Subject Section

Ira: the Long Read Aligner for Sequences and Contigs

Jingwen Ren¹, Mark JP Chaisson¹,*,

¹Quantitative Biology, Biological Science, University of Southern California, Los Angeles, 90007, the United States.

*To whom correspondence should be addressed.

Abstract

Motivation: It is computationally challenging to detect variation by aligning long reads from single-molecule sequencing (SMS) instruments, or megabase-scale contigs from SMS assemblies. One approach to efficiently align long sequences is sparse dynamic programming (SDP), where exact matches are found between the sequence and the genome, and optimal chains of matches are found representing a rough alignment. Sequence variation is more accurately modeled when alignments are scored with a gap penalty that is a convex function of the gap length. Because previous implementations of SDP used a linear-cost gap function that does not accurately model variation, and implementations of alignment that have a convex gap penalty are either inefficient or use heuristics, we developed a method, Ira, that uses SDP with a convex-cost gap penalty. We use Ira to align long-read sequences from PacBio and Oxford Nanopore (ONT) instruments as well as de novo assembly contigs.

Results: Across all data types, the runtime of Ira is between 52-168% of the state of the art aligner minimap2 when generating SAM alignment, and 9-15% of an alternative method, ngmlr. This alignment approach may be used to provide additional evidence of SV calls in PacBio datasets, and an increase in sensitivity and specificity on ONT data with current SV detection algorithms. The number of calls discovered using pbsv with Ira alignments are within 98.3-98.6% of calls made from minimap2 alignments on the same data, and give a nominal 0.2-0.4% increase in F1 score by Truvari analysis. On ONT data with SV called using Sniffles, the number of calls made from Ira alignments is 3% greater than minimap2-based calls, and 30% greater than ngmlr based calls, with a 4.6-5.5% increase in Truvari F1 score. When applied to calling variation from de novo assembly contigs, there is a 5.8% increase in SV calls compared to minimap2+paftools, with a 4.3% increase in Truvari F1 score.

Availability and implementation: Available in bioconda: https://anaconda.org/bioconda/Ira and github: https://github.com/ChaissonLab/LRA

Contact: mchaisso@usc.edu, jingwenn@usc.edu

1 Introduction

Studies of genetic variation often begin by aligning sequences from a sample back to a reference genome, and inferring variation as differences in the alignment. Long read, single molecule sequencing (LRS) is becoming established as a routine approach for sequencing genomes. The two technologies that produce LRS technologies, Pacific Biosciences (PacBio) and Oxford Nanopore (ONT) generate reads over 50kb at error rate 15% or less. Aligning these sequences is a computationally challenging task for which several methods are available including minimap2, ngmlr, and BLASR. They are demonstrated to be quite fast and accurate, but have limitations, particularly when there are large sequence differences between the read and the reference. This problem is amplified in complex, repetitive regions such as variable-number tandem repeats, that only make up 6% of the human genome, but account for nearly 70% of observed structural variation: insertions and deletions at least 50 bases (SV), and in larger SV. A common approach for mapping LRS reads is seeding and chaining, where an approximate alignment is formed based on a subset (chain) of exact matches between the sequence and the genome. The exact matches may be found using various data structures including variable-length matches using a BWT-FM index or suffix arrays, and minimizer based indexing. The chaining algorithm used by BLASR uses a linear cost gap function for sparse-dynamic programming (SDP) and minimizer based indexing. While the chaining algorithm is efficient, it has long been known that linear-cost gap functions do not accurately reflect biological variation, and has been shown to decrease sensitivity for detecting SV from LRS alignments.
Both the minimap2 and ngmrl aligners use gap penalties that are a convex function of gap length, and are demonstrated to be quite effective for mapping LRS reads across SV with alignment gaps that reflect biological variation. In minimap2, a heuristic algorithm is used for chaining, while ngmrl adopts a Smith-Waterman-like dynamic programming algorithm. An exact solution to sparse dynamic programming with a convex gap (CG-SDP) function exists [Epipstein et al., 1992], and is slightly less efficient than linear-cost SDP. However, as presented, the algorithm requires asynchronous processing and has never been implemented for sequence alignment in computational biology. Furthermore, the algorithm does not take into account genome arrangements such as inversions.

An additional challenge for alignment enabled by LRS is discovering variation from de novo assembly. Efficient LRS assembly algorithms enable assembly of human genomes with contiguity on part of the initial release of the human genome [Shahin et al., 2020; Kolmogorov, 2019]. When combined with single-nucleotide polymorphisms phased using trio or long-range sequencing information such as Hi-C and strand-seq, LRS may be assembled into phased diploid genomes [Koren, 2018; Githiky et al., 2019]. The breakpoints of sequence variation may be detected directly from assembly alignments, as opposed to read based variant detection which requires the consensus sequence of noisy breakpoint sequences that may have reference bias. This benefit comes at the expense of requiring accurate alignment of sequences tens of millions of bases long that may harbor rearrangements and duplications. The MashMap method can rapidly detect sequence homology in assemblies with low memory requirements [Kim et al., 2017], however the pairwise alignments are not detailed enough to detect variants less than a few hundred bases, which accounts for the majority of variation between human genomes. It is also possible to align long contigs using minimap2, which can produce output that captures all scales of variation from SNPs to large SV, however the resulting alignments tend to be split at structural variation and can be unmapped in repetitive regions.

To explore the application of the exact solution of seed chaining sparse dynamic programming with a convex gap function to read and assembly alignment, we developed an alignment method lra. We have simplified the implementation of the CG-SDP algorithm [Epipstein et al., 1992b], and extended to allow for inversions and translocations.

2 Results

We compared alignment metrics and variant discovery on data from HG002 including three sequencing datasets: PacBio consensus reads (HiFi), PacBio single-pass reads (CLR), and ONT reads, as well as on contigs from a haplotype-resolved de novo assembly [Koren, 2019]. All data were mapped to build GRCh37 to use curated variants for accuracy analysis [Zook et al., 2020]. The PacBio HiFi data are characterized by high-accuracy reads (>99%) with an average length of 198k, compared to the CLR reads that have an accuracy around 80% and an average read length of 21kb. The ONT data: 88% accuracy and 1.2kb average read length. All read datasets are above 40x coverage of a human genome, which are sufficient for human sequencing studies.

We compared read alignments to minimap2 and ngmrl, two commonly used alignment methods for LRS data. When comparing SAM formatted alignments with minimap2, the lra and minimap2 runtimes are comparable; lra is faster for aligning HiFi reads, while minimap2 is faster for CLR and ONT data, although all are within a factor of two (Table 1). The number of bases aligned by lra and minimap2 are within 0.03-7%.

While the performance metrics for runtime and number of aligned bases are worse for ngmrl, it is important to consider this method for downstream analysis because the potential for sensitivity for SV detection may be more important than runtime. A mapping quality is assigned based on the number of mismatches of secondary alignments close to the same length of the primary alignment. Simulated data was used to measure how mapping accuracy tracks with mapping quality. We compared the mapping accuracy of lra and minimap2 on simulated HiFi, CLR and ONT data of different read lengths: 5kb, 10kb, 20kb and 50kb. Simulated reads were mapped to genome GRCh38. A read is considered as correctly mapping if its longest alignment has ≥ 80% overlap with its true interval. The mapping accuracy is measured by NC/TN, where NC is number of correctly mapped reads and TN is total number of mapped reads (Figure 1). Reads were simulated using a method, alchemy2 included with the lra source that builds an error model from a bam file of aligned reads. While alignment accuracy was up to 0.5% lower than that of minimap2 in repetitive regions (indicated by lower mapping quality), downstream analysis was not affected.

A common application of LRS in human genetics is discovering structural variation [Chaisson et al., 2015; Sedlacek et al., 2018]. We evaluated SV detection from mapped reads across combinations of technologies, aligners, and SV calling algorithms. Reads were aligned with lra, pbmm2/minimap2, and ngmrl, and variants were detected with pbsv and Sniffles [Wenger et al., 2019; Sedlacek et al., 2018]. The pbmm2 method encapsulates minimap2 with technology and SV discovery specific parameters. The pbsv method did not run using ONT reads, and so pbsv analysis of ONT data is omitted. The majority of SV are under 500 bases [Chaisson et al., 2019] and are spanned by LRS alignments. Consequently, variant calls with similar representations of gaps should have converging callsets, which may be confirmed by comparing SV calls from different combinations of algorithms. The most consistent variant calling was found using pbsv on both types of PacBio using lra and pbmm2 alignments. These callsets ranged from 20,574-21,754 SVs, with a difference of 1.4-1.7% between alignment algorithms and 1.5-4.6% between data types (Table 3). Compared to the pbsv calls, there was greater variation between callsets generated using Sniffles across both data types and alignment algorithms, with a range of 15.062-24.067 SVs per callset. There was less variation between the size of callsets of different data types and the same alignment algorithm (average difference of 2.767 call differences per callset), compared to different alignment algorithms applied to the same data (an average difference of 6.412 per callset).

A greater call count may reflect more sensitive detection of SV, or simply fragmentation of variants. To assess the accuracy of variant callsets, we compared callsets across the GIAB Tier 1 calls [Zook et al., 2020] using Truvari (https://github.com/spiralgenetics/truvari). The lra and pbmm2 based callsets outperform ngmrl based calls in precision and recall on all data types (Table 4). While the F1 scores are effectively equivalent between lra and pbmm2 based variant calls on HiFi (0.972 vs 0.968) and CLR (0.969 vs 0.967) using pbsv, there is a substantial improvement on calls on ONT data made by Sniffles (0.950 vs 0.910). This indicates that the greater call count on ONT data includes at least partially attributed to increasing recall, particularly in larger (>500 base) insertions, without affecting precision on the high-quality regions that were ascertained.

While the F1 scores are nearly equivalent on PacBio data, the combination of algorithms may contribute to a more complete evaluation of an LRS genome. The lra and pbmm2 contribute 67 and 59 unique calls, respectively, that were annotated as true positives in the HiFi/pbsv call sets, and 83/112 unique calls from the CLR/pbsv call sets. The average length of the uniquely called true positive SV is between 1.3-4.4kb, highlighting the challenge of detecting longer SV from read based alignments. To further assess the potential for callsets to be more comprehensive, benchmark variants annotated as false-negatives were compared to variants discovered directly from read alignments. Each call where at least 20% of the reads overlapping the call had an SV of the same type with a length between 0.6-1.4× the length of the call were annotated as a support false-negative (sFN). Between 14.5%-68.4% of annotated false negatives from pbsv
variant calls, and 35.0%-95.4% of false-negative calls in Sniffles callsets were considered s/FN. Furthermore, there were 44% more supported false-negative calls in the PacBio HiFi kraft/pvs callsets compared to the pBmm2/pvs callsets, and on average 79% more s/FN calls from lra and minmap2 alignments in Sniffles callsets, an expected outcome of tuning a variant discovery algorithm to input generated by a particular alignment algorithm.

A comparison of inversion calls between algorithms indicates this class of variation is a remaining challenge for SV discovery. Inversion calls for ngmlr alignments were directly obtained from the vcf file generated by Sniffles. For lra and minmap2, inversions calls were first obtained from the PAF output, since a tag indicating inversion is given in the PAF and NA if it was not clear, commonly in pericentric regions. False calls for lra could not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplotypes did not show variation or an inverted duplication structure, false-positive if both haplo...
by the trio-camu assembler [Wenger et al. 2019]. The haplotype assemblies resolve 2.96Gb and 3.06Gb, with N50 values 16.1 and 18.0 Mb. Both alignment methods have similar runtimes to align de novo assembly contigs (< 7 min, 16 threads), although lra requires roughly twice the memory. It was not possible to use ngm2 to align contigs. Variants were detected from the minimap2 alignments using pathools.js. To detect variants using lra, alignments were first filtered by removing short alignments spanning less than 200kbp, and calling SV directly from alignments (e.g. CIGAR strings), selecting calls only from the longest contig when multiple contigs overlap. Calls from different haplotypes that overlap at least 30% were determined as homologous calls and the rest were classified as heterozygous calls. Additionally, calls in centromere regions were removed for both minimap2/pathools.js and lra callsets. This generated 25,708 calls by lra call sets, and 24,149 by minimap2/paftools.js. Truvari analysis of the calls gives an F1 score of 0.9382 by lra calls, and 0.8989 for pbmm2 calls (Table 3).

The high-confidence callset used in Truvari analysis contains fewer than 10,000 SVs, less than half what are expected to be found in a human genome. To gauge the specificity of assembly-based calls outside this set, SVs contained in read alignments were compared to the SV discovered from assemblies. An SV detected from an individual read supported an assembly-based SV of the call was the same type (e.g. insertion or deletion), was within 1kbp, and the ratio of SV lengths was between 0.5 and 2. When using PacBio CLR lra-aligned reads, the majority (99.7%) of SVs, 98.7% of lra deletion SV, and 99.7% of minimap2 deletion SV were supported by reads. This is greater than the precision measured by Truvari on assembly-based calls. 0.9332 for lra, and 0.9295 for minimap2, indicating an under estimate of the callset precision. When inspecting the SV calls that are not supported by read alignments, many are due to differential placement of gaps causing disagreement of SV length between the read and assembly-based calls.

3 Methods

The alignment of reads and contigs to a reference are generally defined by the maximally scoring local alignment of a query q to a set of target sequences collectedly referred to as a target t with a match bonus and penalties for mismatch, gaps, and inversions/translocation. Ira uses a heuristic to find an approximate local alignment employing the commonly used seeding, chaining, and refinement approach that has been applied to all scales of alignment from short-read, long-read, and whole-genome alignment [Kent et al. 2002; Sandiackez et al. 2013; Chaisson and Tesler 2012]. Each alignment proceeds in four broad steps: seed sequence matching, clustering, chaining, and finally alignment refinement.

Many recent advances have been made in sequence mapping using a subsampled index on a reference using minimizers or locally sensitive hashing [Roberts et al. 2004; Linn et al. 2017]. A minimizer index is parameterized by a k-mer size k and window size w, and indexes the position of the lexicographically least canonical k-mer in every sequence of length w across the genome. We develop a variant on the approach of minimizers that uses adaptive thresholds to limit the total number of positions sampled in unique regions of a genome, and increase the sampling of positions near paralog-specific variants that distinguish repetitive regions.

Optimal sets of matches between the query and reference are selected in two phases using CG-SDP. Given a set of fragments \( F \), each fragment is defined by a start point and an end point on a Cartesian plane and a weight: \( \omega(q_j, C, j) \). The basic CG-SDP formulation is to define a chain of fragments \( C \subseteq \{1, \ldots, n\} \) that maximizes

\[
\sum_{j=1}^{n-1} \omega(q_j, C, j) + \sum_{j=1}^{n} \text{gap}(\alpha C_{j-1}, \alpha C_{j+1})
\]

such that \(\alpha C_{j-1}\) is above and to the right of \(\alpha C_j\), and \(\text{gap}\) is a convex function. An algorithm was presented for chaining using a convex gap-cost model [Eppstein et al. 1998], however there are no alignment methods that implement this approach. This method has been implemented in lra and extended to allow for inversions in the optimal chain. Additionally, the original description of the algorithm requires asynchronous processing, which we have updated to use standard serial computation. The details of determining minimizers and the chaining algorithm are given below.

3.1 Building a Minimizer Index and Sequence Matching

We add three additional parameters to generate a minimizer index: \( F^M \), \( N^M \) and \( W^G \) that limit the density of minimizers that are selected. An initial set of minimizers is determined in the standard approach, with minimizer k-mer parameter k and window w [Roberts et al. 2004]. Next, minimizers of multiplicity larger than \(F^M\) are removed. Then the reference is partitioned into intervals of length \(W^G\), and the remaining minimizers starting in each interval are selected in order of their multiplicity in the genome until the first \(N^M\) minimizers are obtained. Ira uses slightly different parameters settings for aligning reads from different sequencing technologies to genome. When indexing the genome for aligning HiFi reads, 867M minimizers from total 1015M were left after filtering by frequency threshold \(F^M\). In total, 117M

Table 3. Structural variant call sets. Summary of SV call sets from different aligners (lra/pbmm2/ngmlr) + SV call sets (pbsv/Sniffles).

| Minimizer | Average call size | Total size | SV type | Number of calls | Average call size | Total size |
|-----------|------------------|------------|---------|-----------------|------------------|------------|
| lra       | 509              | 6.27M      | INS     | 12328          | 524.23           | 6.60M      |
|           | 581              | 4.90M      | DEL     | 8426           | 518.39           | 4.42M      |
|           | 489              | 6.19M      | INS     | 12655          | 523.53           | 6.72M      |
|           | 583              | 5.12M      | DEL     | 8784           | 603.51           | 5.39M      |
|           | 499              | 6.60M      | INS     | 113247         | 397.337          | 4.44M      |
|           | 678              | 7.33M      | DEL     | 10820          | 685.866          | 6.35M      |
|           | 604              | 7.23M      | INS     | 11995          | 380.117          | 4.09M      |
|           | 802              | 7.05M      | DEL     | 8797           | 458.806          | 3.6M       |
|           | 714              | 8.78M      | INS     | 12299          | 415.157          | 4.97M      |
|           | 688              | 6.85M      | DEL     | 10005          | 644.433          | 6.37M      |

| Sniffles  | Average call size | Total size | SV type | Number of calls | Average call size | Total size |
|-----------|------------------|------------|---------|-----------------|------------------|------------|
| lra       | 509              | 6.27M      | INS     | 12328          | 524.23           | 6.60M      |
|           | 581              | 4.90M      | DEL     | 8426           | 518.39           | 4.42M      |
|           | 489              | 6.19M      | INS     | 12655          | 523.53           | 6.72M      |
|           | 583              | 5.12M      | DEL     | 8784           | 603.51           | 5.39M      |
|           | 499              | 6.60M      | INS     | 113247         | 397.337          | 4.44M      |
|           | 678              | 7.33M      | DEL     | 10820          | 685.866          | 6.35M      |
|           | 604              | 7.23M      | INS     | 11995          | 380.117          | 4.09M      |
|           | 802              | 7.05M      | DEL     | 8797           | 458.806          | 3.6M       |
|           | 714              | 8.78M      | INS     | 12299          | 415.157          | 4.97M      |
|           | 688              | 6.85M      | DEL     | 10005          | 644.433          | 6.37M      |
Table 4. Truvari classification of variant calls. Truvari comparisons between lra and pbmm2/minimap2 using the Genome in a Bottle benchmark SV set. Optimal results in each category are shown in bold. TP-base means true positive calls in the benchmark SV curation set, while TP-call means true positive calls in the SV set from each aligner. False positive means the number of non-matching calls from the SV set from each aligner. False negative means the number of non-matching calls from the SV curation set. False negative variants that show underlying read support are given by sFN.

|                  | pbmm2 | minimap2 | lra | minimap2 | lra | pbmm2 | minimap2 | lra | pbmm2 | minimap2 | lra | pbmm2 | minimap2 | lra | pbmm2 | minimap2 |
|------------------|-------|----------|-----|----------|-----|--------|----------|-----|--------|----------|-----|--------|----------|-----|--------|----------|
| TP base          | 9425  | 9408     | 633 | 9197     | 9414| 9076   | 9012     | 8700| 8015   | 7852     | 8082| 6533   | 9123     | 8887| 8471   |
| TP call          | 9459  | 9449     | 8500| 9484     | 9507| 9208   | 9034     | 8713| 8028   | 7942     | 8147| 6567   | 9151     | 8902| 8488   |
| FP               | 331   | 390      | 396 | 358      | 418 | 428    | 281      | 357 | 800    | 2705     | 1492| 1298   | 445      | 940 | 815    |
| FN               | 216   | 233      | 1208| 244      | 227 | 565    | 629      | 941 | 1626   | 1789     | 1559| 3108   | 518      | 754 | 1170   |
| sFN              | 124   | 93       | 176 | 167      | 151 | 268    | 516      | 598 | 573    | 1706     | 1423| 2387   | 439      | 464 | 410    |
| precision        | 0.967 | 0.96     | 0.955| 0.964    | 0.958| 0.956  | 0.97     | 0.961| 0.909  | 0.746    | 0.845| 0.835  | 0.954    | 0.904| 0.912  |
| recall           | 0.978 | 0.976    | 0.875| 0.975    | 0.976| 0.941  | 0.935    | 0.902| 0.831  | 0.814    | 0.838| 0.678  | 0.946    | 0.922| 0.879  |
| F1 score         | 0.972 | 0.968    | 0.913| 0.969    | 0.967| 0.948  | 0.952    | 0.931| 0.869  | 0.779    | 0.842| 0.748  | 0.95     | 0.91 | 0.895  |

Fig. 2. The precision and recall of SV call sets from HG002 HiFi, CLR, ONT, and assembly-based callsets, measured by Truvari using Genome in a Bottle benchmark SV callsets.

minimizers were selected after the final filtering based on $N^M$ and $W^C$. All minimizers from a query sequence are matched against the filtered set of minimizers from the reference. The result of the sequence matching is a set of anchors $A = \{\beta_1, \ldots, \beta_n\}$, where $\beta$ is a tuple $(x_i, y_j, k)$, where $q[x_i, x_{i+1}, \ldots, x_{i+k-1}]$ of the query matches $t[y_j, y_{j+1}, \ldots, y_{j+k-1}]$ of the target.

An additional local minimizer index is built on the query and target, parameterized by standard minimizer parameters $k^i$ and $w^i$, and a tiling length $W^i$. A separate minimizer index is stored for sequences of length $W^i$ tiling across both the query and the target, with $k^i < k$ and $w^i < w$.

The local minimizer index is used to speed the alignment of the query and target once the rough alignment is detected.

3.2 Clustering

Although CG-SDP can be applied to all anchors $A$, for efficiency a greedy approach is used to cluster anchors that would likely be together on an optimal chain, and CG-SDP is applied to clusters. Two clustering steps are performed, rough and fine clustering, where rough clusters contain matches in the same target region, and fine clusters divide rough clusters.
are greedily assigned with first interval starting at the first index in a diagonal within sorting on reverse diagonal and intervals on the query and target that are aligned (Figure 3a). Denoting the Rough clustering partitions anchors into clusters representing approximate 6
interval. Intervals with few elements (defined by a defined by ordering anchors by forward diagonal and then x coordinate. This will be used to detect alignments on the reverse strand, but because the operations are the same as on the forward strand, only subsequent steps using the forward sorted order are given. The set of rough clusters is defined by partitioning O into non-overlapping intervals such that every anchor indexed in an interval has a diagonal within D\(D^R\) of the first anchor indexed in the interval. Intervals are greedily assigned with first interval starting at the first index in \(O\), and subsequent intervals starting on the first anchor beyond \(D^R\) of the previous interval. Intervals with few elements (defined by a minClusterSize parameter) are discarded, and the rough clusters \(R = \{R_1, \ldots, R_N, R\}\) are defined from the set of anchors included in each interval.

### 3.3.1 Rough Clustering

Rough clustering partitions anchors into clusters representing approximate intervals on the query and target that are aligned (Figure 3). Denoting the forward diagonal of each anchor \(i, y = y_i - x_i\), and the reverse diagonal \(r = x_i + y_i\), a sorted anchor order \(O = [o_1, \ldots, o_N]\) is defined by ordering anchors by forward diagonal and then x coordinate. A reverse sorted order \(O^{rev} = [o_{N+1}^{rev}, \ldots, o_N^{rev}]\) is similarly defined sorting on reverse diagonal and x coordinate. This will be used to detect alignments on the reverse strand, but because the operations are the same as on the forward strand, only subsequent steps using the forward sorted order are given. The set of rough clusters is defined by partitioning O into non-overlapping intervals such that every anchor indexed in an interval has a diagonal within \(D^R\) of the first anchor indexed in the interval. Intervals are greedily assigned with first interval starting at the first index in \(O\), and subsequent intervals starting on the first anchor beyond \(D^R\) of the previous interval. Intervals with few elements (defined by a minClusterSize parameter) are discarded, and the rough clusters \(R = \{R_1, \ldots, R_N, R\}\) are defined from the set of anchors included in each interval.

### 3.2.2 Fine clustering

In this step, each rough cluster is processed independently by dividing into non-overlapping fine clusters, where each fine cluster consists of anchors on a close diagonal \(D^F\), where \(D^F < D^R\), with endpoints that do not overlap. The first step of CG-SDP will be applied to chain the fine clusters and find an approximate alignment between two sequences in each fine cluster. The coordinates of each split cluster are set according to the first/last anchors appearing after/before the boundary point. An optimal chain of fragments \(F^{OPT}\) is then found using CG-SDP on \(F\).

### 3.3 Cluster Splitting and Chaining

Each fine cluster \(C_j \in C\) defines a super-fragment \(F\), that unlike the minimizer fragments which have starting and endpoints along the same diagonal, the endpoints of \(C_j\) may not be along a diagonal. The set of all such fragments is \(F\), and an optimal chain of fragments will be defined using CG-SDP. However, considering the rectangles defined by the boundaries of each fragment, the CG-SDP algorithm only selects fragments with non-overlapping rectangles in the optimal chain. Due to the repetitive nature of genomes, this may result in erroneously skipped fragments (Figure 3). To account for this, fine clusters are split at overlapping boundaries (Figure 3b). The start and end coordinates of all fine clusters are stored in a set that is queried to find boundary points appearing in the range of each fine cluster. The coordinates of each split cluster are set according to the first/last anchors appearing after/before the boundary point. An optimal chain of fragments \(F^{OPT}\) is then found using CG-SDP on \(F\).

### 3.4 Cluster Refinement

The optimal chain of super-fragments obtained from section 3.3 contains the anchors from which the optimal alignment will be defined. A pairwise alignment may be created using dynamic programming on the substrings between the minimizer matches, however for high-error rate reads matches are sparse and the length of substrings may be too large to efficiently compute, or have too large of a diagonal gap to use banded alignment. A second anchoring step using the local minimizer index is used to detect shorter and more dense anchors. The local minimizer index contains \(\left\{ \frac{x_{t+1}}{y_{t+1}} \right\}\) and \(\left\{ \frac{x_{t}}{y_{t}} \right\}\) separate minimizer indexes for the target and query sequences that index each tiling substring of length \(W\), accounting for the relative positions of the substrings in each sequence. Every pair of substrings from \(q\) and \(t\) that have an anchor in \(C^{OPT}\) are compared using their local minimizer indexes to generate a resulting set of anchors \(A^{local}\). To reduce runtime of CG-SDP on \(A^{local}\), anchors that are adjacent in Cartesian sorted order and on the same diagonal are merged. The length of any merged anchor is the difference from the last endpoint to the first starting point. The resulting merged fragments are chained using CG-SDP, as described in section 3.6, and the anchors on this chain are denoted as \(A^{local}\).

### 3.5 Alignment Refinement

Banded alignment is used to create a pairwise alignment between anchors in \(A^{local}\). We assume that large gaps between anchors may be modeled using an affine alignment that allows a single large penalty-free gap. Given two sequences \(q^{local}\) and \(t^{local}\), a match matrix \(M\), a gap penalty \(\delta\), and an alignment band \(B\), assume that \(|\{\text{local}^r\}| < |\{\text{local}^r\}|\). A lower-score matrix \(S^{lower}\) for a typical banded alignment band \(B\) and gap penalty \(\delta\), e.g.

\[
S_{lower} = \max\{S_{\text{local}^r} - \delta, S_{\text{local}^r} - \delta\} \quad \text{if} \quad |l - j| \leq B, \quad \infty \quad \text{otherwise}
\]

Next, an upper-diagonal matrix \(S_{upper}\) is calculated that allows for a single transition from the lower matrix with banded alignment.

Denoting the length of the \(\{\text{local}^r\}\) and \(\{\text{local}^t\}\) as \(P\) and \(T\),

\[
S_{upper} = \max\left\{\frac{S_{upper}^{local} - \delta}{\min\{P, T\}}, \frac{S_{upper}^{local} - \delta}{\min\{P, T\}}\right\} \quad \text{if} \quad |l - j| > B, \quad \infty \quad \text{otherwise}
\]

(2)
Table 6. Read based support of assembly calls. The two haplotype assemblies are considered separately to avoid complications of merging into a diploid callset. Calls are produced by lra and minimap2 (mm2), with only deletion and insertion SV classes considered. The total calls are the original calls produced by each method, and filtered are calls excluding centromeres and 50kb of flanking sequence. The supported SV have at least four reads supporting the call from either lra or minimap2 alignments.

| Assembly       | SV type | total | filtered | supported (fraction) |
|----------------|---------|-------|----------|----------------------|
| Haplotype 1, lra | DEL     | 6151  | 5988     | 5900 (0.985)         |
| Haplotype 1, mm2 | DEL     | 5808  | 5804     | 5782 (0.996)         |
| Haplotype 2, lra | DEL     | 6074  | 6029     | 5944 (0.986)         |
| Haplotype 2, mm2 | DEL     | 5680  | 5671     | 5645 (0.995)         |
| Haplotype 1, lra | INS     | 9760  | 9589     | 9522 (0.993)         |
| Haplotype 1, mm2 | INS     | 9640  | 9634     | 9624 (0.999)         |
| Haplotype 2, lra | INS     | 9546  | 9496     | 9435 (0.994)         |
| Haplotype 2, mm2 | INS     | 9394  | 9383     | 9368 (0.998)         |

Fig. 3. Example of clustering anchors prior to CG-SDP. a, Two rough clusters (blue and orange), which are far from each other on the reference. b, The initial four fine clusters defined from the contiguous stretches of unique anchors. c, Fine cluster-1 and fine cluster-2 are merged because their diagonal difference is smaller than \(D\) and projected distances between their endpoints is smaller than \(G\). d, Fine cluster-3 and fine cluster-4 are not merged due to the large diagonal difference. e, Non-unique anchors in the trapezoid between fine cluster-1 and fine cluster-2 are added to the merged fine cluster-1, along with non-unique anchors in the trapezoid defined by the start of rough cluster-2 and the start of fine cluster-3. f, Fine clusters are obtained after rough clustering and fine clustering. g-h, Splitting of overlapping fine-clusters: f, overlap of clusters. g, Boundaries of split clusters defined by a start (red dot) and an end (blue dot). h, the optimal chain of split super-fragments.

3.6 Problem Statement of the Chaining

We define a set of fragments \(\Phi = \{\alpha_1, \ldots, \alpha_n\}\). Each fragment \(\alpha_i\) is associated with a lower left start point \((x_i^l, y_i^l)\) and upper right end point \((x_i^r, y_i^r)\), and a score \(l_i\). For minimizer fragments, the upper right endpoints are a fixed distance from the lower points, e.g. \((x_i^l + k, y_i^l + k)\). For super fragments defined by fine clusters, the endpoints and starting points may not be on the same diagonal. The starting point of a fragment \(\alpha\) is denoted \(s(\alpha)\), and the end \(e(\alpha)\). A point \((x, y)\) is above \((x', y')\) if \(x > x'\) and \(y > y'\) (conversely \((x', y')\) is below \((x, y)\)). The chaining score is defined by Equation (1) where \(gap(\alpha C_j, \alpha C_j + 1)\) is a convex function of \(\left|\left(\alpha C_j + 1\right) - (\alpha C_j)\right|\); the difference between the forward diagonals of the endpoint of \(\alpha C_j\) and the starting point of \(\alpha C_j + 1\). The naive way to solve this problem takes \(O(n^2)\) in time. By applying CG-SDP \(\text{Eppstein et al.} [1992b]\), the runtime is \(O(n \log(n))^2\). Below, the solution provided by Eppstein, Galil, and Giancarlo is described with increase clarity from the original description, with a modification that enables calculation with synchronous computation. Finally, the method is extended to allow for inversions, similar to \(\text{Brudno et al.} [2003]\).
3.6.1 SDP Algorithm with Convex Gap Cost Function - Defining Subproblems

For simplicity, assume both sequences are the same length $l$ and that all points are in $[0, l]$ (e.g. shifted by the minimum $x$ and $y$ coordinate), and are on a $l \times l$ grid. To speed the chaining algorithm, the search for the fragment that precedes another on an optimal chain is divided into multiple overlapping subproblems that may be solved independently and more efficiently than the naive scan, and the globally optimal score for each point is selected from each of the subproblems that overlap it. Each subproblem divides a block of $t 	imes s$ columns or rows of the search space into an $A$-part and $B$-part covering $\left(\frac{s}{2}, \frac{s}{2}\right)$ and $\left(\frac{t}{2}, \frac{t}{2}\right)$ columns/rows respectively, where $A$-part contains all the endpoints and $B$-part contains all the startpoints in the corresponding columns/rows. When the size of a subproblem is only one column/row, the $A$-part of the corresponding subproblem is empty. Each subproblem is described by the label $d \in \{\text{column}, \text{row}\}$, the starting and ending rows and columns of the subproblem $A$, $A_c$, $B$, $B_c$, and a data structure of values used to compute the optimal chain, DAT.A. The full set of subproblems $\text{Sub}(d, s, e)$ are generated recursively as:

\[
\begin{align*}
\{d, 0, 0, s, e, \text{DAT.A}\} & \quad \text{if } s = e, \\
\{d, s, \left[\frac{s}{2}\right], \left[\frac{s}{2}\right] + 1, e, \text{DAT.A}\} \cap \\
\text{Sub}(d, s, \left[\frac{s}{2}\right]) \cap \\
\text{Sub}(d, s, \left[\frac{s}{2}\right] + 1, e) & \quad \text{otherwise}
\end{align*}
\]

(3)

The more detailed pseudocode for this partitioning is given in the supplementary material (Supplementary Algorithm S1). Figure 8 visualizes the column and row subproblem division of a simple case with six fragments.

Each subproblem is processed by finding the optimal endpoint from the $A$-part that precedes each starting point in corresponding $B$-part. For a starting point $p_i$ that is assigned to the $B$-parts of a set of column-based subproblems, the union of the $A$-parts of those subproblems form the entire plane to the left of the point. Similarly for a starting point assigned to the $B$-part of a set of row subproblems, the union of the $A$-parts of those subproblems form the plane below the point. After solving for the optimal preceding endpoint in all of the subproblems in which $p_i$ is contained, the one with maximum score among these endpoints is the global optimal, which represents the optimal chain from all endpoints below and to the left of $p_i$.

Once the subproblems are defined, list elements of DAT.A are allocated and initialized. The following elements are associated with the $A$-part of a column/row subproblem:

- $D_{1i}$: The forward diagonals in increasing/decreasing order overlapped by endpoints in the $A$-part.
- $D_{PV}$: The optimal chaining score for diagonals overlapped by endpoints in $A$-part. $D_{PV}[s]$ holds the optimal value of forward diagonal $D_{1i}[s]$.
- $D_{PV}$: Index of point with optimal score along diagonal. $D_{PV}[s]$ references the point with the best score of $D_{PV}[s]$.

The following elements are associated with the $B$-part of a column/row subproblem:

- $E_{1i}$: The forward diagonals in increasing/decreasing order overlapped by starting points in the $B$-part.
- $E_{PV}$: Similar to $D_{PV}$, the locally best chaining value.
- $E_{PV}$: Index of the forward diagonal in $D_{1i}$ that leads to the best chaining value $E_{PV}[s]$ for the forward diagonal $E_{1i}[s]$.
- $E_{B}$: A block structure used in the calculation of $E_{PV}$.

- $E_{PV}$: Index of the most recent value in $D_{PV}$ used to compute values in $E_{PV}$.

Additionally, for each point, two lists hold references to each subproblem a point is contained in, one structure $SA$ for points in $A$-parts of subproblems, and $SB$ for points in $B$-parts of subproblems. To make the description of the method more concise, the operation $\varphi(D_{1i}, f)$ and $\psi(E_{1i}, f)$ are defined to give the index of diagonal $f$ in $D_{1i}$ and $E_{1i}$, respectively.

3.6.2 SDP Algorithm with Convex Gap Cost Function - Conquering Subproblems

The original description of CG-SDP, uses asynchronous processing during solving subproblems (Eppelein et al., 1992b). Here we give an alternative description of the algorithm, and provide an approach to solve subproblems in a way that allows synchronous processing of points in Cartesian order for a more simple implementation. The framework for solving a subproblem is first described, and then an order of processing points is given to solve for optimal fragment chaining.

For any subproblem, the optimal chains between endpoints in the $A$-part to starting points in the $B$-part are found using the $D_{1i}$, $D_{PV}$, $E_{1i}$, $E_{PV}$ arrays, variable $E_{PV}$ and block list $E_{PV}$. Consider two points $p_i$ in $A$-part, and $p_j$ in $B$-part that are on diagonals $f_a$ and $f_b$ respectively. An invariant for any subproblem is that if the optimal chain ending with the fragment that has endpoint $p_j$ has been solved and is referenced as $\text{Score}(p_j)$, the score of chaining $p_i$ with $p_j$ is $\text{Score}(p_i) + \text{gap}(f_a, f_b)$. Because the gap cost only depends on diagonal and not the coordinates of a point, all points in $A$-part on the same diagonal $f_a$ will have the same cost to chain with $p_j$. More generally, the score to chain any starting point in $B$-part to an ending point in $A$-part is equal to the score of chaining a diagonal in $B$ to a diagonal in $A$.

The score of an optimal chain up to point $p_i$ in $B$-part, $E_{PV}[j]$, where $j = \varphi(E_{1i}, f_a)$, is found for column-based problems as:

\[
E_{PV}[j] = \max_{k: D_{1i}[k] \leq E_{PV}[j]} D_{PV}[k] + \text{gap}(\varphi(E_{1i}[j] - D_{1i}[k]))
\]

(4)

For row-based subproblems:

\[
E_{PV}[j] = \max_{k: D_{1i}[k] > E_{PV}[j]} D_{PV}[k] + \text{gap}(\varphi(E_{1i}[j] - D_{1i}[k]))
\]

(5)

The naive approach to solve for all values of $E_{PV}$ scales quadratically as $O(|E_{PV}| \cdot |D_{1i}|)$. A problem of this form:

\[
E[j] = \min_k D[k] + w(k, j)
\]

(6)

where $w$ is a convex function and $D[k]$ is a linear transformation of $E[k]$, was solved in $O(|D| \log |E|)$ time using the auxiliary block data structure $E_{1i}$ (Gusfield and Giancarlo, 1989; Gusfield, 1997). In this solution, iteration is performed with one pass over the $D$ array. After each iteration $i$, the invariant holds: each element in $E$ has been set to reference which element of the prefix $D[1, \ldots, n]$ gives the minimum value for Equation (6). Efficiency is gained by not updating $E$ explicitly but by storing indexes in the data structure $E_{1i}$ that may be updated in $\text{gap}(E[k])$ time by applying a function $\text{Update}(D[i], E_{1i})$ on each iteration. The data structure $E_{1i}$ has an operation that can reconstruct $E$, $E_{1i} = \text{Update}(E_{1i}, b)$, the details of $E_{1i}$ and $\text{Update}$ may be derived from (Gusfield, 1997). In the application to CG-SDP, $D$ is replaced by $D_{PV}$ and $E$ by $E_{PV}$, and Equation (6) and Equation (7) solve for the optimal chains between diagonals from an $A$-part of a subproblem to diagonals in the $B$-part of the column and row subproblem respectively. An important detail in our implementation is that
in column-based problems an Update operation only affects references in \( E_V \) that are on a greater or equal diagonal than the current element in \( D_V \), and for row-based problems elements in \( E_V \) are only affected for diagonals that are less than the current element in \( D_V \). However, it is not possible to apply Equation 4 and Equation 5 directly. Because points are processed in Cartesian order, but \( D_V \) in forward diagonal order, values of \( D_V \) are not solved in increasing order and not all values of \( D_V \) are guaranteed to be solved when values of \( E_V \) are computed. In \cite{1992} this is accounted for by asynchronous computation. Below, an approach is described to solve with a standard model of computation by calling Update(DV[i], EV[i]) using subsets of DV as they become known. Contrary to the customary model of divide and conquer, where subproblems are completely solved before combining into a global solution, this solves portions of subproblems on each iteration.

Points are processed in order of \( x \) and then \( y \) coordinate, determining the value of the optimal chain up to the current processed point. The value of a starting point is the value of the optimal chain that chains the starting point to an endpoint below it. When no endpoints are below a starting point, the value of this starting point is trivially set to 0. The value of an endpoint is the value of the optimal chain up to the current processed point. The value of an endpoint is set to 0 when solving for starting points that are above it, the value of \( D_V \) must be updated in subproblems for which \( p_0 \) is a point in the A-part. The 3-A list is used to determine which subproblems contain \( p_0 \), in a A-part. Suppose the forward diagonal of \( p_0 \) is \( f_1 \). In each subproblem in SA, Score(p0) is passed to \( D_V[i] \) and \( D_V[j] \) is set to \( \text{max}(\text{Score}(p_0), D_V[i], D_V[j]) \), where \( j = \phi(D_V[i], f_1) \) and \( D_V[j] \) is set to \( 0 \) if \( D_V[j] \) gets updated.

When processing a starting point \( p_0 \), the optimal value must be calculated in each of the B-parts of subproblems that include \( p_0 \), and the global optimal value will be selected among them. For any subproblem, this can be achieved by solving for \( E_V[j] \) using Equation 3 for column-based or Equation 5 for row-based subproblems, where \( j = \phi(E_V[i], f_1) \) and \( f_1 \) is the diagonal of \( p_1 \). For a column-based subproblem, this requires that the values of \( D_V[k] \) have been solved where \( D_V[k] \leq f_1 \), and for row-based subproblems, the values of \( D_V[k] \) must have been solved where \( D_V[k] > f_1 \). These correspond diagonals overlapping points that fall in the region below and to the left of the \( p_0 \) and are exactly what have been processed when solving for points in Cartesian sorted order. Thus all required values in \( D_V \) arrays (though not the entire \( D_V \) array) are available when solving for \( p_0 \).

The value of \( E_V[j] \) is optimal once \( \text{Update}(D_V[k], E_V[k]) \) has been called on all diagonals that contain potential predecessors to points on diagonal \( E_V[j] \). The function \( \text{Update}(D_V[k], E_V[k]) \) must be called only once per element in \( D_V \) and in increasing order. However because points are processed in Cartesian sorted order, values of \( E_V \) are solved in arbitrary order, and calling \( \text{Update}(D_V[k], E_V[k]) \) can reference elements in \( D_V \) multiple times. To account for this, we maintain an index \( E_L \) for each subproblem that keeps track of the last element of \( D_V \) which has been processed by \( \text{Update}(D_V[k], E_V[k]) \). Before solving any points, \( E_L \) is initialized to \( 1 \) in every subproblem, and when processing points will be assigned to reference the most recently updated diagonal from \( D_V \). When processing starting point \( p_0 \) in a column-based subproblem, \( E_L[j] \) must be solved, where \( j = \phi(E_L[i], f_1) \). If \( D_V[E_L[j]] > f_1 \), \( \text{Update} \) has been called on all values of \( D_V \) that \( E_V[j] \) relies on, and the state of \( E_V \) contains the optimal value for \( E_V[j] \) such that it may be calculated immediately from \( \Omega(E_V[j]) \). Otherwise, \( \text{Update}(D_V[k], E_V[k]) \) is called for \( E_L[k] < k < C \), where \( D_V[C] \) is the first diagonal from the left that is larger than \( f_1 \) in array \( D_V \), and \( E_V[j] \) may be calculated from \( \Omega(E_V[j]) \). Similarly, for row-based subproblems if \( f_1 \geq D_V[E_L[j]] \), the value of \( E_V[j] \) may be assigned immediately from \( \Omega(E_V[j]) \) where \( j = \phi(E_V[i], f_1) \). Otherwise, \( \text{Update}(D_V[k], E_V[k]) \) is called for \( E_L[k] < k < C \), where \( D_V[C] \) is the first diagonal from the left that is smaller than or equal to \( f_1 \) in array \( D_V \).
and $E_{ij}[j]$ can be retrieved from $D_{ij}E_{ij}$). By comparing values $E_{ij}[j]$ from all the subproblems in the SIE list of startpoint $p_i$, the maximum value for $p_i$ will be obtained and stored. The pseudocode and detailed example of solving points and conquering subproblems are given in the supplementary material (Supplementary Algorithm 1 and Figure S2).

3.6.3 Extension to Inversion Cases

This extension is inspired by the work of [Brudno et al. 2003]. As explained in the previous section, when chaining fragments in forward direction, two points - a lower left start point $(x_{i-1}^l,y_{j-1}^l)$ and a upper right endpoint $(x_{i-1}^r,y_{j-1}^r)$ would be associated to each fragment $a_i$, $(x_{i-1}^l,y_{j-1}^l)$ can be chained to an endpoint below and to the left of it and $(x_{i-1}^r,y_{j-1}^r)$ can be the predecessor of a starting point above and to the right of it. To allow inversions to happen, fragments must be allowed to be chained in the reverse direction. To account for this, we associate two more points to each fragment $a_i$ that are a upper left start point $(x_{i-1}^l,y_{j-1}^r)$ and a lower right endpoint $(x_{i-1}^r,y_{j-1}^l)$. The start point $(x_{i-1}^l,y_{j-1}^l)$ can be only be chained to endpoint $(x_{i-2}^r,y_{j-2}^r)$ of some other fragment $a_j$ that satisfies $x_{i-1}^l < x_{i-2}^r$ and $y_{j-1}^l < y_{j-2}^r$. The endpoint $(x_{i-1}^r,y_{j-1}^r)$ can precede a starting point $(x_{i-2}^l,y_{j-2}^l)$ of some other fragment $a_j$ that satisfies $x_{i-1}^r > x_{i-2}^l$ and $y_{j-1}^r > y_{j-2}^l$. The endpoint $(x_{i-1}^l,y_{j-1}^l)$ can be represented as $s_1(i,j)$, $s_2(i,j)$ and $e_1(i,j)$ respectively. Chaining fragment $a_i$ to fragment $a_j$ in the reverse direction equals to chaining the start point $s_2(a_i)$ to the endpoint $e_2(a_i)$. The cost of appending $s_2(a_i)$ to $s_2(a_j)$ in the reverse direction is $gap^ogap(a_j,a_i)$, which is a convex function of $(x_{i-1}^l+y_{j-1}^l)-(x_{i-2}^r+y_{j-2}^r)$, the difference between the reverse diagonals of fragments $a_j$ and $a_i$. In a short word, $s_2(a_i)$ and $e_2(a_i)$ of each fragment $a_i$ would be responsible for the possible chaining of $a_i$ in the reverse direction, while $s_1(a_i)$ and $e_1(a_i)$ would be responsible for the forward direction.

We used the same column and row subproblem dividing scheme to sort all $s_1$ and $e_1$ points and assign them into column-1 and row-1 subproblems. Then all $s_2$ and $e_2$ points are sorted and assigned to column-2 and row-2 subproblems in the same way. Arrays $D_1$, $D_2$, $D_3$, $D_4$, $E_1$, $E_2$, $E_3$, $E_4$, $E_{V}$, $E_{V'}$, the block structure $E_{0}$ and variable $E_{I}$ are allocated and initialized for each subproblem. Lists $S_{A1}$ and $S_{A2}$ reference column-1/row-1 and column-2/row-2 subproblems that each endpoint $e_1$ and $e_2$ is in respectively. Similarly, lists $S_{B1}$ and $S_{B2}$ reference column-1/row-1 and column-2/row-2 subproblems that each endpoint $s_1$ and $s_2$ in respectively. The only difference between column-2/row-2 and column-1/row-1 subproblems is that column-2/row-2 stores reverse diagonal instead of forward diagonal in $D_1$ and $E_1$ arrays.

The steps to solve subproblems to allow chaining in both forward and reverse directions are highly similar to section 3.6.2. Points are processed in order of $x$ and then $y$ coordinate. When a starting point $s_1(a_i)$ is being processed, $E_{ij}[j]$, where $j = \psi(E_{ij}, f_i)$ and $f_i$ is the forward diagonal, will be computed from column-1/row-1 subproblems in $S_{B1}$. The maximum $E_{ij}[j]$ of those subproblems is the value of optimal chain up to $s_1(a_i)$, $Score(s_1(a_i))$, where $s_1(a_i)$ is forwardly chained to an endpoint $e_1(a_i)$ below and to the left of it. Similarly, when a starting point $s_2(a_i)$ is being processed, $E_{ij}[j]$, where $j = \psi(E_{ij}, r_i)$ and $r_i$ is the reverse diagonal, will be computed from column-2/row-2 subproblems in $S_{B2}$. The maximum $E_{ij}[j]$ of those subproblems is the value of optimal chain up to fragment $a_i$, $Score(s_2(a_i))$, where $s_2(a_i)$ is reversely chained to an endpoint $e_2(a_i)$ above and to the left of it.

After solving $s_1(a_i)$ and $s_2(a_i)$ for fragment $a_i$, the value of the optimal chain up to fragment $a_i$ can be calculated as $Score(a_i) = \max\{Score(s_1(a_i)), Score(s_2(a_i))\} + l(a_i)$, where $l(a_i)$ is the match bonus of fragment $a_i$. This optimal chain is chosen from all the possible chains that fragment $a_i$ is forwardly or reversely chained to the predecessor in the left. When solving $e_1(a_i), Score(a_i)$, will be passed to array $D_2[j]$, where $j = \phi(E_{ij}, f_i)$ and $f_i$ is the forward diagonal, to column-1/row-1 subproblems in $S_{A1}$. And $D_2[j]$ will be updated to the index of point $e_1(a_i)$, if $Score(a_i) > D_2[j]$. Similarly, when $e_2(a_i)$ is being processed, $Score(a_i)$ will be passed to array $D_4$, of column-2/row-2 subproblems in $S_{A2}$ and $D_4$ will be updated. Therefore, the addition of two points $s_2(a_i)$ and $e_2(a_i)$ for each fragment $a_i$ make it possible to allow $a_i$ to be chained in the reverse direction. Meanwhile, the overall time complexity and storage remain bounded by $O((\log(n))^2)$, where $n$ is the total number of fragments.

3.6.4 Time Complexity

Assume there are $n$ fragments in total, list $S_{B1}/S_{A2}$ contains $O(\log(n))$ subproblems that a point is in. In section 3.6.2, we mention that when a starting point $p_i$ with forward diagonal $f_i$ is being processed, $Update(D_1E_{ij}, f_i)$ is called for $E_{ij} < k < C$, where $D_1E_{ij}$ is the first diagonal from the left that is larger than/smaller than or equal to $f_i$, in column/row subproblems. And $E_{ij}[j]$ can be retrieved in $O(\log(n))$ time from the block structure $E_{ij}$ by calling $D_1E_{ij}$, where $j = \phi(E_{ij}, f_i)$. Procedure $Update$ may be called several times for a starting point in each subproblem. In order to make it easy to quantify the total time complexity of $Update$ procedures, we consider that each $Update$ procedure is called right after $D_1E_{ij}$ is updated by some endpoint $p_i$ with diagonal $f_i$, where $j = \phi(E_{ij}, f_i)$. When an endpoint is being solved, there are $O(\log(n))$ subproblems associated with it and in each subproblem $Update$ takes $O(\log(n))$ to conduct. Therefore, it takes $O((\log(n))^2)$ time to solve the subproblems that are associated with an endpoint. When a starting point is being processed, there are $O(\log(n))$ subproblems it is in and in each subproblem $E_{ij}[j]$ can be computed from the block structure $E_{ij}$ in $O(\log(n))$ time. Therefore, it takes $O((\log(n))^2)$ time to tackle subproblems that are associated with a starting point. Since there are $n$ fragments in total, the time complexity of processing all the points and subproblems is bounded by $O(n(\log(n))^2)$.

4 Discussion

The initial description of CG-SDP was given in 1992 in two publications, one covering an affine-cost gap function, and another with a convex-cost gap function [Brudno et al. 1992][Brudno et al. 1992b]. While there have been many implementations of affine cost SDP, no sequence alignment methods have been implemented using affine gap function with a convex-gap function. In the original description of the algorithm, the processing of a starting point is blocked until all subproblems the point relies on are solved, and then the process is unblocked and processing resumes. We find that this blocking and unblocking strategy is not necessary, and the addition of data structures to keep track of the state of computation of subproblems enables solving the problem with a standard model of computation. We used two additional strategies to effectively employ SDP in lra: an iterative refinement process where a large number of anchors from the initial minimizer search are grouped into a small super-fragments that are chained using SDP, and once a rough alignment has been found a new set of matches with smaller anchors is calculated using the local minimizer indexes. As a result, alignment is of similar speed to state of the art algorithms, without the need for single-instruction multiple-data (SIMD) processing; runtime was slower when using an SIMD alignment library [Kotic and Sikić 2017] possibly due to the overhead of invoking the library functions.

The lra alignments have competitive runtime and memory usage compared to minimap2. Using two different SV discovery algorithms, pbvs and Sniffles, we show it is possible to use lra alignments to discover SV using PacBio HiFi, CLAR, and Oxford Nanopore reads, as well as directly from aligned de novo assembly contigs. The performance for SV detection using PacBio reads and the pbvs algorithm is similar between lra and minimap2, with lra demonstrating a small gain in recall over larger SV events. Importantly, the availability of multiple alignment algorithms

"output" — 2020/11/16 — 0:40 — page 10 — #10

XXX XXX.
can help improve the results of studies that require high sensitivity and specificity, such as Mendelian analysis. The greater improvement in SV discovery metrics on ONT data aligned by lra additionally highlights the utility of using multiple algorithms to analyze sequencing data.

Finally, as de novo assembly of genomes becomes more routine, it is important to have accurate methods of SV detection from contig alignments. We used lra to align a haplotype-resolved genome assembled from PacBio HiFi reads and detect SV. When compared to the GIAB SV benchmark data, the lra alignments show slightly lower precision and recall than read-based SV detection. However, when the assembly-based SV are compared to SV detected in an orthogonal read dataset, nearly all (>98.7%) variants discovered from assemblies are supported by read alignments. This indicates few differences between the HiFi based assembly and the reference are due to assembly error, and the annotated precision of the callset is likely an under estimate. A variant callset is effectively a list of operations that may be applied to the reference genome to reconstruct a sample genome. Because the HiFi assembly has few assembly errors on the same size scale as an SV, this supports the development of an alternative model for validating SV callsets in which the reconstructed genome is compared to the haplotype-resolved assembly, rather than by comparing callsets. This may be used to validate calls outside the high-confidence regions defined in In regions that a haplotype-resolved assembly has been confidently generated

5 Acknowledgments
J.R. is supported by the Viterbi Fellowship. M.J.P.C. is supported by NHGRI U24HG007497.

6 Data Availability
The HG002 assembly data is available in NCBI database under GenBank assembly accession number GCA_004796285.1 and GCA_004796485.2. The HG002 ONT reads are available in NCBI database under BioProject accession number PRJNA678534. The HG002 HiFi and CLR reads can be downloaded using links supplied in the supplement. Variant callsets, and a set of curated inversions and CLR reads can be downloaded using links supplied in the supplement. Variant callsets, and a set of curated inversions and CLR reads can be downloaded using links supplied in the supplement. Variant callsets, and a set of curated inversions and CLR reads can be downloaded using links supplied in the supplement.

References
Baker, B. S. and Giancarlo, R. (2002). Sparse dynamic programming for longest common subsequence from fragments. Journal of algorithms, 42(2), 231–254.
Belyeu, J. R. et al. (2020). Samplot: A platform for structural variant visual validation and automated filtering. bioRxiv.
Brudno, M. et al. (2003). Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. Genome research, 13(4), 721–731.
Chaisson, M. J. and Tesler, G. (2012). Mapping single molecule sequencing reads using basic local alignment with successive refinement (bias); application and theory. BMC bioinformatics, 13(1), 238.
Chaisson, M. J. et al. (2019). Multi-platform discovery of haplotype-resolved structural variation in human genomes. Nature communications, 10(1), 1–16.
Eppstein, D. et al. (1992a). Sparse dynamic programming i: Linear cost functions. Journal of the ACM (JACM), 39(3), 519–545.
Eppstein, D. et al. (1992b). Sparse dynamic programming ii: convex and concave cost functions. Journal of the ACM (JACM), 39(3), 546–567.
Fitch, W. M. and Smith, T. F. (1983). Optimal sequence alignments. Proceedings of the National Academy of Sciences, 80(5), 1382–1386.
Gaih, Z. and Giancarlo, R. (1989). Speeding up dynamic programming with applications to molecular biology. Theoretical computer science, 64(1), 107–118.
Gusfield, D. (1997). Algorithms on strings, trees, and sequences: Computer science and computational biology. Acm Sigact News, 28(4), 41–60.
Jain, C. et al. (2017). A fast approximate algorithm for mapping long reads to large reference databases. In International Conference on Research in Computational Molecular Biology, pages 66–81. Springer.
Jain, C. et al. (2018). A fast adaptive algorithm for computing whole-genome homology maps. Bioinformatics, 34(17), i748–i756.
Kent, W. J. et al. (2003). Evolution’s cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. Proceedings of the National Academy of Sciences, 100(20), 11484–11489.
Kolmogorov, M. (2019). Algorithms for long-read assembly. Ph.D. thesis, UC San Diego.
Koren, I. (2018). Computer arithmetic algorithms. CRC Press.
Li, H. (2018). Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics, 34(18), 3094–3100.
Porubsky, D. et al. (2019). A fully phased accurate assembly of an individual human genome. bioRxiv, 855049.
Roberts, M. et al. (2004). Reducing storage requirements for biological sequence comparison. Bioinformatics, 20(18), 3363–3369.
Rowell, W. J. et al. (2019). Comprehensive variant detection in a human genome with highly accurate long reads. In EUROPEAN JOURNAL OF HUMAN GENETICS, volume 27, pages 1723–1723. NATURE PUBLISHING GROUP MACMILLAN BUILDING, 4 CRINAN ST, LONDON N1 9XW, ENGLAND.
Sedlazeck, F. J. et al. (2013). Nextgenmap: fast and accurate read mapping in highly polymorphic genomes. Bioinformatics, 29(21), 2790–2791.
Sedlazeck, F. J. et al. (2018). Accurate detection of complex structural variations using single-molecule sequencing. Nature methods, 15(6), 461–468.
Shafin, K. et al. (2020). Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes. Nature Biotechnology, pages 1–10.
Wenger, A. M. et al. (2019). Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. Nature biotechnology, 37(10), 1155–1162.
Zook, J. M. et al. (2020). A robust benchmark for detection of germline large deletions and insertions. Nature biotechnology, pages 1–9.
Šojić, M. and Šukić, M. (2017). Edlib. A C/C++ library for fast, exact sequence alignment using edit distance. Bioinformatics, 33(9), 1394–1395.