Method

The SeqSaw method we developed involves four main steps. First, SeqSaw splits sequencing reads into shorter segments with overlaps (step 1 in Fig. S1). For example, a 75 nt read can be split into two overlapping 50 nt segments (1–50 and 26–75). Next, all short segments are mapped to the reference genome, allowing for gapped alignments (step 2 in Fig. S1). Then, the mapping results of the short segments from the same read are assembled to generate valid alignments for full-length reads (step 3 in Fig. S1). Finally, on the basis of the mapping results, SeqSaw predicts junctions according to a series of optional filters (step 4 in Fig. S1).

In step 2, SeqSaw first employs Bowtie [1] to accelerate the process. Bowtie can efficiently map segments to the reference genome without allowing gaps. If a segment can be mapped to a continuous region by Bowtie, SeqSaw will use the mapping location directly in step 3. For segments that cannot be mapped by Bowtie, we use SeqSaw to map them again to the genome by allowing gaps.

Fig. S1. Four main steps of SeqSaw.

1. Mapping short segments allowing for gapped alignments
Fig. S2. The mapping process for the hypothetical example.

The core algorithm of SeqSaw consists of mapping short segments to the reference allowing gaps. A hypothetical example is presented to illustrate the method in Fig. S2.
Animated slides for this example can be downloaded from our website [2] or from SourceForge [3]. Let us take three nine nt “reads” and a 37 nt “reference” as the example (see Fig. S2a). In this mini example, we wanted to align the reads to the reference allowing a gap and no mismatches (see section 1.6 for allowing mismatches). The intron was limited to 20 bp, so the possible hits with a gap longer than 20 bp would be discarded. In this example, the reads were also required to cover at least two bp on each side of the gap.

Two hash tables were built to index the reads. One was static when the sliding window scanned along the reference genome, and the other one was dynamic in the whole process. We call the static table the “head-key hash table” (blue box in Fig. S2a) and call the dynamic one the “tail-key hash table” (yellow box in Fig. S2a).

1.1. Building the head-key hash table

For the two halves of a splice junction read of \( n \) nt long we know that the longer half is at least \( n/2 \) nt long. At first, SeqSaw will try to find all the possible locations for the reads with the 5’ half containing at least \( n/2 \) nt. The locations of reads with the 3’ half over \( n/2 \) nt long will be detected by a reverse search with a symmetrical method. An example for the reverse searching will be given after the whole process is described.

The package first indexes all the reads using the first \( n/2 \) nt (5 nt in the example) of the reads as keys. A total of \( N (N: \text{the count of the reads}) \) key-value pairs (keys and read ids) will be added into the head-key hash table. The head-key hash table will not be changed until the sliding window arrives at the end of the reference genome.

In our hypothetical example (Fig. S2b), three items are added into the head-key hash table. The key-value pairs are AGCTA:read0, AGCTA:read1, and GCATG:read2.

1.2. Head-key hash table lookup and filling the tail-key hash table

After building the head-key hash table, two windows (blue window and red window in Fig. S2c) then slide along the reference base by base. The sequence in the blue window (\( n/2 \) nt long) will be used as a key to access the head-key hash table. The sequence in the red window (suppose \( m \) nt long; two nt in the example) will be used as a key to access the tail-key hash table. The two windows slide base by base simultaneously (arrow ① in Fig. S2c). Once the windows slide one base, SeqSaw first accesses the head-key table by the
sequence in the first window (arrow ② in Fig. S2c), and then accesses the tail-key hash table by the sequence in the second window (arrow ③ in Fig. S2c).

Once a read is retrieved from the head-key hash table, SeqSaw will extend the hit base by base and check how many base pairs can be aligned to the reference from the start position. The mapping information for this read is then added into the tail-key hash table with the last $m$ bases of the read as the “tail” key.

For the case in Fig. S2d, two reads (read0 and read1; blue items in the blue box) are retrieved from the head-key hash table by the key AGCTA (arrow ① in Fig. S2d). Then the two hits are extended base by base along the reference (arrow ② in Fig. S2d). For read0, five bases can be aligned to the reference from the start position, and the residual sequence TGGA is unaligned (gray letters in Fig. S2d). For read1, six bases can be aligned, and the residual sequence is CCT. The mapping information for read0 is then added into the dynamic tail-key hash table with the last two bases GA as the key, and so becomes the mapping information for read1 with CT as the key (arrow ③ in Fig. S2d). Similarly (Fig. S2e), when the windows slide further along the reference, read2 is retrieved from the static head-key hash table, and the matching information is added into the dynamic hash table with the last two bases GA as the key.

1.3. Tail-key hash table look up and generating mapping results

While sliding the windows along the reference, SeqSaw also accesses the dynamic tail-key hash table by the sequence in the second window.

Once a read is retrieved from the tail-key hash table, SeqSaw will check if the residual of the read (the bases which were not aligned to the reference at the first extending process) can be aligned to the reference at the current position. If true, a possible hit for the read will be reported.

For the case in Fig. S2f, where the second window arrives at the 21st base of the reference, read1 with the initial mapping information (red item in the yellow box) is retrieved from the tail-key hash table (arrow ① in Fig. S2f). SeqSaw then checks whether the residuals can be aligned to the reference at this position (arrow ② in Fig. S2f). For this example, the three net residuals (CCT) can be mapped to the reference at position “[18, 21)" (i.e. from the 18th to
the 20th base on the reference). In the first extending process, the first six bases (AGCTAG) of read1 have already been mapped to the position [1, 7) (Fig. S2d), which is known by the information stored in the tail-key hash table. Therefore, a possible hit is found for read1 at [1, 7) + [18, 21) (see Fig. S2g) with the mapping pattern “6+3” (the first half is six nt and the other half is three nt). For the case in Fig. S2h, when the second window slides to the 26th base of the reference, read0 is retrieved from the tail-key hash table and a possible hit at [1,6) + [22,26) is revealed.

Fig. S2i illustrates an ambiguous case that differs from the previous two cases. The nucleotide G (highlighted in red on read2) can be aligned to two possible positions on the reference (highlighted in red on the reference). It can be assigned either as the last base for the first half or as the first base for the second half. The possible hits are [9,16) + [24,26) and [9,15) + [23,26), with the pattern 6+3 or 7+2. In fact, ambiguous cases can also be introduced by allowing mismatches. We cannot decide the exact residual sequence at the first extending step (for the example in Fig. S2e, we cannot decide if the exact residual sequence should be GGA or GA), so a greedy method is used at the first extending step.

1.4. Deleting the out-of-date items from the tail-key hash table

For mammalian genes, there are few known junctions separated by introns more than 400 kb long. SeqSaw therefore restricts the distance between the two halves of a read to a certain value \( L \) (where \( L \) is set to 400 kb by default). For each item in the tail-key hash table, once the windows slide to a position more than \( L \) bp away from the position where the item is generated, this item will be out-of-date.

For the case in Fig. S2j, we suppose that \( L=20 \). Read0 is retrieved from the tail-key hash table and a possible hit is found by the second extending process, but the distance between the hits of the two halves is 26 (see arrow ③ in Fig. S2j). This value exceeds the distance limitation of 20 bp, so the result is filtered out. In fact, the first item in the tail-key hash table is out-of-date from now on.

In order to handle the tail-key hash table efficiently, the out-of-date items should be deleted (see Fig. S2k). A possible method is to check the tail-key hash table once the windows slide to a new position (arrow ① in Fig. S2k), and delete all of the out-of-date items.
Fig. S2k shows the process of checking and deleting dated items from the tail-key hash table. There is only one item retained after checking. However, this checking method is time consuming when checking hundreds of thousands of items at each step, so instead of checking the table frequently, SeqSaw employs a ring queue and a bi-directional list to handle the tail-key hash table more efficiently. This ring queue is long enough to guarantee that the item to be reused is out-of-date, and all the items in the ring queue can be recycled one after another.

1.5. Reverse searching for hits with symmetrical patterns

For the given example, the above steps comprise the entire process for detecting the hits of the patterns 5+4, 6+3, 7+2, and 9+0 (the pattern 9+0 can be treated as the pattern 7+2 with the length of the gap equal to 0). The possible hits with patterns 4+5, 3+6, and 2+7 will be found by a reverse search using a symmetrical method (see Fig. S2l for example). It should be pointed out that only the length of the tail-key decides the minimum read cover length on each side of the splice junction (two nt in this example).

1.6. Allowing mismatches on the keys and processing long reads spanning more than one intron

For detection of junctions covering SNPs and accommodating sequencing errors, SeqSaw also allows mismatches on the two kinds of keys. SeqSaw uses the conventional space seeds indexing method to build and access the hash table. This method is also widely used by SeqMap [4], Eland (Cox, 2006 unpublished package), and other software.

For the current version, SeqSaw allows up to 4 mismatches on the whole read and 2 mismatches on each key. The length of the reads supported range from 50 to 200, which covers the span of typical Illumina sequencing data. The core algorithm of SeqSaw only tolerates one gap on each read. To handle longer reads that span two or more introns, SeqSaw first splits these reads into several 50-nt sequences with overlap and then performs SeqSaw mapping.

2. Dealing with paired-end reads

Paired-end reads can contribute to the achievement of higher specificity of the reported junctions. When SeqSaw deals with paired-end reads, the read directionality and the order of
the two reads in a pair must be consistent with the experimental protocol. In addition, the distance between hit locations of the two reads is not allowed to be too long, and is limited to a certain threshold (set to 400 kb by default). Alignments that do not satisfy these requirements are flagged to indicate failure of paired-end analysis.

3. Post-processing for junction prediction

SeqSaw is implemented to map exon and junction reads to the reference genome. Both exonic and spliced alignments for all reads can be reported. By aggregating spliced alignments for all reads, putative splice junctions can be predicted. Generally, there is a tradeoff between the total number of reported junctions and detection accuracy. Au et al. [5] proposed several statistical criteria to filter the initial aggregates of spliced reads. These optional filters were also employed by SeqSaw, with some necessary modifications. A detailed description of the optional filters can be found in the online Supplement [6].

4. Sliding alignments

![Sliding alignments reported by SeqSaw.](image)

While processing real RNA-seq data, we observed some spliced reads are ambiguously aligned if taking into account non-canonical junctions. An example is shown in Fig. S3, where the nucleotides CTCAG can be assigned to either part of the spliced read when mapping it to the reference genome. The three mapping results shown in Fig. S3 are all acceptable. SeqSaw only reports one of these alignments (suppose the second alignment) and states how many nucleotides can be slide to left (2 for this example) and right (3 for this example).

References


