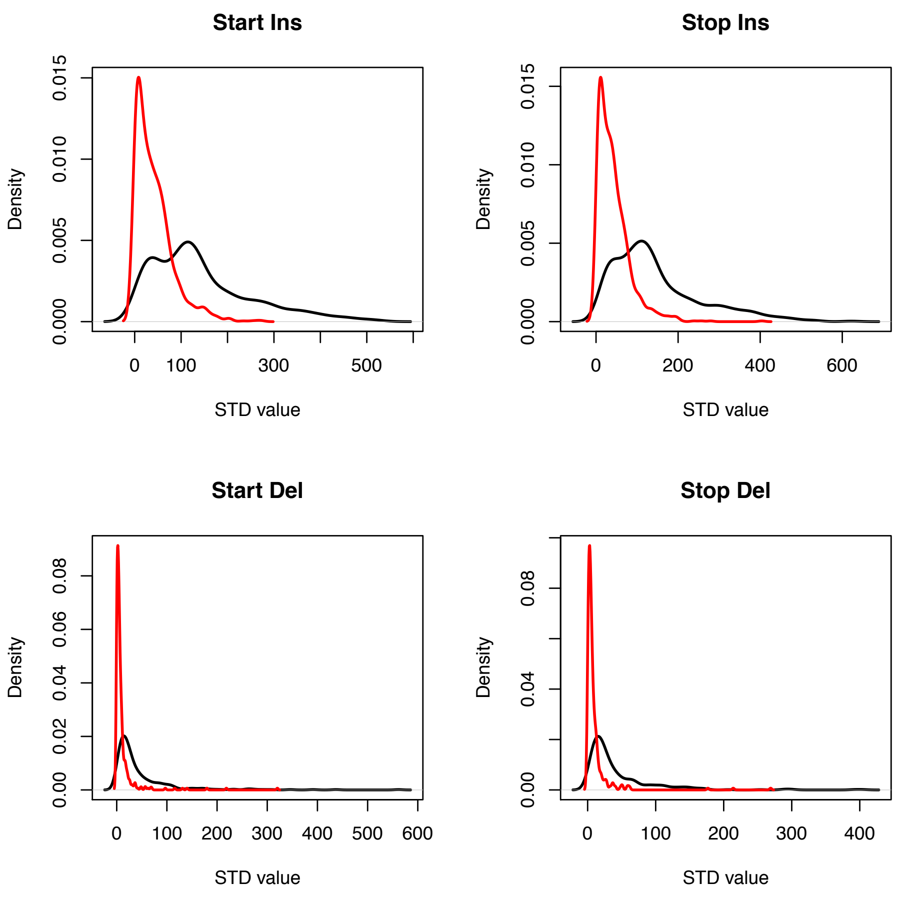
Supplemental Figure 2.9 shows the impact of coverage on the reliability of our standard deviation based filter. Given only 5 or more observed read breakpoints we can reliably distinguish between the uniform and the normal distribution. Thus, Sniffles uses the standard deviation of read breakpoints to filter out alignment artifacts or phantom SVs. However, even for genuine SV we sometimes observe outliers in the read break point distribution. These outliers artificially increase s making it more difficult to distinguish them from random breakpoints. **Supplemental Figure 2.7** shows an example, of different mixtures of noisy breakpoints and precise breakpoints. Here we observe a large impact of already 10-20% of the reads being more disturbed on σ. This has a direct impact on the comparison between the phantom events and real SV as shown in **Supplemental Figure 2.8**.



Supplementary Figure 2.9. Box-and-whisker plot of standard deviation given different number of points/coverage levels from 1-100. Left: Uniform random variables were simulated from an interval of 100bp. Right: Random variables were simulated from a normal distribution with mean=0 and σ =5. Note that both plots converge to the expected σ=28.8 for the uniformly random distribution and σ=5 from the normal distribution. For each point we run the sampling 1,000 times. Box plots where generated using the default values from R.

To account for this, we do a 1st and 4th quintile filtering. **Supplementary Figure 2.10** shows the difference between *σ* with and without quintile filtering for genuine SVs from the SKBR3 data set. If the set of breakpoints is originally approximately a random/uniform distribution, σ will not change when doing quintile filtering (see **Supplementary Figure 2.4**). However, for real SVs with a small fraction of outlier breakpoints σ will dramatically reduce (even to 0) when applying out filter (see **Supplementary Figure 2.7**).

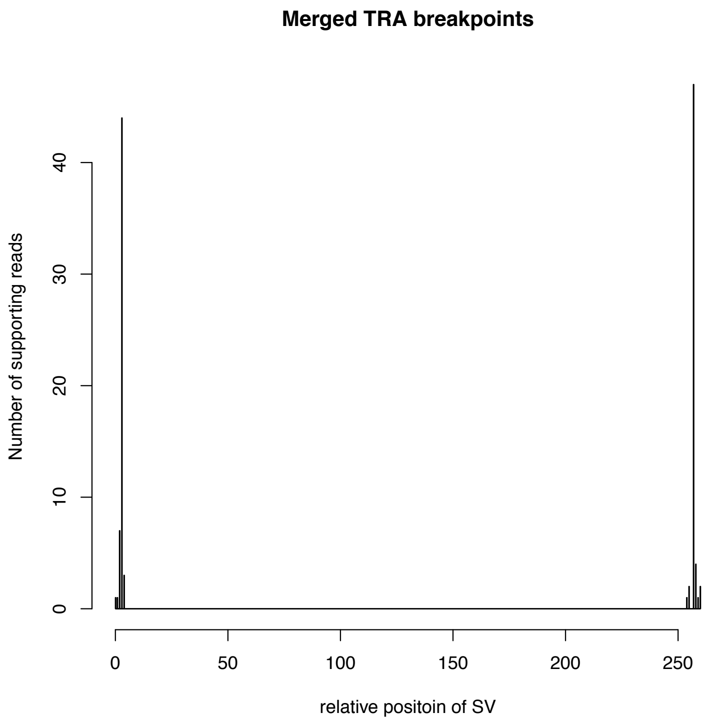
Next we need to compare the so trimmed-*σ* to an expected *σ* given the type and length of the SV. Here, we use a uniform distribution, with σ equal to the length of the region (d') * sqrt(1/12). Since we compare it to the σ of a trimmed distribution, we need to correct this threshold accordingly. Over simulations of random variables within d' iterating 1000 times we identified a ratio of filtered vs. not filtered of 1.99949. Thus, a SV is filtered out when the filtered std > d'*(sqrt(1/12)/2) as this is most likely a set of spurious events rather than a true SV.



Supplementary Figure 2.10. Filtered (red) and unfiltered (black) breakpoint distribution over insertion and deletions on chr17 of the SKBR3 data set. As one can see the filtering enables a more precise prediction of noise vs. precise events.

2.2.4.2 Detection of falsely merged SV calls

In section 2.2.3 we described how Sniffles merges SVs of the same type and with breakpoints in close proximity to account for sequencing error, repeat rich sequences, and low complexity regions. However, sometimes this causes two separate SVs in close proximity to be falsely merged. To detect such falsely merges SV calls, Sniffles evaluates if the read breakpoint positions associated with a single SV follow a bimodal distribution. **Supplementary Figure 2.11** shows such a case for two merged translocation breakpoints. There are multiple ways to test for bimodal distributions such as evaluating the skewness and kurtosis [6,7] or Hartigan's Dip Test Statistic for Unimodality [6] etc. Nevertheless, the statistics require a minimum number of observations (read breakpoints in our case) to be reliable, which is higher than the read coverage in a typical dataset. Therefore, they are not applicable here, and instead we apply a heuristic that we have found performs well in practice.



Supplementary Figure 2.11. Example of two translocation breakpoints merged to one event leading to a bimodal distribution in the breakpoint position.

Our heuristic approach first clusters the breakpoints within the minimum size of an SVs together. Next, Sniffles counts the number of clusters that have more breakpoints clustered than the minimal number of reads supporting a SV. If this is the case, we replace the original entry with two new entries according to the clustering.

2.3 Genotyping

To infer genotype and allele frequency for SV calls, Sniffles also needs to access information about reads that do not support SVs. Therefore, while scanning the alignments Sniffles also records the start and location of reads that do not support a SV, but otherwise match the filter requirements (score ratio and MQ quality) in a separate binary file. After all alignments are scanned, Sniffles analyzes the file of read placements and check for every read if it overlaps one or more SV calls and a read counter is increased for the respective SV call. This is efficiently computed using Sniffles self-balancing binary tree that is used to store all SV calls. Finally, Sniffles uses this information to estimate the genotype based on the allele frequency of the SVs and prints the result to the VCF file.

2.4 Phasing and read clustering

If the phasing/read clustering mode is activated, Sniffles records the number of reads that support each SV as well as their read IDs. This is computed using a hash table with the read IDs as key and the variant IDs as value. In addition, Sniffles scans the hash table and groups all variants spanned by a single read. This is computed by storing the lowest observed variant ID for each read in cases there are multiple variants assigned to one read name. For each so detected variant, Sniffles updates the hash table to set the variant ID to the minimum observed variant ID of the subgroup to indicate their clustering. In addition, Sniffles stores the number of reads that carry the same variant IDs. Given a user definable threshold (default: 1) on the minimum number of reads supporting the same two variants, Sniffles cluster the variant IDs together. The clustered IDs are indicated by the same number plus a '_' with a running number from 0 to the number of variants associated to aid in downstream analysis and further phasing.

2.5 Table of Sniffles Parameters

Parameter	Default	Description
-m/ --mapped_reads	NA	Sorted .bam file either form NGMLR or BWA-MEM (please use -M flag during mapping with BWA)
-s/ --min_support	10	Minimum number of reads that support a SV to be reported.
--max_num_splits	7	Maximum number of split segments a read may be aligned before it is ignored.
-q/ --minmapping_qual	20	Minimum mapping quality value of alignment to be taken into account.
-l/ --min_length	30	Minimum length of SV to be reported.
-v/ --vcf	NA	Name of the vcf file to be reported
-b/ --bedpe	NA	Name of the bedpe file to be reported
-d / --max_distance	1kb	Maximum distance to group SV together. Sniffles also estimates this parameter during runtime to group together SVs reported by different reads.
-r/ --min_seq_size	2kb	Ignores reads that only report alignments with not longer then bp.
--tmp_file	NA	Optional file prefix to write tmp files to.
-n/ --num_reads_report	0	Number of read names to be reported that support the SV in the vcf file.
--cluster_support	1	Number of overlapping reads before grouping two SVs together.
--cluster	false	Performs read based phasing to mark SVs that occur together.
--genotype	false	Performs genotype estimation.
--ignore_std	false	Ignore filtering by standard deviation
--version	NA	Shows the current version.
-h/ --help	NA	Shows the help message.

3. Read and Structural Variation Simulations

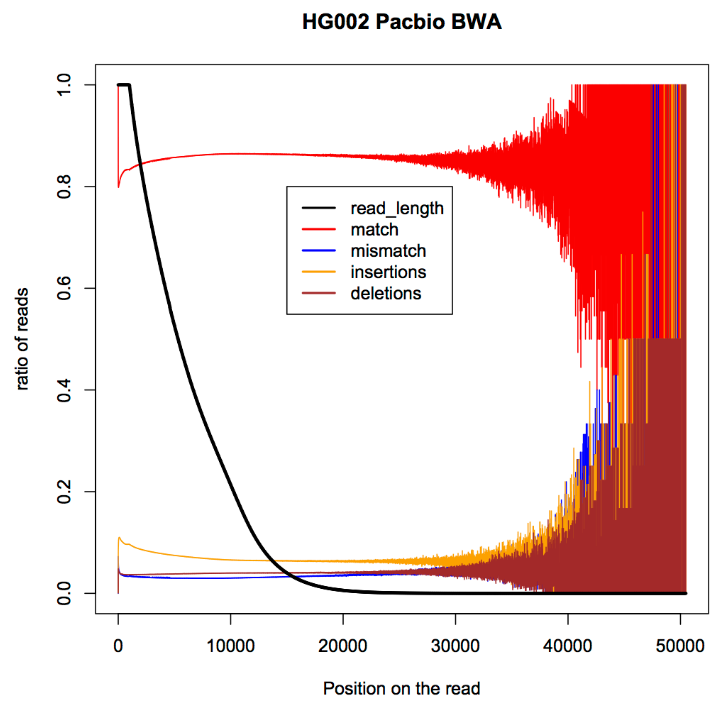
3.1 SV Simulation

We used our toolkit SURVIVOR[8] to simulate SVs (insertions, inversions, deletions, duplications, translocations). We also extended the simulator to introduced nested events such as an inversion flanked by two deletions or a tandem duplication which has one copy inverted. For the evaluation, each breakpoint is treated as a separate event, e.g. a caller must call the inversion and the two deletions separately to be precisely correct.

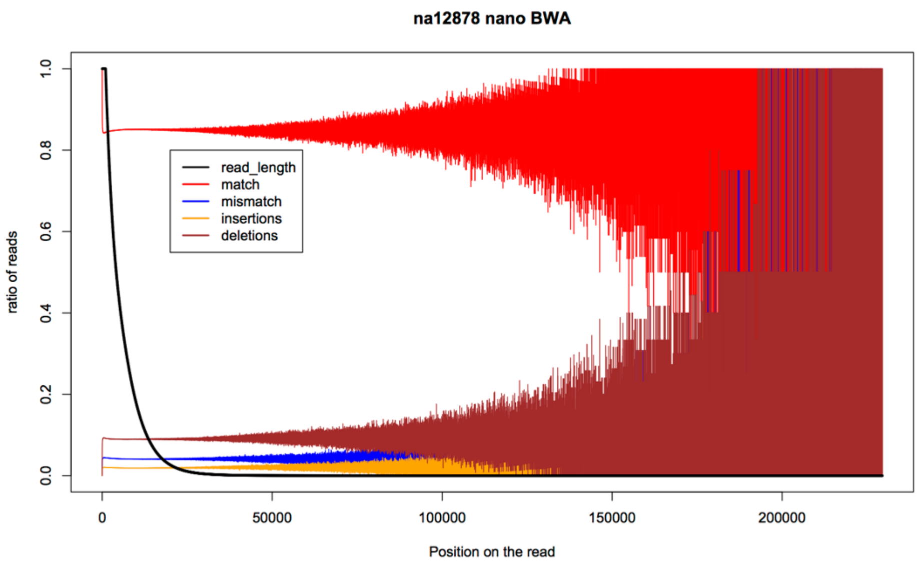
3.2 Long read simulation

We extended SURVIVOR with an error profile generator and read simulator that can generate long reads based on an error profile and a specified coverage. The first module scans the aligned reads. For this analysis we analyzed the BWA-MEM alignments for PacBio: HG002 from GiaB and for Oxford Nanopore: NA12878 requiring a minimum length of 1kb of the alignments. SURVIVOR measures the ratio of reads showing a deletion, insertion, match or mismatch per read position. Furthermore, it records the number of reads that overlapped with each position in the error profile. Thus, we obtain the probability that a read stops after a certain length and the error probabilities of each position. **Supplementary Figure 3.1** and **3.2** shows the error profiles for Oxford Nanopore and PacBio data, respectively. We note that the error profiles become highly variable towards the end because of the limited numbers of reads available for sampling.

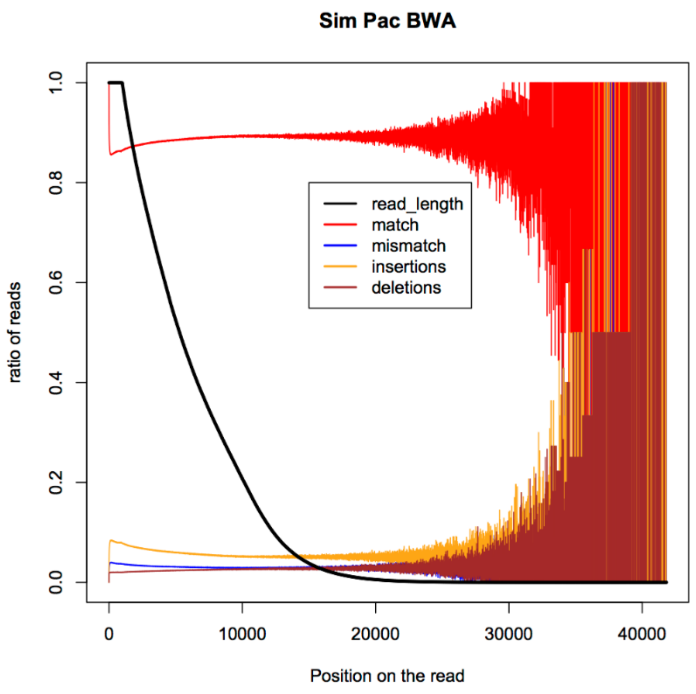
Next, the second module simulates the reads based on the error profile and the reference genome. It computes the number of reads given the median read length and start to simulate reads. For each read, chromosome and position of interest are chosen randomly. A new position is being chosen if the region of the read happens to be across a stretch of N's. We allow up to 10% of the read being N's. The so selected subsequence is then altered given the before generated error profile per position and its length is determined by the recorded distribution in step 1. Each insertion and deletion is only 1bp long to simulate sequencing errors. Supplementary Figure 3.3 and 3.4 shows the re-measured error profiles from simulated data.



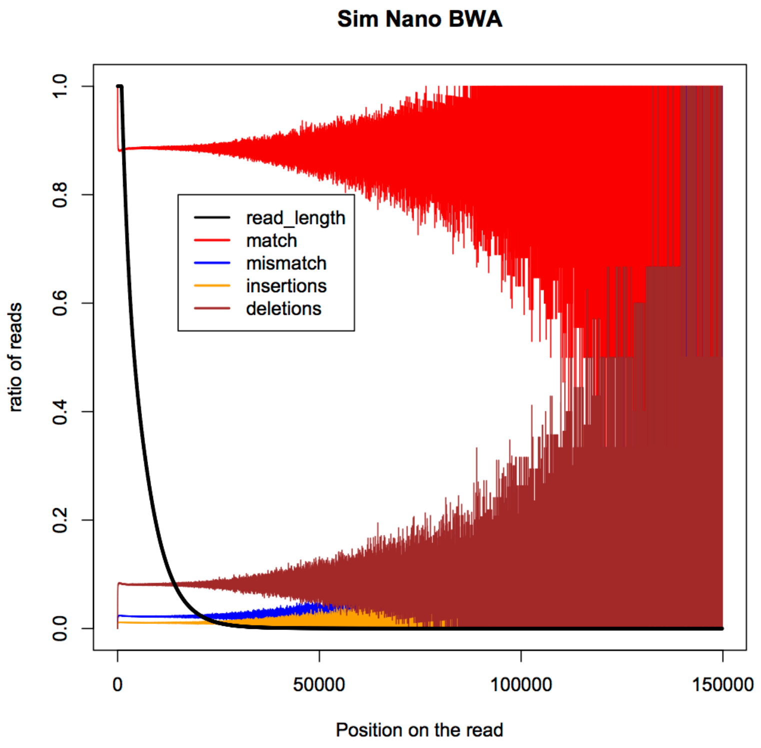
Supplementary Figure 3.1. Measured error profile from PacBio using HG002 from GiaB using BWA-MEM alignments.



Supplementary Figure 3.2. Measured error profile from Oxford Nanopore over NA12878 using BWA-MEM alignments.



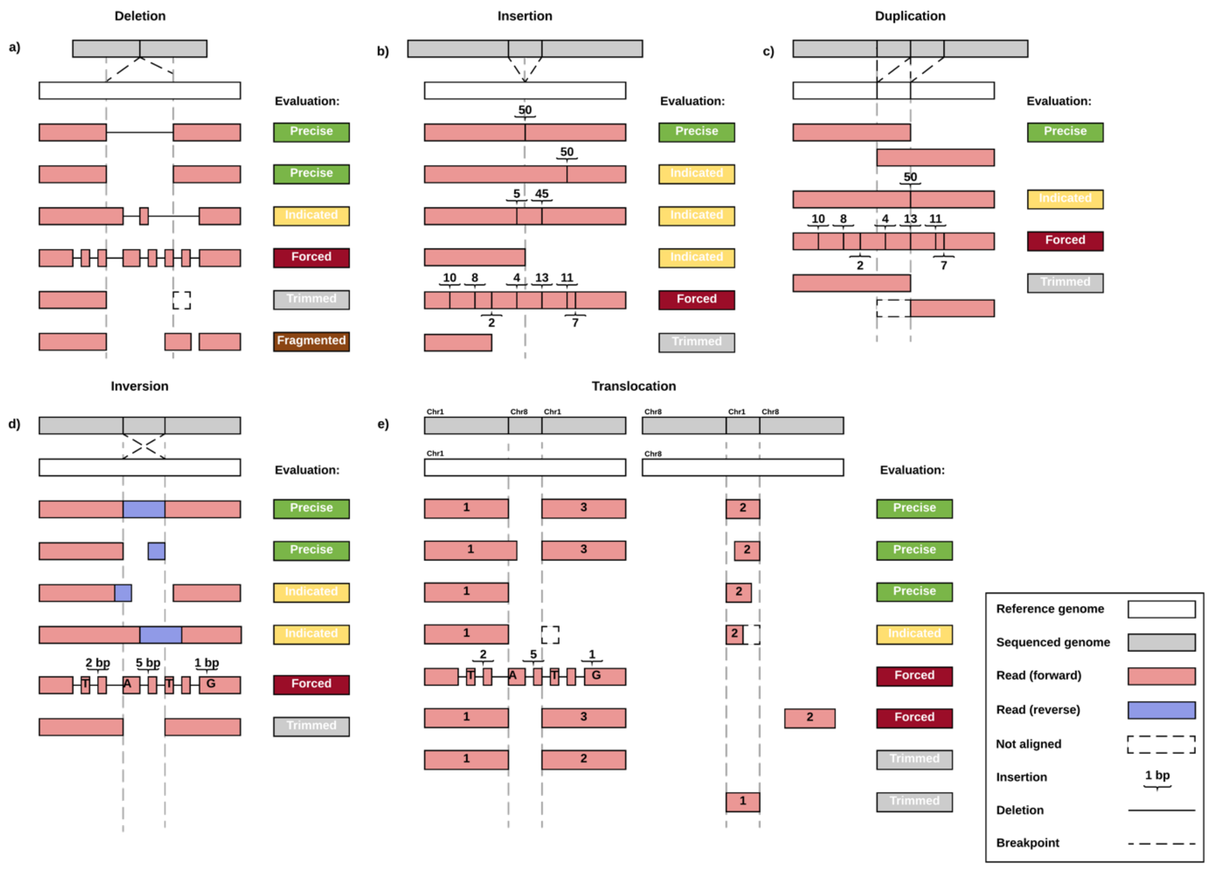
Supplementary Figure 3.3. Error profile of simulated PacBio data based on error profile measured in Supplementary Figure 3.1.



Supplementary Figure 3.4. Error profile of simulated Oxford Nanopore data based on error profile reported in Supplementary Figure 3.2

3.3 Alignment and SV accuracy characterization

As described in the methods we characterized reads mapping across SVs as being *precisely*, *indicated, trimmed, forced, fragmented* and *unaligned*. Supplementary Figure 3.5 summarizes the evaluation and categories.



Supplementary Figure 3.5. Evaluation schematic for the long read mappers used in this study over simulated reads and SVs.

4. Datasets

4.1 Simulated data

We used SURVIVOR to simulate PacBio-like long reads with an average length of around 21kbp. We simulated data sets that hold only one specific type of SVs and a specific average length of the SV. Specifically, we simulated a size range of 100bp, 250bp, 500bp, 1kbp, 2kbp, 5kbp and 10kbp, and for each data set we simulated a total of 20 SVs always given the same type (Indel, inversions, duplications, translocation, inverted duplications, inversion flanked by deletions). For translocations we simulated a region of a defined size (e.g. 250bp) to be swapped with a region of the same length from a different chromosome. To simulate INVDEL variants we simulated an Inversion flanked with two deletions that are 10% of the total size. Thus one data set consists of 20 inversions and 40 deletions.

For each data set we simulated 347,538 PacBio-like long reads using SURVIVOR; 39.8 million Illumina like 100bp paired end reads with an insert size of 500bp using Mason[9] and 662,943Nanopore reads using readsim (version 1.6) with an average of 6kb read length similar to that data from Jain, et al. [10]

Furthermore, in a separate analysis, we induced SVs into the reference genome. This way we fully preserve the characteristics of the sequencing technologies. For Pacbio we used the data set of NA12878 [11] and modified the human genome we mapped to using SURVIVOR. We introduced 140 inversions, 140 translocations and 140 indel events, each reaching between 100bp and 3kbp. For Nanopore we utilized the reads from NA12878[10] mapping the reads to the same modified human genome. For Illumina, we used the publicly available dataset ERR19414

4.2 Coverage based filtering of Arabidopsis CVI:

We used Sambamba (v0.6.6) [12] to compute the base pair coverage across the CVI PacBio mapping. Next we used SURVIVOR (option 23) to identify and cluster regions with zero coverage within 10kb intervals. We excluded these regions from further analysis in the Col-0 x CVI F1, as the zero coverage indicates these regions are specific to Col-0.

4.3 Runtime comparison over NA12878

We benchmarked the different long read mappers used in this study on a PC equipped with 4 x AMD Opteron 6348 Processors with each having 12 cores and 512 GB RAM in total. Every program was executed with 10 threads over the 1x subsampled NA12878 Nanopore[10] and 1x Pacbio[11] NA12878 data set. Minimap2 was the fastest with 546 seconds, MECAT required 1,236 seconds while NGM-LR required 4,788 seconds (1.3 hours) followed by BWA-MEM requiring 6,133 seconds (1.7 hours). BlasR required 18,518 seconds (5.1 hours). We stopped GraphMap after running for a week without finishing. GraphMap required also the most memory (106.9 GB), whereas BWA-MEM required the least (6.0 Gbytes). NGM-LR required 10.2 GB to map all the reads. We also benchmarked Sniffles on the full data sets mapped by NGM-LR: Sniffles required 12,102 seconds (3.3 hours) for the 44x Pacbio data set, and 7,744 seconds (2.2 hours) for the 28x Nanopore data set. Supplementary Table 10 shows all the results including the memory consumption for all mappers.

4.4 NA12878 comparison of published data sets.

We have obtained the calls set (NA12878.wgs.illumina_platinum.20140404.svs_v2.vcf) previously descried from the Platinum genomes project and the 1000 genomes project for NA12878. The SVs are currently hosted under dbGaP: phs001224.v1.p1 (also see https://www.illumina.com/platinumgenomes.html). The data set consisted of 1802 deletions.

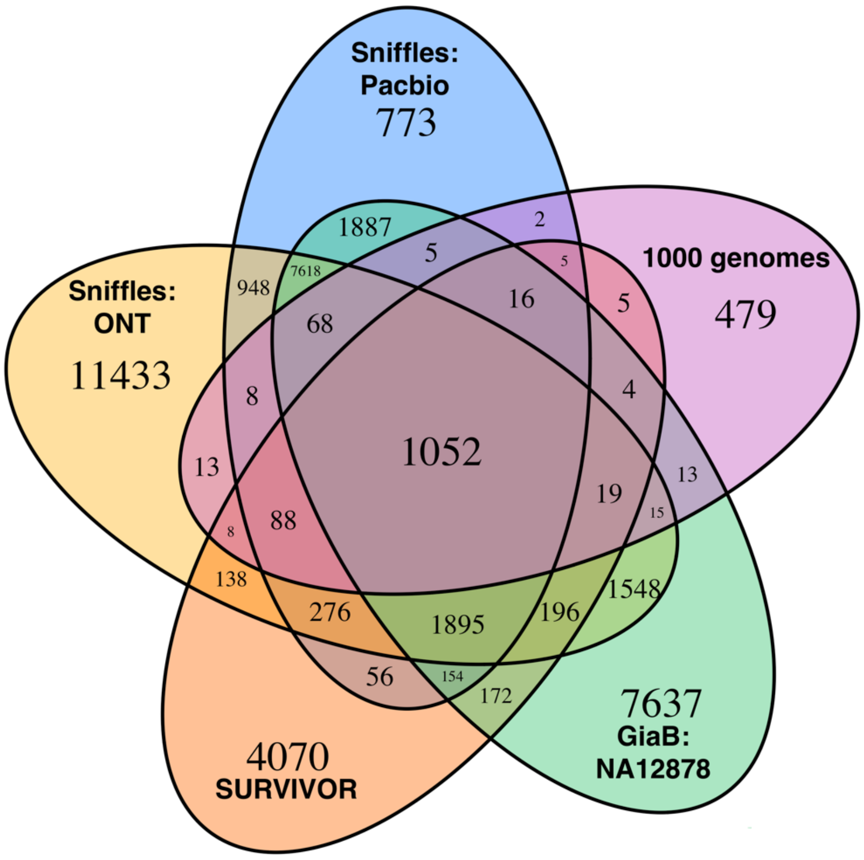
Furthermore, we compared the results obtained here with the GiaB calls from 7 different SV callers from here:
ftp://ftp-trace.ncbi.nih.gov/giab/ftp/data/NA12878/NA12878_PacBio_MtSinai/

This file included 29,922 SVs (12,740 deletions and 17,182 insertions) of 50bp or larger. The results were compared using SURVIVOR with option 5 allowing for a 1kbp distance and type unspecific overlapping. Awk was used to obtain a Venn diagram (**Supplementary Figure 4.1**) and perform the comparisons.

4.5 Comparison of short read based calls vs. Sniffles calls based on NA12878

4.5.1 Assessment of indels

We customized the script (pairend_distro.py) provided in the Lumpy[13] package to obtain the mean and standard deviation of insert sizes across the entire Illumina data set. The mean insert size was 311.70. Using SURIVOVR we converted the insertion and deletions with a length of 50bp to 3kbp from the VCF file to a BED file containing the chromosome, start and stop coordinates of the PacBio-based or Oxford Nanopore-based SVs. Note that the stop coordinate is with respect to the reference genome, so does not contain the length of the insertion. We then identified reads within 300bp of the start and end breakpoint of an insertion and deletion, and computed the mean and standard deviation of the insert size for each region. These were tested for a significant deviation compared to the global average insert size using a two sided, one sample t test. We considered indels achieving a p-value <0.01 as significant and thus summarized the results based on this threshold.



Supplementary Figure 4.1: Venn diagram of the NA12878 comparison between different call sets.

4.5.2 Assessment of Illumina based translocations.

Using SURVIVOR, we identified translocation calls identified by at least 2 of the short read callers. During manual inspection in IGV, we noted that many of these translocations either overlapped with insertions called by Sniffles or very short insertions with a length just below the Sniffles size cutoff of 50 bp. To count how many of the SURVIVOR translocation overlapped with insertions and other SV called by Sniffles, we reran Sniffles with –l 10 (to obtain a list of SVs of 10bp or larger) as well as –s 5 (to allow SVs to be reported also if only 5 reads supporting them) and converted the SURVIVOR calls to BED format. In the BED file, each translocation was represented by two 400bp intervals (accounting for the insert size of the Illumina data) centered around its break points. Next, we split the Sniffles calls into 5 separate VCF file, where each file contained only one of the following SV types: translocation, insertion, duplication, deletion, inversions. We used bedtools[14] to identify if at least one of the break points of a translocation called by SURVIVOR, overlapped with SVs in one of the five Sniffles VCF files. If a SURVIVOR translocation overlapped with more than one Sniffles call we counted only the first overlap we found. For example, for a SURVIVOR translocation that overlaps with a translocation and an insertion called by Sniffles we only counted the overlap with the translocation.

For the remaining SURVIVOR translocations that did not overlap with a Sniffles call, we checked whether they overlapped with a repeat annotation or a region with substantially increased read coverage. To this end, we computed the coverage for the 400 bp regions center around the translocation break points using bedtools multicov. As a baseline we used the coverage of randomly shuffled 400bp intervals (bedtools shuffle). We considered a translocation as falling in a high coverage region, if at least one of its breakpoints shows a higher coverage than all the random intervals.

4.6 Human subsampling experiments

The theoretical assessment is implemented in SURVIVOR (option 34), which simulates a genome with a specified coverage and read length and tests if a minimum X number of reads overall with at least Y positions a given SV breakpoint. We fixed the number of SVs (700), the minimum overlap of a read across a SV (50bp) and the minimum number of 5 reads required.

We used seqtk (version: 1.1-r93-dirty) to9 subsample the raw read data to 5x, 10x, 15x, 20x, 30x using the average read length of 4,334 and 9,872 for NA12878 and SKBR3 Pacbio data sets, respectively. For Nanopore we used the average read length of 6,432bp. Next, we mapped the subsampled read files using NGM-LR and called SV using Sniffles with different parameters for minimum read support (s: 1,2,3,4,5,6,7,8,9,10). These SVs calls as well as the original SV call set created over the full coverage (s:10) were filtered for alternative contigs calls and compared using SURVIVOR[8] (option 5) requiring min 50bp and a maximum of 1kb in breakpoint distance. We then summarize the number of calls per threshold as well as the number of recalled SVs compared to the original coverage.

5. Evaluation of NGMLR and Sniffles

5.1 Simulation and Evaluation over simulated SVs

5.1.1 Evaluation of NGMLR on simulated human data

Using the error profiles and read lengths measured from two human datasets (**Supplementary Section 3.2**), we simulated 50x PacBio-like and 50x Oxford Nanopore-like read data sets from two human chromosomes (chr21 and chr22). In the simulation, we included a total of 840 homozygous SVs consisting of equal numbers of indels, duplications, balanced translocations, and inversions ranging from 100bp to 50kbp in size (**Methods**). **Figure 3a** summarizes the results when evaluating NGMLR, BWA-MEM, BLASR, GraphMap, LAST, MECAT, and Minimap2 aligning these reads to the entire human genome[24-26]. Each bar represents one data set consisting of 20 SVs of a certain type and length, and categorizing the read alignments as: *precisely* capturing the breakpoints (+/- 10bp) and the correct type of the SV (green); *indicating* the type but without exact break points (yellow); *trimmed* so that the region of the read containing the SVs was not aligned (gray); *forced*, such as the BWA-MEM alignments in **Figure 2** (red); *fragmented* so that a read is split more often than necessary (brown); or the entire read was *unaligned* (white) (**Methods** and **Supplementary Table 1**). Across all SV types, NGMLR outperforms the other mappers with an average 80.32% precisely aligned versus 52.77% for Minimap2, 51.68% for LAST, 26.31% for BWA-MEM, 17.82% for BLASR, 9.76% for MECAT and 5.70% for GraphMap. Even when counting the precise and the indicated representation together, NGMLR outperforms with an average 91.83% versus 69.43% for Minimap2 as next closest competitor. Other than NGMLR, essentially all of the other aligners performed poorly with the alignments near SVs (See **Supplemental Figure 5.1** for an example from MECAT and **Figure 2** for an example from BWA-MEM.

Next, we compared the performance of NGMLR, BWA-MEM, GraphMap, Minimap2, and LAST in mapping simulated Oxford Nanopore-like reads, using their respective parameter suggestions (BlasR and MECAT were excluded, as they are only applicable to PacBio reads). Again, NGMLR substantially outperformed other mappers for *precisely* aligning reads (72.42% vs 51.13% for the second best Minimap2), or when considering both *precise* and *indicating* alignments (88.57% versus 69.04% for Minimap2) (**Figure 3c**). LAST was the next most accurate aligner (44.40% *precisely* aligned), followed by BWA-MEM (24.89% *precisely* aligned). GraphMap performed rather poorly on these data, with on average only 18.19% of reads aligned *precisely* or *indicating* the SV as it *forces* 61.13% the reads to align across the SV.

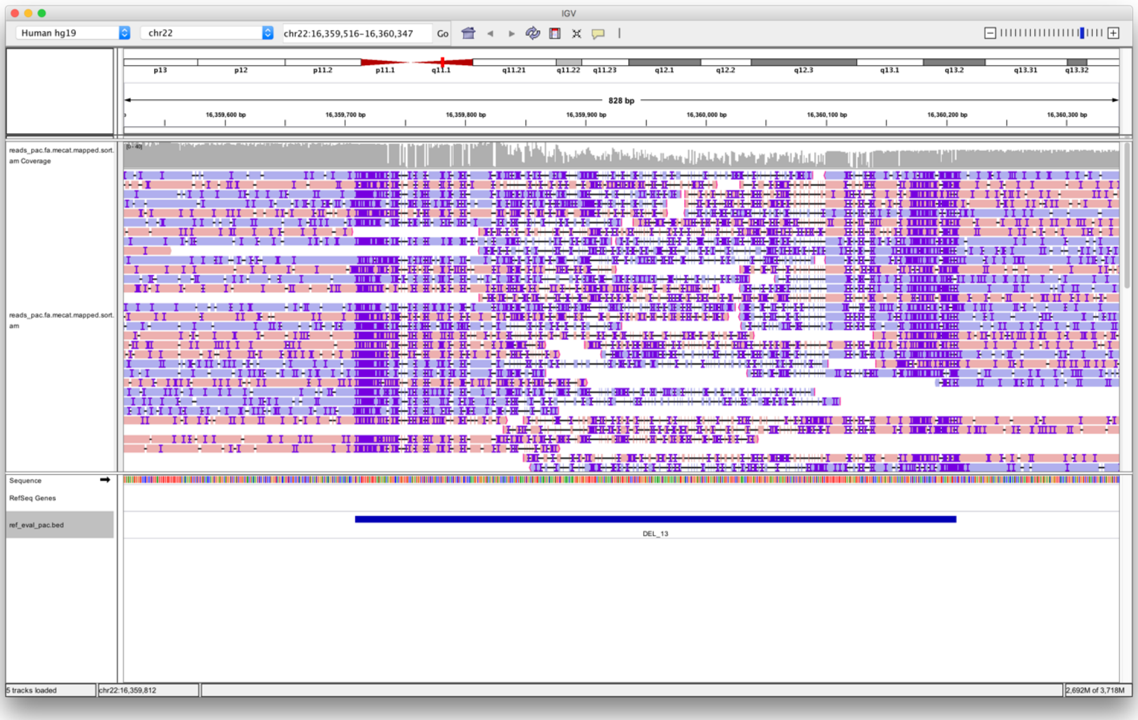


Figure 5.1: IGV screenshot from MECAT aligning reads through a deletion.

5.1.2 Evaluation of Sniffles based on simulated human data

Next, we evaluate the performance of Sniffles compared to alternate short and long read SV detection approaches using the alignments reported above [14-16,18] (**Figure 3b**). We were able to use Sniffles with either NGMLR or BWA-MEM, but the output formats for the other aligners are not compatible with Sniffles. This is because the SAM/BAM format is currently not well resolved for very long reads so some of the tools have been adopting incompatible formats and/or renaming the reads with new identifiers. We also extended the analysis to include simulated short reads to be analyzed by our consensus algorithm SURVIVOR[8]. SURVIVOR aggregates the outputs from Lumpy, Manta and Delly and excludes variants reported by only a single caller. We find this increases specificity without sacrificing much sensitivity[8]. Similar to the read alignments, we classified SVs to be: *precisely* detected if they are reported within +/- 10bp (green); *indicated* if they are within +/- 1kbp and ignoring the type (yellow); *not detected* (red); and *false positive* (brown) (**Methods** and **Supplementary Table 2**).

Over all SV types, the combination of Sniffles and NGMLR performs the best with an average of 94.20% *precisely* detected SVs and an FDR of 0.00%. The most problematic class was short (100bp) tandem duplications, as they are identified as insertions rather than tandem duplications, and hence classified as *indicated*. The second best result was achieved using Sniffles with BWA-MEM alignments, with on average 79.11% *precisely* detected SVs and a 0.68% FDR. With the more noisy BWA-MEM alignments, Sniffles detects the presence of an SV, but cannot reliably predict the exact location or sometimes even the type of SV. For example, both deletions and inversions cause an accumulation of mismatches in the BWA-

MEM alignments (**Figure 2)**. PBHoney, which relies on BlasR alignments, *precisely* detected only 32.58% of simulated SVs and missed 25.18%. Most of the 40.73% *indicated* SVs from PBHoney came from misinterpreting tandem duplications as insertions. For the short-read analysis, SURVIOR detected 18.81% as *precisely* and 57.89% as *indicated* of the simulated SVs, similar to what has been previously reported for short read analysis[6,8], although the consensus-based analysis reduced the FDR to 0.17%.

Finally, we benchmarked the performance of Sniffles using BWA-MEM and NGMLR on the Oxford Nanopore-like reads described above (**Figure 3d)**. Using Sniffles with NGMLR, 82.25% of SVs are *precisely* identified, whereas 76.35% are *precisely* identified with BWA-MEM. Nevertheless, due to the higher rate of sequencing errors in the Oxford Nanopore-like data, Sniffles using either aligner has a slight FDR of calling 1-4 additional events per data set.

5.1.3 Benchmarking NGMLR and Sniffles with genuine long human reads

The simulated read results establish a baseline of performance, although may not capture the full complexity of real sequencing data. To benchmark more realistic datasets, we next analyzed genuine PacBio[35] and Oxford Nanopore[36] reads from the well-studied NA12878 human genome. Since a complete truth set of SVs is not available for this genome, we modified the reference human genome to introduce 700 homozygous SVs at random locations: 140 insertions (by deleting from the reference), 140 deletions (by adding new sequence), 140 inversions, and 140 balanced translocations creating 280 translocation events. The mean indel and inversion size was 1.6kb. We did not attempt to simulate tandem duplications, as this would require detecting and modifying tandem duplications preexisting in the reference.

In this analysis, we can only evaluate the sensitivity of alignments, but not false positives since there are additional true SVs in the sample compared to the reference. NGMLR showed a clear improvement over BWA-MEM (58.65% vs 32.35%) for *precisely* aligned reads across the SVs (**Supplementary Table 3**), although the shorter average length of the genuine reads limited the number of reads that could be *precisely* aligned. For example, if an insertion is longer than the read length, then the read can only *indicate* the SV. When counting *precise* and *indicated* together, NGMLR achieved a substantially better result than BWA-MEM (76.96% vs 49.21%). Furthermore, NGMLR considerably reduced the number of *forced aligned* reads compared to BWA-MEM (3.01% vs 24.21%). Using the Oxford Nanopore reads from NA12878 we observe a similar trend with NGMLR giving the most *precise* alignments (51.56% vs. 27.35%) with the lowest percent of *forced* reads (5.94% vs. 29.15%).

Using these alignments and the alignment of 50x coverage of genuine Illumina sequencing from this sample[6], we next benchmarked the available SV callers (**Supplementary Table 4)**. Sniffles and NGMLR achieved the highest rate of *precisely* called SVs with 95.14% and 88.29% SVs using the PacBio and Oxford Nanopore reads, respectively. In contrast, the short read-based SURVIVOR analysis had a much lower sensitivity (76.57%) considering either *precise* or *indicated* variants.

5.2 Trio-based analysis of Structural Variations

Next, we focused on a trio based analysis using genuine sequencing reads as the simulated structural variations may not capture the full complexity of true variants.

5.2.1 Assessment based on PacBio sequencing of an *Arabidopsis* trio

The first trio was of the model species *Arabidopsis thaliana* (Col-0, CVI and the Col-0 x CVI F1 progeny) previously sequenced[37]. This is a particularly challenging dataset as the rate of heterozygosity in the F1 is approximately 1 SNP every 200bp along with thousands of reported SVs [37]. Using Sniffles with default parameters, we identified 355 SVs in the reference strain Col-0 and 9,652 SVs in CVI (**Table 1)**, of which 42 (Col-0) and 6,679 (CVI) were homozygous. Based on Mendelian inheritance, we expected all homozygous SVs identified in the parental cultivars to be in the F1 as heterozygous variants. Indeed, when comparing the homozygous calls from Col-0 to the F1 only 4 SVs were not identified. On closer inspection, one missed insertion was reported as 47bp in F1 vs. 53bp in Col-0, and similarly a deletion was reported as 48bp in F1 vs. 53bp in Col-0. Both of these events were initially not found due to the minimum size cutoff of 50bp. Sniffles can detect the remaining two SVs – another deletion and a duplication – in the F1 by reducing the coverage threshold as the deletion was supported by only 4 reads and the duplication by only 3 reads.

When comparing CVI to the F1 calls, Sniffles initially missed 370 (5.54%) SVs that were reported in CVI and not in the F1. Most of the missed variants are explained by a few straightforward explanations: 159 lacked sufficient coverage of supporting reads in the F1; 101 had slightly different SV sizes reported below the minimum size; 43 were interpreted as different SV types; and 50 occurred within Col-0 specific regions in F1 (**Supplementary Section 4.4**). After considering these factors, only 17 (0.25%) SVs present in the CVI data set were missed by Sniffles for the F1 data set. In contrast, the Illumina-based SURVIVOR calls in the F1 data set had a much lower recall rate compared to the PacBio-based Sniffles in Col-0 (47.3% recall) and CVI (70.6% recall).

Next, we compared the SVs identified in the F1 and not found in the parents either due to missed calls in the parental genomes (false negatives) or additional calls the F1 (false positives). For Sniffles, it identified 767 SVs that were unique to F1, which is an inconsistency rate of 7.22%. Upon closer investigation, we identified the major cause of the differences to be the shorter read lengths of the parent sample that caused certain repetitive regions to have lower coverage. When adjusting this coverage parameter to a minimum of 5 reads, the inconsistency rate dropped to 3.36%. When further allowing for larger distances (10kbp) to group SVs together the inconsistency dropped further to 1.21%.

5.2.2 Genome-in-a-Bottle (GiaB) Human Trio Analysis

Next, we investigated the performance of Sniffles based on the human Ashkenazi trio data set from GiaB[38] (**Table 1 and Supplementary Table 6)**. Similar to Arabidopsis, we analyzed the concordance of Mendelian inheritance between samples as an indicator of performance, although some SVs (e.g. mobile element insertions in the son) may be incorrectly classified. We adjusted the coverage threshold for Sniffles to a minimum of 5 reads (-s 5) to account for the reduced coverage of the parents compared to the son (32x compared to 69x, also see downsampling results below). We compared these results to the Illumina-based call sets from 80x coverage in all of the samples.

Sniffles reported 5,244 and 5,964 SVs as homozygous in the father and mother, respectively. Within the son we re-identified 93.84% and 94.01% of the SVs from the father and the mother, respectively. Most of the missed variants could be explained through minor adjustments in parameters. For example, when we relax the size cutoff to consider variants just below 50bp, Sniffles misses only 187 (3.57%) and 126 (2.11%) for the father and mother, respectively, and most of the remainders have slightly less coverage than our cutoff. In contrast, when using SURVIVOR, we identified only 1,586 and 1,668 homozygous SVs for father and mother, respectively, approximately 3 times less than found using Sniffles. Of these, 164 (10.34%) and 203 (12.17%) could not be identified in the son.

We next tested how many calls are in the son that are not within the parents to investigate potential false positive calls (**Supplementary Table 6**). Using the same parameter settings, Sniffles had the lowest number of such calls in the son for deletions (515 vs. 677), inversions (66 vs. 75) and translocations (90 vs. 1,550) compared to SURVIVOR. Only for tandem duplications SURVIVOR has 75 events that are unique to the son versus 115 that Sniffles calls. On investigation, most of the Sniffles calls found only in the son were due to the lower coverage of the parents. When taking this into account we found 1,065 SVs unique in the son but not in the parents, which equals an 5.57% inconsistency.

Overall, Sniffles and NGMLR had the highest recall rate as well as the lowest Mendelian discordance rate. In contrast, the short read approaches showed an unreasonably high number (1,550) of inconsistent translocations in the son.

Supplemental References

1 Sedlazeck, F. J., Rescheneder, P. & von Haeseler, A. NextGenMap: fast and accurate read mapping in highly polymorphic genomes. *Bioinformatics* **29**, 2790-2791, doi:10.1093/bioinformatics/btt468 (2013).

2 Kurtz, S. *et al.* Versatile and open software for comparing large genomes. *Genome Biol* **5**, R12, doi:10.1186/gb-2004-5-2-r12 (2004).

3 Gusfield, D. *Algorithms on strings, trees, and sequences: computer science and computational biology*. (Cambridge University Press, 1997).

4 Chaisson, M. J. & Tesler, G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics* **13**, 238, doi:10.1186/1471-2105-13-238 (2012).

5 Li, H. *et al.* The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**, 2078-2079, doi:10.1093/bioinformatics/btp352 (2009).

6 Hartigan, J. A. & Hartigan, P. M. The Dip Test of Unimodality. *Ann Stat* **13**, 70-84, doi:DOI 10.1214/aos/1176346577 (1985).

7 Balanda, K. P. & Macgillivray, H. L. Kurtosis - a Critical-Review. *Am Stat* **42**, 111-119, doi:Doi 10.2307/2684482 (1988).

8 Jeffares, D. C. *et al.* Transient structural variations have strong effects on quantitative traits and reproductive isolation in fission yeast. *Nat Commun* **8**, 14061, doi:10.1038/ncomms14061 (2017).

9 Holtgrewe, M. Mason-A Read Simulator for Second Generation Sequencing Data. (Institut für Mathematik und Informatik, Freie Universität Berlin, 2010).

10 Jain, M. *et al.* Nanopore sequencing and assembly of a human genome with ultra-long reads. *bioRxiv*, doi:doi: 10.1101/128835 (2017).

11 Zook, J. M. *et al.* Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nat Biotechnol* **32**, 246-251, doi:10.1038/nbt.2835 (2014).

12 Tarasov, A., Vilella, A. J., Cuppen, E., Nijman, I. J. & Prins, P. Sambamba: fast processing of NGS alignment formats. *Bioinformatics* **31**, 2032-2034, doi:10.1093/bioinformatics/btv098 (2015).

13 Layer, R. M., Chiang, C., Quinlan, A. R. & Hall, I. M. LUMPY: a probabilistic framework for structural variant discovery. *Genome Biol* **15**, R84, doi:10.1186/gb-2014-15-6-r84 (2014).

14 Quinlan, A. R. & Hall, I. M. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* **26**, 841-842, doi:10.1093/bioinformatics/btq033 (2010).