OMICSOFT ALIGNER INTRODUCTION

OSA is Omicsoft's read aligner designed for high performance alignment of short or long reads to reference libraries (genomes or transcriptomes).

Key features of OSA:

1. High performance for both single end alignment and paired end alignment. Works on reads with 17-65536 nucleotides.

2. For Illumina reads (17nt or longer), allow any number of mismatches and a single indel in the reads. Can identify and report all available alignments in the genome with specified mismatches/indel.

3. For long reads (e.g. 454 reads or PacificBio reads), allow any number of mismatches and indels in the reads. Use banded Smith-Waterman to find the optimal alignment.

4. Directly align RNA-Seq reads to genome by considering both known and novel exon junctions

5. Directly align fusion reads: reads spliced by two different genes or genomic regions

6. Automatic quality based trimming

7. Automatic adapter trimming

8. Support all common input formats (.fasta; .fastq; .qseq; .sff); directly support .gzip files allowing savings in file space

9. Support multiple threads

INDEXING OF REFERENCE

Before the alignment is performed, the reference library needs to be indexed. This index file(s) will be loaded into memory (either loaded as a whole block or loaded as batches) before the alignment of the first read.

There are two types of indexes: 64-bit index and 32-bit index. 64-bit index stores the positions of 14-mers and 32-bit index stores the positions of 12-mers. Only one third of the genome is indexed to reduce the index size. An index is comprised of three parts:

1. A k-mer hash table. For 64-bit index, k=14. For 32-bit index, k=12. This table has a fixed size of 4^{k+1} bytes

2. A sequence position table of size 4N/3 bytes where N is the total nucleotides of the reference library

3. A compressed sequence of size 3N/8 bytes. Each nucleotide is stored as 3 bits.

For the human genome, the 64-bit index takes about 5.8G memory and this is the memory requirement for all alignment modules. 32-bit index file size is about 5.2G, but the memory usage is fixed at <1G memory because the index is broken into small pieces and loaded into memory sequentially. Another major difference of 64-bit and 32-bit index is that the offset table for 32-bit index is significantly smaller (64M in 32-bit vs. 1024M in 64-bit).

Index building is very fast. Both 64-bit or 32-bit index building for human genome take a few minutes to run. Dynamic index building may occur in some advanced mapping (e.g. assembly novel exon junctions) and will be automatically built by the aligner. Omicsoft provides common reference library for automatic downloading, and the first time the user uses the alignment modules, the index will be built into local cache.

BASIC ALIGNMENT

For each read, OSA first scans the read quality from the right end and trims the low quality base from the right end until the first high quality base is found. Trimming significantly improves the alignment.

The genomic coordinates of each read will be determined from the union of all the position lists for each k-mer (k=14 for 64-bit index and k=12 for 32-bit index). For human size genome, there are ~4 positions for each k-mer. Therefore, a 100-nt read will contain ~400 positions on average. Testing through each position for the numbers of mismatches will give the full list of mapping positions for the read given a predefined number of mismatches (testing simply means comparing the read sequence with the reference sequence). Since most of the alignments have a small number of mismatches (e.g. <=2), various heuristic searches are implemented in OSA to avoid searching through the full list. For example, a perfectly matched read must have the true genomic position contained in at least one of the consistent set of k-mers, where the consistent set has a step of 3 nt and has a shift of 0, 1 or 2 nt from the start read position in either the forward or reverse complement direction. Based on this rule, we can simply use the intersection (instead of union) of all the position lists (in practice, we may use the intersection of the two smallest position lists as the starting point) for verification. For a 100-nt perfectly matched read, the average # of positions that needed to be tested against the reference is decreased to ~1.5 based on a simulation study. This will significantly improve the alignment performance because we observe at least 40% reads to be perfectly matched in real datasets. Alignment of reads with a few mismatches is optimized using extended rules.

Another optimization that we have implemented is the comparison of read sequence and reference sequence. Each 64-nucleotide block of the reference sequence (or read sequence) is stored as three 64-bit words: one for A/G status, one for C/T status, and one for N status (1 = ambiguous nucleotide). We store each nucleotide in 3 bits - not 3 bits in the same byte, but 3 bits in different 64-bit words. The read and reference can be combined bitwise using an *exclusive-or* function for A/G word and C/T word, then by taking an *or* function for the combined result s and an *or* function with the N status word. We have found that this bitwise comparison is 10x faster than naive comparison where nucleotides are coded as single bytes.

Multiple positions might be aligned for a single read with the same number of mismatches. The major source of non-unique mapping we found in practice is from short reads. Longer reads are more likely to be uniquely mapped. Reads from repetitive regions will have multiple positions aligned. Sequencing error or mutation is another possible cause of the non-unique mapping. Paired-end reads can resolve a lot of non-unique mapping by constraining the pair of the reads to concordant positions.

For non-unique mapping we allow users to report all alignments up to a user defined number. If the total alignment number is larger than the user defined number, no alignment will be reported.

By default, OSA determines the maximal penalty of an alignment. The maximal penalty is defined as the maximal number of mismatches allowed plus the gap penalty if an indel is present in the alignment. Usually we set the gap penalty to one or two (default is two). By default, Omicsoft automatically set the maximal penalty for each read to Max (2, (read length - 31) / 15) based on trimmed read length. Below is a table of automatic penalty for reads with 17- 106 nt.

Read length	Penalty
17-75	2
76-90	3
91-106	4

User can override the automatic penalty setting and set the number to any fixed number. Penalty values of 2, 3, 4 are mostly common for reads < 100 nt.

DETECTION OF INDELS

OSA can detect single insertion or deletion in the alignment, with any mismatches defined by the user. We use a single penalty for all sizes of indels (i.e. only open gap penalty, no extension penalty).

There are two types of indels: middle indels and end indels. Middle indels have at least one mappable k-mer for each end, and the position list of each end can be used to determine the indel type and indel size. Middle indels can efficiently detect large indels. End indels depend on candidate positions generated by the long end of the read broken by the indel. For each candidate position, mismatches are first counted. If the mismatch number is sufficiently slow, the candidate position will be tested across the different possible indel sizes. End indels have to be constrained to a small indel size (e.g. <10) to make the algorithm efficient and also to make the distal end uniquely determinable from the long end.

Indel detection is a few times slower than gap-free alignment, because the heuristic searches cannot be used and all candidate positions need to be considered. In practice, if the read can be aligned without gaps, we will not try to detect indels. The only exception will be intron detection which is described in the section below, in which case introns are not penalized.

ALIGN RNA-SEQ READS TO GENOME

OSA can align transcriptional reads that cross two or more exons. The exon junctions might be known or novel. OSA can align 12M 100nt reads to the genome with a standard CPU core in an hour (~10 times faster than the latest version of TopHat). The high performance comes from the two aspects. First, we used a different work flow which will be described below. Second, unlike TopHat, we never cut longer reads into 25nt shorter reads and align those shorter reads separately. A large proportion of 25nt reads cannot be uniquely mapped and this will add to both the computation time and the ambiguity to resolve the real position(s).

First, we describe the work flow of the RNA-Seq read mapping. Then we will describe the exon junction detection in details. The RNA-Seq read mapping consists of five steps:

1. Based on a gene model, generate a transcriptome reference library from the genome. Each isoform will be an independent sequence entry. This reference library will be indexed and reused. If the reference library (transcriptome) is already in the local cache, this step will be skipped. Omicsoft provides Ensembl gene models for automatic downloading, and the user can always build their own gene model from one or more .gtf/.gff files.

2. Align all reads to the transcriptome. This alignment must allow for non-unique mapping because the isoforms share a large proportion of exons/sequences. Remap the transcriptome coordinates to the genomic coordinates and save the alignment results to a binary file. Note most of the non-unique transcriptome coordinates will be converted to a single genome coordinates. Exon junctions are also generated in this step. All the exon junctions in this step are considered as known.

3. Align those unmapped reads to the genome by considering one or more exon junctions. Exon junctions detected in this step are considered as novel. Write all the exon junctions detected in this step to a reference library by extracting/extending the exon sequences from the genome and concatenating the exon sequences into a pseudo transcript sequence.

4. Build the index based on the pseudo transcriptome generated in step 3 from novel exon junctions. Map all unmapped reads (excluding reads mapped in step 2 or step 3) to the pseudo transcriptome. Remap the transcriptome coordinates to the genome coordinates.

5. Combine all mapped reads from step 2, 3 and 4.

Steps 1 and 2 will guarantee that reads coming from known transcripts will be mapped regardless how many exons the read may span. It also is the key to the high performance of our aligner because the transcriptome is usually smaller and the alignment is gap-free.

The exon junction detection is based on an extended middle indel detection algorithm. Only deletion was considered (as introns are deletions from the reference), and only three canonical acceptor-donor di-nucleotide patterns are considered: GT-AG, GC-AG and AT-AC. A long read (e.g. 100 nt read) may cross more than one exon-exon junction and this will need to be considered. First we sort the candidate positions and collect the position lists where the range of contiguous positions are within a user-defined length (default=50000), then we consider each position collection by allowing for multiple deletions from the reference sequence.

The exon junction detection algorithm performed in step 3 will not be able to detect novel exon junctions with a distal short fragment or if the fragment is long enough (>k+2) but contains mismatches. This is why we added step 4 to rescue these reads. After we build the pseudo transcriptome, the alignment is gap-free and mismatches or short distal fragment are no longer barriers for the full alignment. The basic idea behind this step is that for a real novel exon junction with sufficient coverage, we should be able to see at least some exon junction reads covering enough nucleotides from all exons. In practice, we observed that step 5 can rescue ~2% previously unmapped reads.

PAIRED END ALIGNMENT

Paired-end reads need a different strategy from single-end reads. Paired-end reads have at least two advantages over single-end reads: it can help resolving the ambiguity of non-uniquely mapped reads and it can help assembling the transcriptome in downstream analysis. Concordant alignments are searched at first with the best paired alignment defined as the minimal total mismatch# of the paired alignments, and if concordant alignments cannot be found, best alignment(s) for each read will be reported. OSA uses the following steps to perform paired-end alignment:

1. Get the best alignment for each read. All equally good alignments are returned and paired to get the concordant solutions. If concordant solutions are found, steps 2 and beyond are skipped.

2. If read 1 has a uniquely best alignment, try to perform a constrained alignment on read 2, where the candidate positions of read 2 are constrained by the position of read 1 (by default we extend the read 1 position by expected insert size + 6 * standard deviation of insert size). If concordant solutions can be found, return the result directly.

3. If read 2 has a uniquely best alignment, try to perform a constrained alignment on read 1, where the candidate positions of read 1 are constrained by the position of read 2 (by default we extend the read 1 position by expected insert size + 6 * standard deviation of insert size). If concordant solutions can be found, return the result directly.

4. Consider all the possible pairings from suboptimal alignments from read 1 and read 2. Instead of searching through all possible pairs at all different mismatch level and finding the best paired alignment, we consecutively constrain the mismatch# of single alignments and break the searching if a concordant solution is found in early stage. This strategy does not affect the accuracy the algorithm.

5. If concordant solutions cannot be found, best alignment(s) for each read will be reported.

FUSION READ ALIGNMENT

Fusion reads are defined as reads that are spliced by two different genes or genomic regions. The two gene/genomic regions can be on the same chromosome and different chromosomes. Fusion read alignment is extremely difficult, because the possible combinations of fusion are huge and the search space cannot be reduced by distances or sequence patterns which were used in exon junction detection. Moreover, two sources of the fusion read may come from same or different strands of the genome. It also cause difficulty for 32-bit alignment, where only partial genome is loaded into memory but the single read may cover genomic regions from different index batches. The most difficult part of the fusion alignment is the complexity of mixed fusion junction and exon junction where each or both ends of the fusion read may contain one or more exon junctions, if the reads are RNA-Seq reads.

OSA uses the following steps to handle fusion alignment:

1. First align all the reads to the genome. Consider exon junctions if input reads are RNA-Seq reads. Only keep unaligned reads for fusion gene alignment.

2. For the unaligned reads in step 1, evenly cut each read into two pieces. Align each piece to the genome (considering exon junctions for RNA-Seq reads). Keep those reads that have at least one piece aligned. It is obvious that a real fusion read will have at least one half mapped to the reference. Step 1 and step 2 can exclude >95% reads from further consideration.

3. For each read, cut the read into two reads at every "mappable" fusion junction. A "mappable" fusion junction is defined as a position that is at least 25nt (or user defined length) from either end of the read. This step implicitly assumes that the fusion read must be at least 50nt long after trimming. A 75nt read will have 26 possible cutting positions while a 100nt read will have 51 possible cutting positions. After the cutting, OSA will align each pair of reads to the genome using single-end alignment mode.

4. Consider the group of read pairs that originally came from a single read. A true fusion read with at least 25bp from either fusion source will have at least one of the read pairs aligned to the genome, and the total penalty is small. We determine the fusion junction position by choosing the position(s) with the smallest total penalty number of the two reads. Note the two reads may have different strand orientation and this information is important to define the fusion. In this step we also create a pseudo fusion transcript library by extracting/extending both fusion source sequences and concatenating them into a single pseudo transcript/sequence. For each entry in the library, we count the support# based on number of reads supporting the fusion junction.

5. Build the index based on the pseudo fusion library generated in step 4 from detected fusion junctions. Map all unmapped reads (excluding reads mapped in step 1/2/3) to the pseudo fusion library. Reads mapped in this step are considered as additional support for the fusion junction.

LONG READS ALIGNMENT

OSA implemented a high performance long read alignment algorithm specially designed for 454 reads and PacificBio reads. The alignment algorithm works for reads with any length between 17-65536, and is (one of) the fastest long read alignment algorithm in the market (it is at least 3x faster than BWA-SW). For long reads (reads > 100 nt), the alignment is mostly unique and accurate, and we have found the similarity between OSA and BWA-SW is greater than 99.99%. Like BWA-SW, our alignment algorithm does a banded Smith-Waterman alignment on the target region, therefore allowing for any number of mismatches/indels within the bandwidth. The long read alignment algorithm uses the following work flow:

1. If read length is equal to or less than the pre-defined seed length (default = 38), directly use the standard alignment algorithm described in the previous sections. Note the standard algorithm still allows for any number of mismatches and up to one indel. It is not realistic to search for multiple indels for reads <= 38.

2. If read length is great than the seed length, the aligner will try to anchor the read to the genome at first. The anchoring process is based on three seeds: middle seed, starting seed and end seed. Each seed has a fixed length (default=38) and is composed by cutting the read from the middle/start/end position. First, the middle seed read is mapped to the genome using the standard alignment algorithm allowing for one indel and two mismatches. If middle seed can be mapped (uniquely or non-uniquely), we record the genomic positions and skip the mapping of start/end seeds. Otherwise, we try start seed. If start seed can be mapped, we record the genomic positions and skip the mapping of end seed. Otherwise, we try end seed. If end seed can be mapped, we record the genomic positions. Otherwise, we abort the rest of the steps and report that the read is not mappable.

3. For the genomic positions reported in step 2, we extend the alignment by performing a banded Smith-Waterman alignment. The alignment(s) with the best score will be reported. The Smith-Waterman algorithm will search for software clipping (S in CIGAR), insertions (I in CIGAR) and deletions (D in CIGAR) and report the optimal score with the constrain that the total indel size is less than (bandwidth+1)/2 (default bandwidth = 33).

At this stage, we only support sing-end long read alignment.

FUTURE SUPPORT

Currently OSA only supports nucleotide space alignment, although Omicsoft does have the plan to extend the algorithm to color space. Extension of the basic algorithm is straightforward by indexing the color space reference library and mapping the color bases, but the incorporation of indels or introns into the alignment is more difficult.

Automatic adapter stripping is limited to 3'End at this stage. We plan to add more capabilities for automatic adapter stripping, including automatic extracting of multiplex IDs (MIDs).

Our aligner can also be extended to align bisulphite treated reads.