URMAP, an ultra-fast read mapper

Robert Edgar

Unaffiliated, Corte Madera, CA, United States of America

ABSTRACT

Mapping of reads to reference sequences is an essential step in a wide range of biological studies. The large size of datasets generated with next-generation sequencing technologies motivates the development of fast mapping software. Here, I describe URMAP, a new read mapping algorithm. URMAP is an order of magnitude faster than BWA with comparable accuracy on several validation tests. On a Genome in a Bottle (GIAB) variant calling test with 30× coverage 2×150 reads, URMAP achieves high accuracy (precision 0.998, sensitivity 0.982 and F-measure 0.990) with the strelka2 caller. However, GIAB reference variants are shown to be biased against repetitive regions which are difficult to map and may therefore pose an unrealistically easy challenge to read mappers and variant callers.

Subjects Bioinformatics, Computational Biology, Molecular Biology
Keywords Next generation sequencing, Read mapping

Submitted 26 February 2020
Accepted 20 May 2020
Published 24 June 2020

Corresponding author
Robert Edgar, robert@drive5.com

Academic editor
Elliot Lefkowitz

Additional Information and Declarations can be found on page 18

DOI **10.7717/peerj.9338**

Copyright
2020 Edgar

Distributed under
Creative Commons CC-BY 4.0

OPEN ACCESS

INTRODUCTION

Background

Next-generation sequencing has enabled dramatic advances in fields ranging from human functional genomics (*Morozova & Marra, 2008*) to microbial metagenomics (*Gilbert & Dupont, 2011*). Data analysis in next-generation studies often requires mapping of reads to a reference database such as a human genome, human exome, or a collection of full-length microbial genomes. Mapping is a special case of sequence database search where the query sequence is short, database sequences are long, and sequence similarity is high. For a given query sequence (read), the primary goal of mapping is to report the best match if possible, otherwise to report that the best two or more alignments are sufficiently similar to each other that the best match is ambiguous.

Prior work

Many mapping algorithms have been proposed. Representative examples include BWA (*Li & Durbin, 2009*), Bowtie (*Langmead et al., 2009*), Bowtie2 (*Langmead & Salzberg, 2012*), SOAP (*Li et al., 2008*), SOAP2 (*Li et al., 2009b*), Minimap2 (*Li, 2018*), FSVA (*Liu, Wang & Wang, 2016*), SSAHA (*Ning, Cox & Mullikin, 2001*), Hisat2 (*Kim et al., 2019*) and SNAP (*Zaharia et al., 2011*). Mappers utilizing the Burrows-Wheeler Transform (BWT) (*Burrows & Wheeler, 1994*) are the current *de facto* standard, with BWA and Bowtie2 in particular having more than 39,000 citations combined at the time of writing (Google Scholar accessed 31st Dec 2019). When first utilized in read mapping, BWT had the important advantage that it creates a compact index with size comparable to the reference database. For the human genome, this is ∼3 GB, which is small enough to be stored in RAM with the commodity computers of that time. Currently, computers with 32 GB or more RAM are readily available, which has raised the question of whether additional memory could enable better mapping performance. In particular, the authors of SNAP claim (*Zaharia et al., 2011*) that its use of a ~27 GB hash table index for the human genome gives both faster speed and higher accuracy than BWA. FSVA also uses a hash table, reportedly (*Liu, Wang & Wang, 2016*) achieving faster speed than BWT though with somewhat lower accuracy.

URMAP algorithm

URMAP uses a hash table index on k-mers, i.e., fixed-length words of length k, where $k = 24$ is recommended for the human genome. The index is designed to keep information relating to a given hashed word (*slot*) close together in RAM to minimize memory cache misses. Slots found exactly once in the reference (*pins*) are flagged. For a given query, URMAP first searches for a pair of non-overlapping pins which are close together in the reference (a *brace*, see Fig. 1). If a brace is found, an alignment is attempted and the search terminates immediately if successful. Otherwise, a seed-and-extend strategy (*Altschul et al., 1990*) is followed which prioritizes low-abundance slots.

Performance testing

Recent assessments of mapping accuracy, in particular those of SNAP and FSVA, have used the wgsim program in the SAMtools package (*Li et al., 2009a*) to simulate reads of a human genome. Mutation rates (more correctly, variation rates) of 0.1% were used in both cases, with 0.09% single-nucleotide polymorphisms (SNPs) and 0.01% indels. The base call error rate was set to 0.4% for testing FSVA and to various different values for testing SNAP. Differences, i.e., base call errors, SNPs and indels, are introduced by wgsim with equal probability for each type at each position, giving a Poisson distribution for inter-difference spacing where closely-spaced SNPs and base call errors are rare. With a mutation rate of 0.1%, most reads of length 150nt simulated by wgsim have no mutation, and most reads with mutations have exactly one single-base variant. In real human genomes, variants tend to cluster, e.g., in non-coding regions (*Altshuler et al., 2010*; *Montgomery et al., 2013*). Thus, average accuracy over all reads on a wgsim test gives little insight into mapper performance on the more challenging, and often more biologically interesting, reads with multiple differences compared to the reference. Ilumina base call errors also tend to cluster, for example towards the end of a read (*Minoche, Dohm & Himmelbauer, 2011*), and in practice there are therefore many more reads with multiple errors than a Poisson distribution would predict.

Urbench performance test

In this work, I introduce Urbench, a new benchmark test using experimentally determined variation from a well-characterized human genome. Simulated read sequences are combined with quality scores from a recent 2×150 Illumina run. At each base, a substitution error is introduced with the probability implied by its quality score, with the goal of generating a more realistic distribution of base call errors compared to earlier benchmarks. Mapping sensitivity and error rates are measured separately on reads which do, or do not, contain variants. Systematic errors are identified where most reads of a given locus are

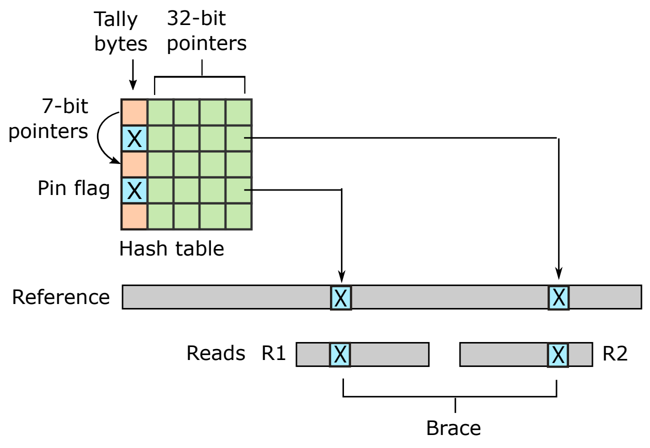


Figure 1 Schematic of the URMAP algorithm. Words in the plus strand of the reference sequence are indexed using a hash table with 5 bytes per row comprising a tally byte and a 32-bit pointer. Pins, i.e., words with a hash value that is unique across both strands of the reference, are indicated by a reserved tally value (pin flag). URMAP searches for a brace, i.e., a pair of pins close in the reference, one in the forward read (R1) and one in the reverse read (R2). If a brace is found, it is almost certain to be the correct location. Words found more than once in the reference are indexed using a linked list with forward pointers which are stored in tally bytes if they fit into 7 bits, otherwise in the 32-bit pointer field. The first bit of the tally is set if the row is in a list but not the head.

Full-size DOI: 10.7717/peerj.9338/fig-1

mapped to the same incorrect locus. Systematic errors are presumably more likely to cause false-positive inferences in later analysis than errors spread over many incorrect positions.

METHODS

URMAP index

Hash table

The positions of words of length k in the plus strand of the reference are stored in a hash table. A word W is converted to an integer $w(W) \in [0, 4^k)$ in the usual way by considering letters to be base-4 digits $A = 0$, $C = 1$, $G = 2$ and $T = 3$. The murmur64 hash function (https://en.wikipedia.org/wiki/MurmurHash) is used to convert w to an integer (*slot*) $s \in [0, H)$ where H is the table size by $s = \text{murmur64}(w(W)) \bmod H$. For brevity, a word with a given slot value will be referred to simply as a slot. To reduce collisions, the table size H should be a prime number substantially larger than the reference; for the human genome $H = 5 \times 10^9 + 29$ is recommended. The table design is intended to minimize size and optimize adjacency of data relating to a given slot with the goal of avoiding memory cache misses.

Hash table row

Each hash table entry (*row*) is five bytes: one byte (the *tally*) containing a one-bit flag and sometimes a 7-bit pointer to another row, and a four-byte value which is usually a 32-bit reference coordinate, or rarely a pointer to another row. Using 32-bit coordinates limits the total reference sequence size to $2^{32} = 4$ GB; references up to 1024 GB could be accommodated by using five pointer bytes.

Pins

A *pin* is a slot found exactly once in the reference, considering both plus and minus strands. This is an important special case because if a pin is found in a read it probably maps correctly to the same pin in the reference, though it may also be a false positive due to sequencing error or a genome variant. A pin is indicated by a reserved tally value. The term ”pin” was chosen by analogy with a metal (or virtual) pin used to mark a location on a paper (or online) geographical map.

Singletons

A *singleton* is a slot that is found exactly once in the plus strand of the reference and one or more times in the minus strand. A singleton is also indicated by a reserved tally value. Note that while the minus strand is not indexed, there is nevertheless an important distinction between pins and singletons because the reverse-complement of a singleton occurs at least once in the plus strand of the reference, while the reverse-complement of a pin does not occur. Thus, while a pin found in a read implies only one candidate alignment to the plus strand of the reference considering both strands of the query, a singleton implies at least two candidates.

Linked lists

To store positions of a slot occurring more than once in the plus strand of the reference, a linked list is stored in nearby empty rows. Where possible, 7 bits of the tally are a pointer to the next row in the list, represented as the number of rows to skip. Overflows where this number does not fit into 7 bits are handled by storing a pointer to the next row in the 32-bit value instead of a reference coordinate. Overflows and list ends are indicated by reserved tally values.

Over-abundant slots

Slots exceeding an abundance threshold are excluded from the index. This is a speed optimization to avoid constructing a large number of candidate alignments. The loss in sensitivity is small because abundant slots are usually found in repetitive sequence which maps ambiguously unless there is distinctive sequence elsewhere in the read, and unique reference sequence of length $\geq k$ necessarily contains a pin. By default, the abundance threshold t is set to 32. A slot that occurs more than t times on either reference strand is excluded. The minus strand is also considered in order to exclude cases where a repeat occurs with high abundance on the minus strand but low abundance (in particular, only once) on the plus strand. If a slot in such a repeat were indexed, this would tend to lead to an over-estimate of the probability that one of its plus strand alignments is correct because high-scoring secondary alignments to the minus strand would not be discovered.

Absent slots

A slot that does not occur in the reference, or is not indexed because it is over-abundant, is *absent*, as opposed to a slot which is present in the index. An absent slot may appear in a read, and the index must therefore indicate that the corresponding row does not contain a reference coordinate for that slot. This is accomplished by the first bit of the tally, which is set to one for present slots and zero for absent slots. The reserved tally values for pins and singletons have the first bit set to one to indicate that these slots are present, and the reserved values for 7-bit pointer overflows and the end of a linked list start with a zero bit because these slots were found to be absent and the corresponding rows were therefore available for use in a list. Linked list pointers in tally bytes are limited to 7 bits to ensure that the first bit is zero.

Collisions

A hash table collision occurs when two different reference words have the same slot value. A collision between the query and index occurs when a word in the read is different from a word in the index and has the same slot value. Collisions are not represented in the index or explicitly checked during search. This strategy saves index space without compromising search time because in the rare cases where a collided slot is aligned, the alignment will be abandoned quickly due to excessive mismatches in the flanking reference sequence. When aligning, it is faster to check the flanking sequence first than to verify that the seed matches because in the typical (non-collision) case the seed always matches while flanking sequence often does not.

Word length

Increasing k increases the frequency of pins in the reference and also increases the number of words per query that are changed by a difference and hence the probability that a read does not contain a pin, or any indexed slot, due to read errors and variants. The choice of k is thus a compromise between speed and sensitivity. With the human genome, $k = 24$ is recommended because of the ~3G 24-mer slots, ~0.9G (30%) are pins, and on average a reference segment of length 150nt contains 38 pins. A query sequence with <7 single-base differences is guaranteed to have at least one 24-mer match, noting that 7 differences eliminate all 24-mer matches only in the tiny fraction of possible distributions where they are maximally disruptive, and reads with ≥7 differences will often have at least one preserved 24-mer.

URMAP search algorithm

Query word search order

With 24-mers, query words in a read of length 150 are processed at intervals (*strides*) of length 29 using modulo 127 to keep the position within the read (because there are 127 24-mers in a read of length 150). For example, the first three words processed are at positions 0, 29 and 58. At a given position, both strands are considered, so for example the words at the first position (zero) in both plus and minus strands are both processed before moving on to the plus and minus words at position 29. The stride value 29 is chosen to be relatively prime with 24, which ensures that the following loop will visit each query word exactly once:

for (int *j* = 0; j <127; ++j) { QueryWordPosition = (29*j)%127; /* ... */ }.

The simple form of this loop without conditional branches may enable loop unrolling or vector parallelization by the compiler, and regardless is designed to be efficiently executed on modern processors. In general, given the word length k, the stride is identified as the smallest prime number with value $\geq k$ +5. The use of a stride $>k$ is motivated by the observation that neighboring words are not independent. If a query word is not a pin, fails to align, or is not indexed, this is likely to be because the word is in a repetitive region or variant, or contains a sequencing error. The immediately following words are likely to have the same problem, and the chances of finding good alignments early in the search are improved by skipping ahead.

First pass: brace search

In its first pass through the query words, URMAP seeks a pair of pins that are close together in the reference. As noted in the Introduction, such a pair is called a brace (Fig. 1). This term was chosen because the noun ”brace” has two relevant meanings: two of a kind, and a device that connects, fastens or stabilizes. With paired reads, one pin is sought in the forward read (R1) and the other in the reverse read (R2). While a pin may be a false positive due to sequencing error or a variant, a brace is almost certain to be a true positive match. If a brace is found and is aligned successfully (is a *good brace*), the search terminates immediately. Previous read mapping algorithms do not terminate when the first high-scoring alignment is found, even if it has no mismatches, because equally high-scoring alignments may exist elsewhere in the reference. By contrast, when a good brace is found the likelihood that a different position is correct is vanishingly small. Noting that a typical read contains several pins, and most pins are true positives, the search for a good brace in a read pair proceeds as follows, with the goal of minimizing the number of hash table accesses and attempted alignments in typical cases. The first pins in both reads (the forward and reverse read, known as R1 and R2 respectively) are identified. If this pair is not a good brace, the next pin is identified in R1, giving a new potential brace, then the next pin in R2, and so on. This process continues until a good brace is found or all words in both reads have been processed. Almost all pin pairs which are not braces can be

Identified as such because they are too far apart in the reference, which requires only the coordinates in their hash table rows. It is very rare for non-overlapping false positive pins to appear close in the reference, and therefore brace tests almost never fail in the more expensive alignment stage. Since most human 2$\times$150 read pairs contain a correct brace, the brace search pass identifies the correct reference coordinate for most reads with remarkable efficiency.

Second pass: low-abundance slot search

If no brace is found, the hash table row for each query word has been accessed exactly once. Each row access almost certainly triggers a memory cache miss because of the large size of the hash table (~25 GB for the human genome). To accelerate access to these rows in subsequent passes, the first pass copies them to a small (few kB) per-thread buffer (PTB). Other data which may be used repeatedly, such as query slot values and the reverse-complemented query sequence, is also stored in the PTB, which is designed to be compact and contiguous to maximize the chance that it will be available in a fast memory cache. The second pass attempts to align all non-pin slots with abundance ≤ 2. Most of the index data needed for this task is already present in the PTB, though some additional rows may be required for slots with abundance two. If a high-scoring alignment is found in the second pass, the search terminates.

Third pass: high-abundance slot search

In the rare case that no high scoring alignment is found in the first two passes, alignments are attempted for the remaining slots.

HSP construction

Following BLAST (*Altschul et al., 1990*) and many subsequent algorithms, URMAP constructs ungapped alignments using a seed-and-extend strategy. The seed is an indexed slot found in the query, which implies an alignment of length k. The seed is extended into flanking sequence using gapless x-drop alignment which stops if the score falls more than x below the maximum so far observed (x is a heuristic parameter). If the score exceeds a threshold, the alignment is designated a high-scoring segment pair (HSP) and stored, otherwise the reference location is added to a list of failed extensions. The lists of HSPs and failed locations are consulted before extending to prevent redundant attempts to align the same reference location. If the HSP covers the entire query sequence, then the alignment is considered successful.

Gapped alignments

In the human genome, indel variants are rare (*Altshuler et al., 2010*), and Illumina indel errors are very rare (*Schirmer et al., 2015*), and therefore a large majority of correct alignments of human reads are expected to be gapless. Computing an ungapped alignment is much faster than a gapped alignment, and URMAP therefore constructs gapped alignments only if no HSP covers the query. Gapped alignments are constructed by extending the top few HSPs into semi-global alignments using a variant of the Viterbi algorithm (*Viterbi, 2006*) where the alignment is constrained to include the HSP and the terminal regions are banded, greatly reducing the number of dynamic programming matrix cells which must be computed. Here, semi-global means that the entire query sequence must be included but not the entire reference.

MAPQ calculation

MAPQ is an integer value representing the estimated probability P_{error} that the reference coordinate of the top-scoring alignment is wrong,

$P_{error} = 10^{-\text{MAPQ}/10}$.

Let T be the score of the first alignment in order of decreasing score, and $S \leq T$ be the score of the second alignment. If only one alignment is found, S is set to $T/2$ as a prior estimate of the second-best score rather than zero because the URMAP search algorithm may terminate early if a high-scoring alignment is found. The first alignment is likely to be correct (P_{error} is small) if $T \gg S$, and conversely P_{error} is at least $\sim$0.5 if $T \approx S$ (because if exactly two alignments X and Y have equal scores, there is a 1/2 chance that X is wrong; 2/3 chance if there are three, and so on). Thus MAPQ should increase monotonically with T - S. Also, MAPQ should decrease monotonically with decreasing T because alignments with more differences are less likely to be correct. The best possible alignment score is the query sequence length $|Q|$ because identities contribute 1 to the score, and the ratio $T/|Q|$ therefore ranges from one to zero as the top alignment score ranges from best possible to worst possible. This ratio is a natural choice to down-weight $T - S$, and a simple formula with the desired properties is MAPQ = $(T- S)\ T/|Q|$. Empirically, using $(T/|Q|)^2$ rather than $T/|Q|$ was found to give a more accurate estimate, and URMAP therefore uses

MAPQ = $(T - S)\ (T/|Q|)^2$.

I am aware of no justification for this formula beyond its empirical success and the intuitive considerations above. However, the simplicity of this formula and its lack of tuned parameters (except perhaps the power of $T/|Q|$) suggest that this or a similar result may be derivable from more rigorous theoretical considerations.

URMAPv algorithm

Some applications are more tolerant of mapping errors than variant calling, for example nucleosome position inference in cell-free DNA (*Snyder et al., 2016*). With this in mind, I sought a set of parameters for URMAP giving faster execution time while maintaining useful accuracy. The most important speed optimization is reducing the maximum indexed slot abundance t from 32 to 3. Other optimizations include tweaks to heuristic parameters which trigger early termination of various search stages, such as the x in x-drop. Here this algorithm is called URMAPv; it is invoked by the *-veryfast* command-line option. In practice, the execution time of URMAPv is often dominated by file i/o, and thus represents a point of diminishing returns in speed optimization for mapping.

Tested methods

The following methods were tested: BWA v0.7.17-r1188, Bowtie2 v2.3.4.1, SNAP v1.0beta.24, FSVA GitHub commit 8cec132 (dated Jul 29, 2016), Minimap2 v2.17-r94, Hisat2 2.1.0 and URMAP v1.0.1300. This is a somewhat arbitrary selection from the many published methods designed include the most popular software (BWA and Bowtie2) together with potentially competitive recently-published methods. The beta version of SNAP was used because the release binaries failed on some tests.

Urbench benchmark

I implemented a benchmark, Urbench, which models mapping of shotgun 2×150 Illumina reads, i.e., the current *de facto* standard, to the human reference genome. Variants and sequencing error were introduced based on experimental results rather than by simulating Poisson distributions as in previous benchmarks.

Reference sequence

Genome Reference Consortium Human genome build 38 (GRCh38) (*Church et al., 2011*) was used as the reference sequence.

Variant genome

I chose to use NA12878, a well-studied human genome from the Platinum Genomes project (*Eberle et al., 2017*). Simulated variants were selected from variants in NA12878 identified by barcoded long-molecule sequencing (*Zhang et al., 2019*). Many of these variants are phased, i.e., assigned to a parental chromosome, over regions of tens to hundreds of kb.

Unphased variants were randomly assigned to a parental chromosome. While these variants may be less reliable than consensus predictions derived from a range of methods, I believe that their distribution in the diploid genome is more realistic than a consensus because long-molecule sequencing enables phasing and also mapping to repetitive sequence which is inaccessible to conventional short-read methods (see Unmappable Regions below). For the Urbench test, false positives and incorrectly phased variants in NA12878 are unimportant because a reference variant is present in the simulated genome regardless of whether it is truly present in NA12878, while failing to introduce variants in repetitive regions would result in an unrealistically easy test.

Simulated read pairs

Two source genome sequences were used to simulate reads: the reference genome GRCh38 and variant genome NA12878. Using the reference models a situation where the correct sequence for each read is identical to the reference (i.e., does not contain a variant), which is probably the case for most reads in practice. For each genome, 1M loci were selected. For each locus, ten read pairs were simulated at random positions such that R1 or R2 contained the locus (Fig. 2). This enables systematic errors to be identified, i.e., cases where the majority of reads for a given locus are assigned to the same incorrect reference segment. With NA12878, each locus was the position of a variant so that all simulated read pairs contains at least one variant. With GRCh38, loci were randomly selected positions.

Sequencing error

Simulated read sequences were combined with quality scores from run SRR9091899 in the Sequence Read Archive (*Leinonen, Sugawara & Shumway, 2011*), which is a recent (submitted 2019) 2×150 Illumina shotgun dataset. At each base, a substitution error was introduced into the nucleotide sequence with the probability implied by its quality score.

Accuracy metrics

Per-read sensitivity S_r is defined as the fraction of reads which are mapped to the correct coordinate with high confidence as reported by the mapper. Following the BWA paper, high confidence was determined as MAPQ $\geq$10, corresponding to P_{error} $\leq$0.1. Per-read error E_r is defined as the fraction of reads with MAPQ $\geq$10 which are mapped to an incorrect position. Per-locus sensitivity S_l is defined as the fraction of loci where at least three reads have MAPQ $\geq$10 and the majority of these are mapped to the correct position. Per-locus error rate E_l is defined as the fraction of loci with at least three reads having MAPQ $\geq$10 where the majority of these are mapped to the same incorrect position. E_l is interpreted as assessing systematic errors that are more likely to be harmful to downstream analysis than randomly-distributed errors. These metrics are measured separately for reads of the reference (*ref*) and of the variant genome (*var*) giving a total of eight accuracy metrics S_r^{ref}, E_r^{ref}, S_l^{ref}, E_l^{ref}, S_r^{var}, $E_r^{var} S_l^{var}$ and E_l^{var} which are expressed as percentages.

Pairwise method comparison

To enable a compact summary comparison of the eight accuracy metrics for a pair of methods X and Y, I defined the *mean improvement* of X over Y (MI_{XY}) as the mean of

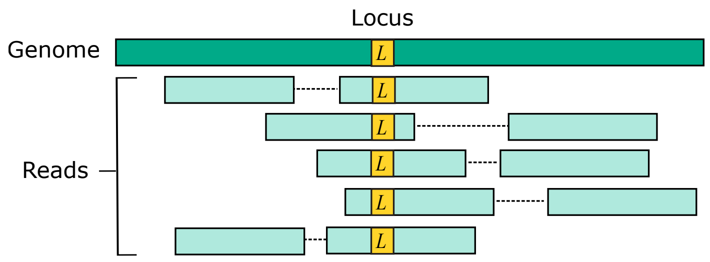


Figure 2 Design of the Urbench benchmark. For each locus *L* in a source genome (NA12878 or GRCh38), ten simulated reads pairs are generated (five shown in figure) such that either R1 or R2 contains the locus. This enables systematic errors to be identified where a majority of reads of a given locus are mapped to the same incorrect location. With NA12878 a locus is the position of an experimentally determined variant (SNP or indel) in one of the parental chromosomes, with GRCh38 a locus is a randomly-chosen position. Base call substitution errors are introduced with probabilities given by quality scores in sequencing run SRR9091899.

Full-size DOI: 10.7717/peerj.9338/fig-2

S_X - S_Y - E_X + E_Y over the four combinations of genome (*ref* or *var*) and read or locus (*r*, *l*) and the *improved metric count* (IM$_{XY}$) of *X* vs. *Y* as the number out of the eight metrics where *X* has a better value than *Y* (higher if sensitivity, lower if error rate). If IM$_{XY}$ is 8, then all the metrics for *X* are better than *Y*, and *X* is unambiguously better than *Y* by the Urbench test, denoted *X*>>*Y*. Conversely if IM$_{XY}$ is zero then all metrics for *X* are worse, MI$_{XY}$ is negative, and *X* is worse than *Y*, denoted *X*<<*Y*. If six out of eight *X* metrics are better; this is denoted by X>6*Y*, if five out of eight *X* metrics are worse, this is written *X*<5*Y*. The magnitude of the improvement is indicated by MI and written in parentheses, e.g., *X*<<(-1.2)*Y* or *X*>6(4.0)*Y*. The total improvement (TI$_X$) of method *X* over the other tested methods is calculated as the total of MI over pairwise comparisons with other methods.

Speed

The time required to map a given set of reads depends on the computer architecture (e.g., the processor type, number of cores, and sizes and speeds of L1 and L2 memory caches) and overhead due to file input/output (i/o). Reads are typically provided in compressed FASTQ format (fastq.gz extension), which requires potentially expensive decompression, and output is typically written to large SAM (uncompressed) or BAM (compressed) files. The overhead of file i/o (including decompression and/or compression, if applicable) can be substantial for the faster mappers and varies widely with the computing environment. With this in mind, I chose to measure speed using a method designed to isolate mapping by reducing i/o overhead as much as possible, as follows. 10M reads were selected at random from SRR9091899, giving a total of ∼2.4 GB compressed (∼6GB uncompressed) FASTQ data (∼1.2 GB 3Gb each for R1 and R2). These files are small enough to be cached in memory by the operating system, which minimizes i/o time on a computer with sufficiently large RAM. Using a PC with a 16-core Intel i7-7820X CPU and 64 GB RAM (more than twice the size of the largest genome index), I ran each mapper three times in succession using from 12 to 20 threads, first with uncompressed then compressed reads, and selected the shortest time after subtracting the time required to load the genome index. Speed is expressed as a multiple of the shortest time for BWA, i.e., the speed of BWA is 1.0 by definition. With this method, the measured relative speed of two mappers can be interpreted as a limit on the ratio in practice which would be approached by the fastest possible i/o.

Accuracy of MAPQ

The accuracy of MAPQ was assessed as follows. For each integer value (q) of MAPQ reported by a mapper, the total number of reads n^q and number of incorrectly mapped reads n^q_{error} were calculated, giving the measured error frequency $f^q_{error} = n^q_{error}/n^q$ and the measured mapping quality at the reported quality q is then

$$q_{measured} = -10 \log_{10} (f^q_{error}).$$

If the MAPQ values are accurate, then $q_{measured}$ should be $\sim q$ for all reported values of q, which was assessed by constructing a scatterplot of $q_{measured}$ against q.

Wgsim validation

For comparison with previous work, I implemented a simulated dataset using wgsim v1.7 with three reference genomes: *Homo sapiens*, *Drosophila melanogaster* and *Arabidopsis thaliana* using GenBank (Benson et al., 2013) assemblies GRCh38, GCA_004798075.2 and GCA_000835945.1 respectively. For each genome, 1M paired reads were simulated at each of three different lengths: 150, 250 and 300 nt. Default parameters were used for wgsim: base call error rate 0.02, mutation rate 0.001, indel fraction 0.15 and indel extension probability 0.3. The random number seed was fixed using the -S 1 option to enable reproduction of the dataset. Accuracy was measured using the S_r and E_r metrics as defined for Urbench.

Variant calling test

To validate that URMAP is compatible with variant calling, I implemented a test based on Genome in a Bottle (GIAB) reference data (Krusche et al., 2019; Zook et al., 2014). I used 30× coverage 2×150 HiSeq reads of sample HG002 (Ashkenazim son, reads at ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/NIST_HiSeq_HG002_Homogeneity-10953946/HG002_HiSeq300x_fastq/140528_D00360_0018_AH8VC6ADXX/Project_RM8391_RM8392 via Zook et al., 2016). To call variants, I used bcftools in samtools v1.7 and strelka2 v2.9.10 (Kim et al., 2018). Accuracy of variant calls was measured using the vcfeval command in rtg-tools v3.11 (Cleary et al., 2015), which reports the following metrics: number of true positives (TPs), number of false positives (FPs), number of false negatives (FNs), precision, sensitivity and F-measure. Evaluation was restricted to the high-confidence regions defined in https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002_NA24385_son/latest/GRCh38/HG002_GRCh38_GIAB_highconf_CG-Illfb-IllsentieonHC-Ion-10XsentieonHC-SOLIDgatkHC_CHROM1-22_v.3.3.2_highconf_noinconsistent.bed via Zook et al. (2016).

Unmappable regions

I used the following procedure to identify and characterize regions in GRCh38 which cannot be mapped by conventional paired-read sequencing. Paired reads of GRCh38 were simulated at 10× coverage with no variants or sequencing error, representing a best-case scenario for mapping. For a given mapper, read length L and MAPQ threshold Q, a base in GRCh38 was considered mappable if at least one read with MAPQ $\geq Q$ covered the base, otherwise the base was designated unmappable. Unmappable regions were identified as contiguous segments of unmappable bases of length $\geq L$ containing <10 uncalled bases (i.e., Ns or other wildcard letters) per window of length L. The consensus of two or more mappers was obtained by intersecting their unmappable regions using bedtools v2.26 (*Quinlan & Hall, 2010*). Unmappable regions were intersected with the GIAB high-confidence regions in HG002 and the following repeat annotation tracks in the UCSC human genome browser (*Kent et al., 2002*): genomicSuperDups, repeatMasker and simpleRepeats.

RESULTS

Speed and mapping accuracy on Urbench

Speed and accuracy on Urbench are shown in Figs. 3, 4 and 5 (underlying values reported in Supplementary Tables 1 and 2). As with most benchmarks in biological sequence analysis, results should be interpreted with caution because of the limitations of simulated data and the many somewhat arbitrary decisions that must be made in designing the benchmark and its performance metrics; other defensible designs would no doubt give somewhat different method rankings and numerical values for sensitivity and error rates. With these caveats in mind, some general trends can be observed. URMAP and URMAPv are the fastest methods, with URMAP ~10× faster than BWA and Bowtie and URMAPv ~28× faster, noting that in practice the speed improvement may be less due to file i/o overhead. Hisat2 (~9×) and SNAP (~8×) have similar speed to URMAP. All methods were faster with uncompressed FASTQ, showing that the added time for decompression exceeds the time saved by reading smaller files. Four methods, BWA, URMAP, SNAP and Bowtie2, stand out as more accurate than the others (Minimap2, Hisat2, URMAPv and FSVA) because all methods from the first group have at least 6 better metrics (shown as >6, >7 or >>in Fig. 5) with a positive mean improvement compared to all methods in the second group with the exception of SNAP >5(3.4) URMAPv. The four top methods (BWA, URMAP, SNAP and Bowtie2) have similar accuracy, with no method having more than 6 out of 8 better (hence 3 or 4 worse) accuracy metrics than another, and all pairwise comparisons exhibit only small mean improvements ranging from BWA >5(0.5) URMAP to BWA >5(2.0) Bowtie2. Thus, the accuracy differences between BWA, URMAP, SNAP and Bowtie2 are small and ambiguous, and I believe these differences are unlikely to be consequential in practice for most applications.

MAPQ accuracy on Urbench

Figure 6 is a scatterplot of reported vs. measured MAPQ on Urbench. Hisat2 is not shown because it generated only three distinct MAPQ values: MAPQ = 0 (measured MAPQ = 1.8), MAPQ = 1 (measured 2.0) and MAPQ = 60 (measured 18). Bowtie2 and URMAP are close to the diagonal, showing reasonably good estimates of MAPQ though Bowtie2 tends to underestimate and URMAP tends to overestimate. The other tested methods have

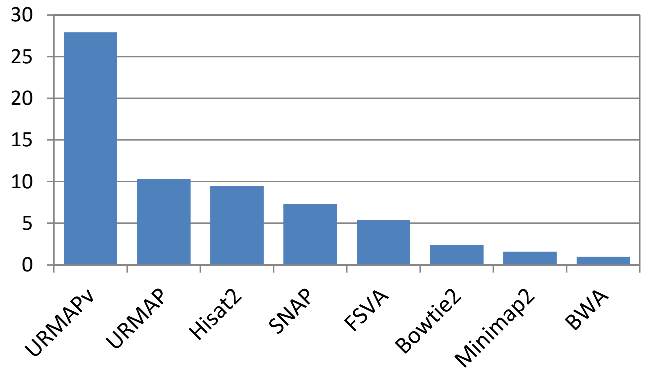


Figure 3 Speed on Urbench. Speed is measured relative to BWA with file i/o overhead minimized.
Full-size DOI: 10.7717/peerj.9338/fig-3

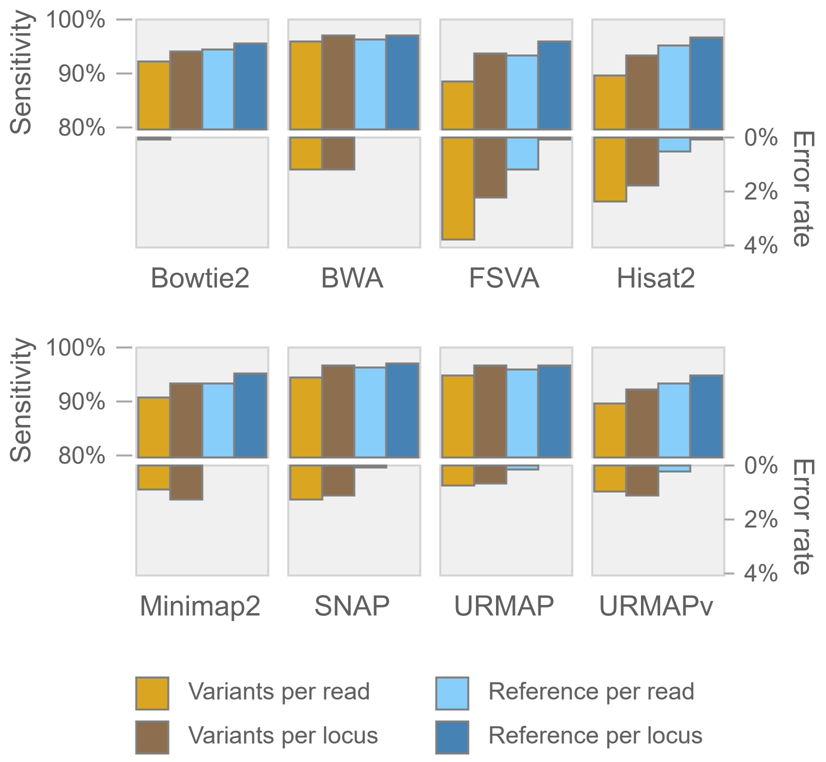
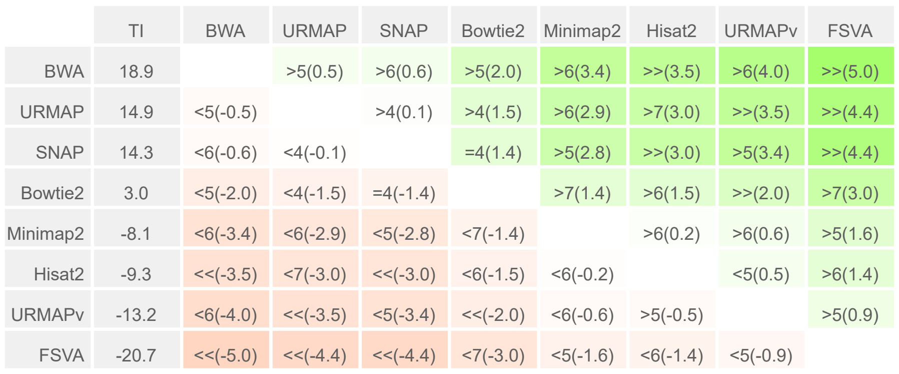


Figure 4 Mapping accuracy on Urbench. Accuracy metrics are sensitivity and error rate with MAPQ ≥10, expressed as percentages.
Full-size DOI: 10.7717/peerj.9338/fig-4

much stronger tendencies to overestimate. For example, with MAPQ = 50, the measured MAPQ for BWA is 15.7 and the measured MAPQ for SNAP is 5.6.



	TI	BWA	URMAP	SNAP	Bowtie2	Minimap2	Hisat2	URMAPv	FSVA
BWA	18.9		>5(0.5)	>6(0.6)	>5(2.0)	>6(3.4)	>>(3.5)	>6(4.0)	>>(5.0)
URMAP	14.9	<5(-0.5)		>4(0.1)	>4(1.5)	>6(2.9)	>7(3.0)	>>(3.5)	>>(4.4)
SNAP	14.3	<6(-0.6)	<4(-0.1)		=4(1.4)	>5(2.8)	>>(3.0)	>5(3.4)	>>(4.4)
Bowtie2	3.0	<5(-2.0)	<4(-1.5)	=4(-1.4)		>7(1.4)	>6(1.5)	>>(2.0)	>7(3.0)
Minimap2	-8.1	<6(-3.4)	<6(-2.9)	<5(-2.8)	<7(-1.4)		>6(0.2)	>6(0.6)	>5(1.6)
Hisat2	-9.3	<<(-3.5)	<7(-3.0)	<<(-3.0)	<6(-1.5)	<6(-0.2)		<5(0.5)	>6(1.4)
URMAPv	-13.2	<6(-4.0)	<<(-3.5)	<5(-3.4)	<<(-2.0)	<6(-0.6)	>5(-0.5)		>5(0.9)
FSVA	-20.7	<<(-5.0)	<<(-4.4)	<<(-4.4)	<7(-3.0)	<5(-1.6)	<6(-1.4)	<5(-0.9)	

Figure 5 Pair-wise method comparisons on Urbench. Methods are sorted by decreasing total improvement (TI) (see Methods). Cells are colored according to mean improvement. A pairwise comparison of the method in row X vs. the method in column Y is given using the notation described in Methods; e.g., BWA >5(2.0) Bowtie2 means that BWA has five of eight metrics that are better than Bowtie2 with a mean improvement of 2.0. The symbols >> and << indicate that all metrics are better or worse, respectively, e.g., URMAP >>(4.4) FSVA means that URMAP is better than FSVA by all metrics with a mean improvement of 4.4.

Full-size DOI: 10.7717/peerj.9338/fig-5

Table 1 Accuracy on GIAB variant calling test. Accuracy metrics according to the rtg vcfeval command when variants are called by strelka2 (top) and bcfeval (bottom). Bowtie2 + strelka2 exhibits anomalously poor performance. FSVA failed with a segfault on this test.

+Strelka2	TP	FP	FN	Precision	Sensitivity	F-measure
Bowtie2	2.081M	0.052M	1.425M	0.975	0.594	0.738
BWA	3.475M	0.012M	0.031M	0.997	0.991	0.994
Hisat2	3.421M	0.017M	0.084M	0.995	0.976	0.985
Minimap2	3.454M	0.011M	0.052M	0.997	0.985	0.991
SNAP	3.466M	0.012M	0.040M	0.997	0.989	0.993
URMAP	3.442M	0.008M	0.064M	0.998	0.982	0.990
URMAPv	3.339M	0.001M	0.167M	0.997	0.953	0.974

+Bcftools	TP	FP	FN	Precision	Sensitivity	F-measure
Bowtie2	2.840M	0.191M	0.665M	0.937	0.810	0.869
BWA	3.306M	0.103M	0.199M	0.970	0.943	0.956
Hisat2	3.229M	0.084M	0.277M	0.975	0.921	0.947
Minimap2	3.286M	0.076M	0.219M	0.977	0.938	0.957
SNAP	3.271M	0.098M	0.234M	0.971	0.933	0.952
URMAP	3.334M	0.098M	0.172M	0.972	0.951	0.961
URMAPv	3.21M	0.080M	0.381M	0.800	0.894	0.844

Mapping accuracy on wgsim test

Figure 7 shows accuracy results on the wgsim test (underlying values in Table S1). Most methods exhibit similar or improved accuracy as read length (L) increases, with the exception of Hisat2 which has substantially lower sensitivity at $L = 250$ and $L = 300$.

Table 2 Re-scaled MAPQ results. Accuracy metrics of variants called by strelka2 and bcftools respectively given Bowtie2 alignments of chr20 when MAPQ values are multiplied by the factor shown in the first column. Note that accuracy metrics are sensitive to MAPQ scaling.

	strelka2			bcftools		
MAPQ	**Precision**	**Sensitivity**	**F-measure**	**Precision**	**Sensitivity**	**F-measure**
x 0.5	0.973	0.360	0.525	0.944	0.646	0.767
x 0.8	0.974	0.530	0.687	0.937	0.785	0.854
x 1.0	0.972	0.573	0.721	0.920	0.792	0.851
x 1.2	0.973	0.588	0.733	0.903	0.795	0.846
x 1.5	0.966	0.730	0.832	0.966	0.731	0.832

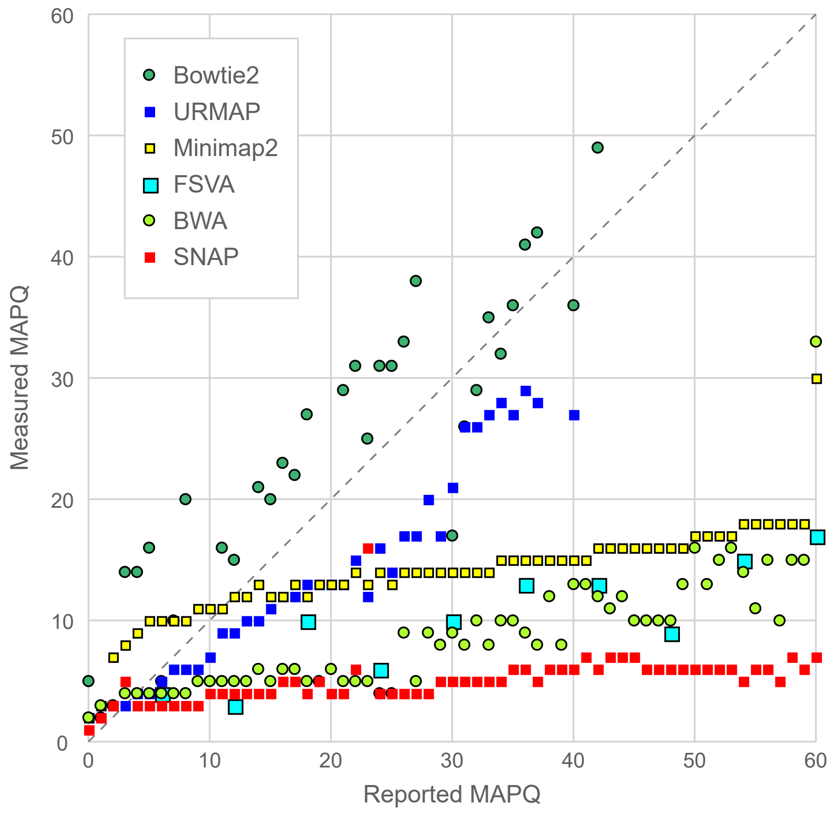


Figure 6 Scatterplot of reported MAPQ vs. measured MAPQ. For each integer value of MAPQ, the measured MAPQ is determined by the frequency of incorrectly mapped reads in the Urbench benchmark. Hisat2 is not shown because it reports only three distinct MAPQ values (see main text).
Full-size DOI: 10.7717/peerj.9338/fig-6

GIAB variant calling test

Results on the variant calling test are shown in Table 1. With strelka2, accuracy of all methods is high (all metrics > 0.97) except for Bowtie2 which has much lower sensitivity (0.59) and F-measure (0.74). Lower accuracies are observed with bcftools, except for Bowtie2 which has higher sensitivity (0.81) and F-measure (0.87). To investigate this, I called variants using modified Bowtie2 SAM files where the MAPQ values were multiplied

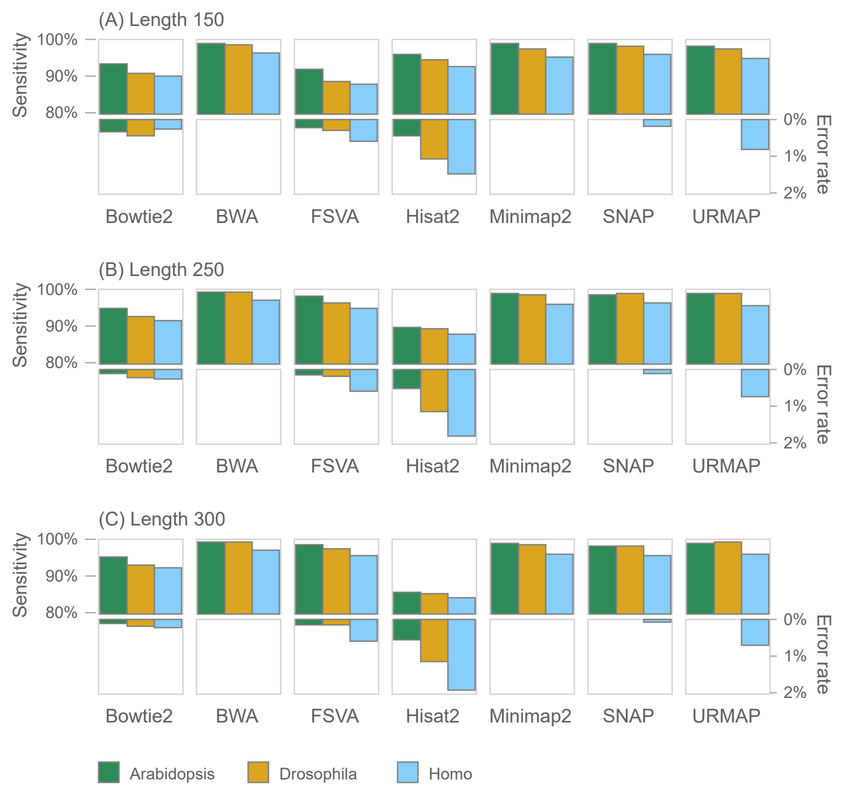


Figure 7 Mapping accuracy on wgsim test. Accuracy metrics are sensitivity and error rate with MAPQ $\geq$10, expressed as percentages. Tests were performed with three simulated read lengths: (A) 150, (B) 250 and (C) 300, respectively.

Full-size DOI: 10.7717/peerj.9338/fig-7

by constant factors 0.5, 0.8, 1.0, 1.2 and 1.5, respectively. To reduce computational cost, I used only SAM records for chr20. Results are shown in Table 2, which show strong variation with the MAPQ scaling factor especially with strelka2. The number of reads mapped by each method on the GIAB test is given in Table S4.

Unmappable regions

I measured unmappable regions with read length $L = 150$ to determine a consensus of BWA, Bowtie2 and URMAP. The MAPQ threshold was set to 3 (error probability $\geq$0.5), which presumably would be considered not reliably mapped in a downstream analysis. As shown in Fig. 8, there is high agreement between the three mappers. A total of 51M bases are not mappable by at least one mapper; 97% of which (49.6M bases, collectively designated UnmapQ3) are not mappable by any of these mappers. With MAPQ $\leq$10, the consensus increases by only a small amount to 50.8M bases and with MAPQ $\leq$1 (error probability $\geq$ 0.8) the consensus decreases marginally to 49.5M. These results show that identification of unmappable regions is robust against choices of mapper and MAPQ threshold. Intersecting UnmapQ3 with UCSC browser tracks gave the following numbers of bases: genomicSuperDup (high-identity long segmental duplications) 20.M, repeatMasker 36.9M, simpleRepeats 25.8M and all three tracks combined 47.7M. Thus,

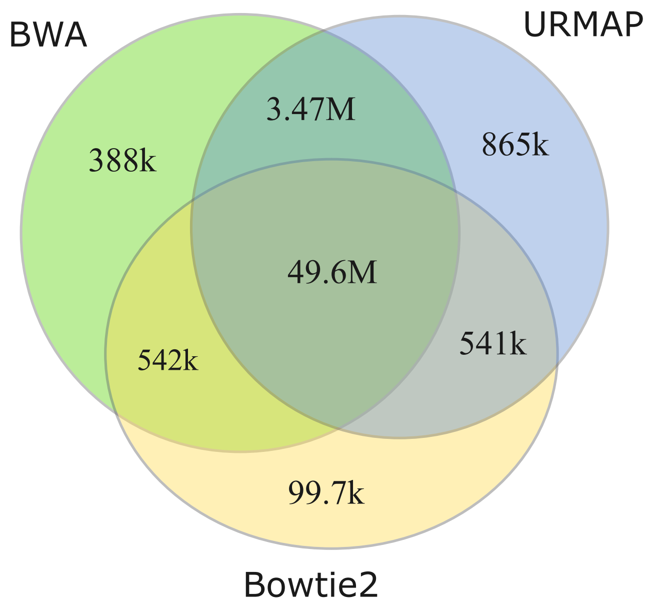


Figure 8 **Mapper agreement on unmappable regions in the human reference genome.** Venn diagram showing agreement of BWA, Bowtie2 and URMAP on unmappable regions with 2 $\times$150 reads of GRCh38 with MAPQ $\leq$3. These mappers agree that 49.6M bases (intersection of the three regions) are not mappable.

Full-size DOI: 10.7717/peerj.9338/fig-8

as might be expected, unmappable regions are mostly comprised of well-known repeats. There are 747 high-confidence GIAB HG002 variants in UnmapQ3, corresponding to a frequency of 1.5×10^{-5} variants per base, or equivalently one variant every 66kb on average when the diploid genome is mapped onto the reference. By contrast, the frequency in the rest of the genome (i.e., regions which are mappable by 2×150 reads) is 0.0015 or one variant every 650 bases.

DISCUSSION

GIAB variant call accuracy as a benchmark of mapping accuracy

Tables 1 and 2 show that the accuracy of variant calls on a GIAB with a single variant caller is not a robust test of mapping accuracy. The accuracy of a variant calling pipeline varies with the choice of caller in addition to the mapper, as might be expected, and also that there are dependencies between the variant caller and mapper such a given pair may have low accuracy together (e.g., Bowtie2 + strelka2) while each has higher accuracy in a different pipeline (e.g., Bowtie2 + bcftools and BWA + strelka2). With both strelka2 and bcftools, accuracy with Bowtie2 is substantially lower than the other tested methods, which is inconsistent with the good performance of Bowtie2 on Urbench and the wgsim test. Bowtie2 differs from all other tested methods by over-estimating rather than underestimating MAPQ (Fig. 6), suggesting that variant callers, especially strelka2, may be sensitive to MAPQ bias.

GIAB bias against regions which are challenging to mappers

The density of reference variants in unmappable regions is 100× lower than the genome average, showing that the GIAB high-confidence variants are strongly biased against repetitive regions which have ambiguous alignments to conventional 2×150 reads. The GIAB high-confidence regions for HG002 cover 2.5 Gb, i.e., exclude 27% of the genome, suggesting that there may be extensive bias against repetitive regions which are mappable by the conservative definition used here but nevertheless challenging for short-read mapping.

Number of aligned reads as a test of mapping accuracy

If mappers are compared by the number of aligner reads, a larger number does not necessarily indicate better performance because there could be more errors. In practice, there is usually little difference between a read which is not mapped and a read which is mapped with very low MAPQ, e.g., MAPQ = 0 corresponds to P_{error}=1 and MAPQ = 1 to P_{error} = 0.8, so alignments with MAPQ ≤1 should be ignored by most downstream analysis. Alignments with higher MAPQ are more relevant, but mappers have widely varying MAPQ biases (Fig. 6) such that MAPQ values and thresholds are not directly comparable between different mappers.

CONCLUSIONS

URMAP is an order of magnitude faster than BWA while achieving comparable accuracy. A speed-optimized variant of the algorithm is >25× faster than BWA with accuracy that is slightly lower but nevertheless likely to be useful in applications where the best possible accuracy is not required and/or computational cost is a limiting factor. On a GIAB variant calling test with 30× coverage 2×150 reads, URMAP achieves high accuracy (precision 0.998, sensitivity 0.982 and F-measure 0.990) with the strelka2 caller. However, GIAB reference variants are shown to be biased against repetitive regions which are difficult to map and may therefore pose an unrealistically easy challenge to read mappers and variant callers.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

The author received no funding for this work.

Competing Interests

The author declares that he receives income from the sale of scientific software through his personal web site at https://drive5.com.

Author Contributions

- Robert Edgar conceived and designed the experiments, performed the experiments, analyzed the data, prepared figures and/or tables, authored or reviewed drafts of the paper, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

Source code is available at https://github.com/rcedgar/urmap/.

Benchmark data are available at OSF: Edgar, R. (2020). ”Urbench”. OSF. Dataset. https://osf.io/th4qv/.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj.9338#supplemental-information.

REFERENCES

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. 1990. Basic local alignment search tool. *Journal of Molecular Biology* **215**:403–410 DOI 10.1016/S0022-2836(05)80360-2.

Altshuler DL, Durbin RM, Abecasis GR, Bentley DR, Chakravarti A, Clark AG, Collins FS, De La Vega FM, Donnelly P, Egholm M, Flicek P, Gabriel SB, Gibbs RA, Knoppers BM, Lander ES, Lehrach H, Mardis ER, McVean GA, Nickerson DA, Peltonen L, Schafer AJ, Sherry ST, Wang J, Wilson RK, Deiros D, Metzker M, Muzny D, Reid J, Wheeler D, Wang SJ, Li J, Jian M, Li G, Li R, Liang H, Tian G, Wang B, Wang J, Wang W, Yang H, Zhang X, Zheng H, Ambrogio L, Bloom T, Cibulskis K, Fennell TJ, Jaffe DB, Shefler E, Sougnez CL, Bentley IDR, Gormley N, Humphray S, Kingsbury Z, Koko-Gonzales P, Stone J, McKernan KJ, Costa GL, Ichikawa JK, Lee CC, Sudbrak R, Borodina TA, Dahl A, Davydov AN, Marquardt P, Mertes F, Nietfeld W, Rosenstiel P, Schreiber S, Soldatov AV, Timmermann B, Tolzmann M, Affourtit J, Ashworth D, Attiya S, Bachorski M, Buglione E, Burke A, Caprio A, Celone C, Clark S, Conners D, Desany B, Gu L, Guccione L, Kao K, Kebbel A, Knowlton J, Labrecque M, McDade L, Mealmaker C, Minderman M, Nawrocki A, Niazi F, Pareja K, Ramenani R, Riches D, Song W, Turcotte C, Wang S, Dooling D, Fulton L, Fulton R, Weinstock G, Burton J, Carter DM, Churcher C, Coffey A, Cox A, Palotie A, Quail M, Skelly T, Stalker J, Swerdlow HP, Turner D, De Witte A, Giles S, Bainbridge M, Challis D, Sabo A, Yu F, Yu J, Fang X, Guo X, Li Y, Luo R, Tai S, Wu H, Zheng H, Zheng X, Zhou Y, Marth GT, Garrison EP, Huang W, Indap A, Kural D, Lee WP, Leong WF, Quinlan AR, Stewart C, Stromberg MP, Ward AN, Wu J, Lee C, Mills RE, Shi X, Daly MJ, DePristo MA, Ball AD, Banks E, Browning BL, Garimella KV, Grossman SR, Handsaker RE, Hanna M, Hartl C, Kernytsky AM, Korn JM, Li H, Maguire JR, McKenna A, Nemesh JC, Philippakis AA, Poplin RE, Price A, Rivas MA, Sabeti PC, Schaffner SF, Shlyakhter IA, Cooper DN, Ball EV, Mort M, Phillips AD, Stenson PD, Sebat J, Makarov V, Ye K, Yoon SC, Bustamante CD, Boyko A, Degenhardt J, Gravel S, Gutenkunst RN, Kaganovich M, Keinan A, Lacroute P, Ma X, Reynolds A, Clarke L, Cunningham F, Herrero J, Keenen S, Kulesha E, Leinonen R, McLaren WM, Radhakrishnan R, Smith RE, Zalunin V, Korbel JO, Stütz AM, Humphray IS , et al. 2010. A map of human genome variation from population-scale sequencing. *Nature* **467**:1061–1073 DOI 10.1038/nature09534.

Benson DA, Cavanaugh M, Clark K, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW. 2013. GenBank. *Nucleic Acids Research* **41**.

Burrows M, Wheeler D. 1994. A block-sorting lossless data compression algorithm. Technical report 124, Palo Alto, CA, Digital Equipment Corporation.

Church DM, Schneider VA, Graves T, Auger K, Cunningham F, Bouk N, Chen HC, Agarwala R, McLaren WM, Ritchie GRS, Albracht D, Kremitzki M, Rock S, Kotkiewicz H, Kremitzki C, Wollam A, Trani L, Fulton L, Fulton R, Matthews L, Whitehead S, Chow W, Torrance J, Dunn M, Harden G, Threadgold G, Wood J, Collins J, Heath P, Griffiths G, Pelan S, Grafham D, Eichler EE, Weinstock G, Mardis ER, Wilson RK, Howe K, Flicek P, Hubbard T. 2011. Modernizing reference genome assemblies. *PLOS Biology* **9**:e1001091 DOI 10.1371/journal.pbio.1001091.

Cleary JG, Braithwaite R, Gaastra K, Hilbush BS, Inglis S, Irvine SA, Jackson A, Littin R, Rathod M, Ware D, Zook JM, Trigg L, DeLaVega FMM. 2015. Comparing variant call files for performance benchmarking of next-generation sequencing variant calling pipelines.

Eberle MA, Fritzilas E, Krusche P, Källberg M, Moore BL, Bekritsky MA, Iqbal Z, Chuang HY, Humphray SJ, Halpern AL, Kruglyak S, Margulies EH, McVean G, Bentley DR. 2017. A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *Genome Research* **27**:157–164 DOI 10.1101/gr.210500.116.

Gilbert JA, Dupont CL. 2011. Microbial metagenomics: beyond the genome. *Annual Review of Marine Science* **3**:347–371 DOI 10.1146/annurev-marine-120709-142811.

Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler a. D. 2002. The human genome browser at UCSC. *Genome Research* **12**:996–1006 DOI 10.1101/gr.229102.

Kim D, Paggi JM, Park C, Bennett C, Salzberg SL. 2019. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature Biotechnology* **37**:907–915 DOI 10.1038/s41587-019-0201-4.

Kim S, Scheffler K, Halpern AL, Bekritsky MA, Noh E, Källberg M, Chen X, Kim Y, Beyter D, Krusche P, Saunders CT. 2018. Strelka2: fast and accurate calling of germline and somatic variants. *Nature Methods* **15**:591–594 DOI 10.1038/s41592-018-0051-x.

Krusche P, Trigg L, Boutros PC, Mason CE, De La Vega FM, Moore BL, Gonzalez-Porta M, Eberle MA, Tezak Z, Lababidi S, Truty R, Asimenos G, Funke B, Fleharty M, Chapman BA, Salit M, Zook JM. 2019. Best practices for benchmarking germline small-variant calls in human genomes. *Nature Biotechnology* **37**:555–560 DOI 10.1038/s41587-019-0054-x.

Langmead B, Salzberg SL. 2012. Fast gapped-read alignment with Bowtie 2. *Nature Methods* **9**:357–359 DOI 10.1038/nmeth.1923.

Langmead B, Trapnell C, Pop M, Salzberg SL. 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* **10**:357–359.

Leinonen R, Sugawara H, Shumway M. 2011. The sequence read archive. *Nucleic Acids Research* **39(suppl_1)**:D19–D21.

Li H. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**:3094–3100 DOI 10.1093/bioinformatics/bty191.

Li H, Durbin R. 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* **25**:1754–1760 DOI 10.1093/bioinformatics/btp324.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. 2009a. The sequence alignment/map format and SAMtools. *Bioinformatics* **25**:2078–2079 DOI 10.1093/bioinformatics/btp352.

Li R, Li Y, Kristiansen K, Wang J. 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics* **24**:713–714 DOI 10.1093/bioinformatics/btn025.

Li R, Yu C, Li Y, Lam TW, Yiu SM, Kristiansen K, Wang J. 2009b. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics* **25**:1966–1967 DOI 10.1093/bioinformatics/btp336.

Liu S, Wang Y, Wang F. 2016. A fast read alignment method based on seed-and-vote for next generation sequencing. *BMC Bioinformatics* **17**:Article 466 DOI 10.1186/s12859-016-1329-6.

Minoche AE, Dohm JC, Himmelbauer H. 2011. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and genome analyzer systems. *Genome Biology* **12**:Article R112 DOI 10.1186/gb-2011-12-11-r112.

Montgomery SB, Goode DL, Kvikstad E, Albers CA, Zhang ZD, Mu XJ, Ananda G, Howie B, Karczewski KJ, Smith KS, Anaya V, Richardson R, Davis J, MacArthur DG, Sidow A, Duret L, Gerstein M, Makova KD, Marchini J, McVean G, Lunter G. 2013. The origin, evolution, and functional impact of short insertion-deletion variants identified in 179 human genomes. *Genome Research* **23**:749–761 DOI 10.1101/gr.148718.112.

Morozova O, Marra MA. 2008. Applications of next-generation sequencing technologies in functional genomics. *Genomics* **92**:255–264 DOI 10.1016/j.ygeno.2008.07.001.

Ning Z, Cox AJ, Mullikin JC. 2001. SSAHA: a fast search method for large DNA databases. *Genome Research* **11**:1725–1729 DOI 10.1101/gr.194201.

Quinlan AR, Hall IM. 2010. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* **26**:841–842 DOI 10.1093/bioinformatics/btq033.

Schirmer M, Ijaz UZ, D'Amore R, Hall N, Sloan WT, Quince C. 2015. Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform. *Nucleic Acids Research* **43(6)**:e37-e37.

Snyder MW, Kircher M, Hill AJ, Daza RM, Shendure J. 2016. Cell-free DNA comprises an in vivo nucleosome footprint that informs its tissues-of-origin. *Cell* **164**:57–68 DOI 10.1016/j.cell.2015.11.050.

Viterbi AJ. 2006. A personal history of the Viterbi algorithm. *IEEE Signal Processing Magazine* **23(4)**:120–142.

Zaharia M, Bolosky WJ, Curtis K, Fox A, Patterson D, Shenker S, Stoica I, Karp RM, Sittler T. 2011. Faster and more accurate sequence alignment with SNAP. ArXiv preprint. arXiv:1111.5572.

Zhang L, Zhou X, Weng Z, Sidow A. 2019. Assessment of human diploid genome assembly with 10x Linked-Reads data. *Gigascience* **8**:Article giz141 DOI 10.1093/gigascience/giz141.

Zook JM, Chapman B, Wang J, Mittelman D, Hofmann O, Hide W, Salit M. 2014. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nature Biotechnology* **32**:246–251 DOI 10.1038/nbt.2835.

Zook JM, Catoe D, McDaniel J, Vang L, Spies N, Sidow A, Weng Z, Liu Y, Mason CE, Alexander N, Henaff E, McIntyre ABR, Chandramohan D, Chen F, Jaeger E, Moshrefi A, Pham K, Stedman W, Liang T, Saghbini M, Dzakula Z, Hastie A, Cao H, Deikus G, Schadt E, Sebra R, Bashir A, Truty RM, Chang CC, Gulbahce N, Zhao K, Ghosh S, Hyland F, Fu Y, Chaisson M, Xiao C, Trow J, Sherry ST, Zaranek AW, Ball M, Bobe J, Estep P, Church GM, Marks P, Kyriazopoulou-Panagiotopoulou S, Zheng GXY, Schnall-Levin M, Ordonez HS, Mudivarti PA, Giorda K, Sheng Y, Rypdal KB, Salit M. 2016. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data* **3**:160025 DOI 10.1038/sdata.2016.25.