mosaik assembler

MICHAEL STRÖMBERG

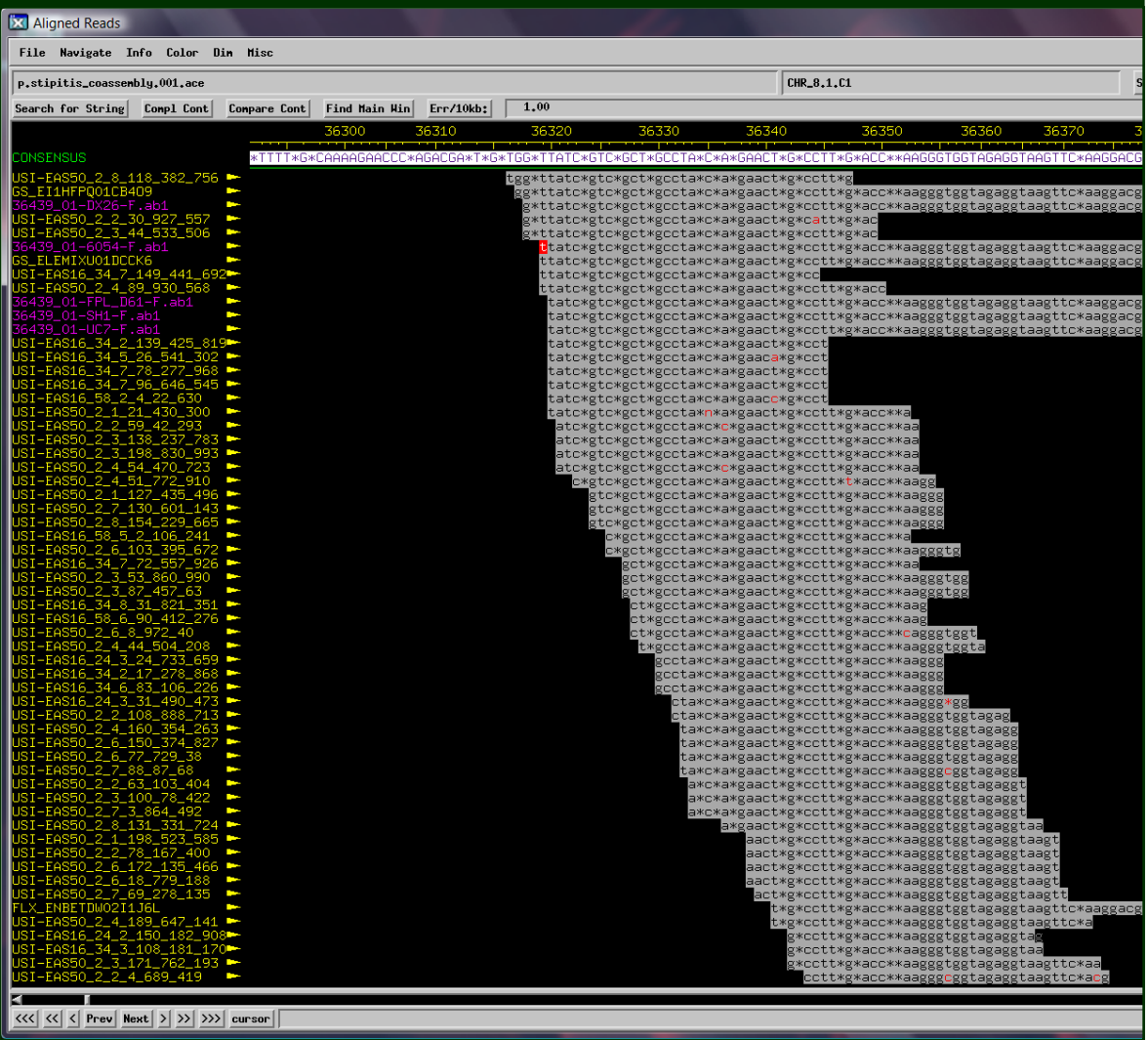


Table of Contents

1. Introduction

MOSAIK is a reference-guided assembler comprising of four main modular programs (Figure 1.1):

- Mosaik**Build**
- Mosaik**Aligner**
- Mosaik**Sort**
- Mosaik**Assembler**.

Mosaik**Build** converts various sequence formats into Mosaik's native read format. Mosaik**Aligner** pairwise aligns each read to a specified series of reference sequences. Mosaik**Sort** resolves paired-end reads and sorts the alignments by the reference sequence coordinates. Finally, Mosaik**Assembler** parses the sorted alignment archive and produces a multiple sequence alignment which is then saved into an assembly file format.

At this time, the workflow consists of supplying sequences in ***FASTA, FASTQ, Illumina Bustard & Gerald, or SRF file formats*** and producing assembly files (***phrap ace and GigaBayes gig formats)*** which can be viewed with utilities such as consed or EagleView.

1.1. What makes MOSAIK different?

Unlike many current read aligners, MOSAIK produces gapped alignments using the Smith-Waterman algorithm. Additionally, our program goes beyond producing pairwise alignments and produces reference-guided assemblies with gapped alignments. These features make it ideal for downstream single nucleotide polymorphism (SNP) and short insertion/deletion (INDEL) discovery.

MOSAIK is written in highly portable C++ and currently targetted for the following platforms: Microsoft Windows, Apple Mac OS X, Linux/x86, Linux/Itanium2, and Sun Solaris/UltraSPARC operating systems. Other platforms can easily be supported upon request.

MOSAIK is multithreaded. If you have a machine with 8 processors, you can use all 8 processors to align reads faster while using the same memory footprint as when using one processor.

MOSAIK supports multiple sequencing technologies. In addition to legacy technologies such as Sanger capillary sequencing, our program supports next generation technologies such as Roche 454, Illumina, AB SOLiD, and experimental support for the Helicos Heliscope.

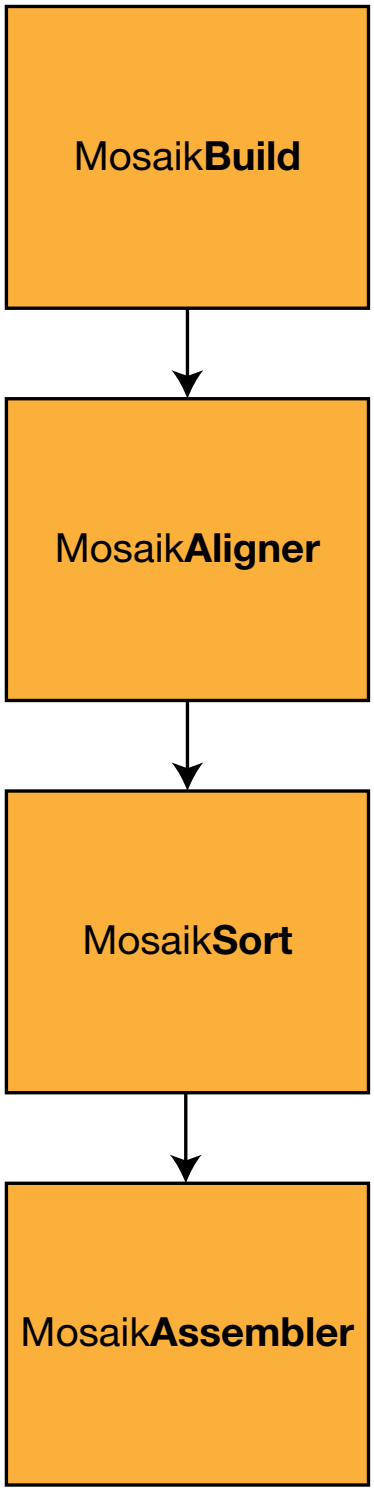


Figure 1.1 **MOSAIK Pipeline.**

This figure illustrates the basic MOSAIK pipeline. Mosaik**Build** converts external read formats, Mosaik**Aligner** pairwise aligns the reads, Mosaik**Sort** sorts the alignments and resolves paired-end reads, and finally Mosaik**Assembler** creates a gapped assembly file.

1.2. Overview

The primary MOSAIK programs (MosaikBuild, MosaikAligner, MosaikSort, MosaikMerge, and MosaikAssembler) are discussed individually in Chapter 2.

The MOSAIK utility programs (MosaikText, MosaikCoverage, and MosaikJump) are discussed individually in Chapter 3.

Chapter 4 discusses how to fine tune performance for a variety of reference genomes and sequencing technologies.

For the impatient, a one page crib sheet with the most important parameters to the primary MOSAIK programs is provided in the appendix.

1.3. Contact Information

Feel free to contact me to suggest improvements, submit bug reports, or just to offer some moral support:

Michael Strömberg
Biology Department
Boston College - Higgins Hall
140 Commonwealth Ave
Chestnut Hill, MA 02467
USA

email: mikaels@bc.edu

2. The MOSAIK suite

2.1. MosaikBuild

To speed up the assembly pipeline, compressed binary file formats are used extensively throughout MOSAIK. Mosaik**Build** translates external read formats to a format that the aligner can readily use. In addition to processing reads, the program also converts reference sequences from a FASTA file to an efficient binary format.

Mosaik**Build** readily converts FASTA, FASTQ, Illumina Bustard, Illumina Gerald, and SRF files. Pyrosequences in SFF files can be converted to the FASTA format with the PyroBayes utility (*http://bioinformatics.bc.edu/marthlab/PyroBayes*). With files containing both bases and base qualities, such as FASTQ and SRF, MosaikBuild can convert an entire directory of read files into a single MOSAIK read file. This is handy, for example, when converting a run of Illumina paired-end reads that are separated by lanes and mate-pairs.

Mosaik**Build** automatically handles FASTA and FASTQ in both the uncompressed or compressed (gzipped) state. There's no need to uncompress the files prior to importing them into MOSAIK.

Mosaik**Build** stores the specified sequencing technology (**-st** parameter) and read type (single ended vs paired-end) so that it can be tracked throughout the entire pipeline.

Using MosaikBuild (reference sequences)

```
MosaikBuild -fr c_elegans_chr2_ref.fa.gz -oa
c_elegans_chr2_ref.dat
```

Here we convert the reference sequence for chromosome 2 in C. elegans from a FASTA file to a native MOSAIK reference archive.

In all of the MOSAIK programs, parameters can be placed in any order.

Using MosaikBuild (reads)

```
MosaikBuild -q c_elegans_chr2_mate1.fq.gz -q2
c_elegans_chr2_mate1.fq.gz -out c_elegans_chr2.dat
-st illumina -p B_
```

Here we convert the paired-end Illumina reads from a pair of FASTQ files to a native MOSAIK read archive. In this example we prepend each read name with "B_" so that we know that these reads are from the Bristol strain during visualization and SNP discovery.

Box 2.1 **MosaikBuild Parameters.**

FASTA

-fr, -fr2	specifies the files containing the bases for the first mate (-fr) and the second mate (-fr2).
-fq, -fq2	specifies the files containing the base qualities for the first mate (-fq) and the second mate (-fq2).
-assignQual	assigns a base quality for every base.

FASTQ

-q, -q2	specifies the files for the first mate (-q) and the second mate (-q2).

SRF

-srf	specifies the short read format file (currently only single-ended reads are supported).

Illumina Bustard

-bd	specifies the Illumina Bustard directory.
-il	specifies which lanes to use. e.g. -il 137 will select lanes 1, 3, and 7.
-split	split the reads into two equal portions for paired-end reads.

Illumina Gerald

-gd	specifies the Illumina Gerald directory.
-il	specifies which lanes to use. e.g. -il 137 will select lanes 1, 3, and 7.

Shared Options

-cs	translates the reference sequence from base space to colorspace.
-p	adds the specified prefix to each read name.
-rl	limits the number of reads parsed.
-tp, -ts	trims the first (-tp) or last (-ts) bases by the specified amount.
-tpr, -tsr	trims the first (-tpr) or last (-tsr) characters from the read name by the specified amount.
-tn	sets the maximum tolerance of internal N's. By default reads with more than 4 internal N's are discarded.
-st	specifies the sequencing technology: 454, helicos, illumina, sanger, solid
-out	specifies the output file when converting reads.
-oa	specifies the output file when converting the reference sequence.

2.2. MosaikAligner

MosaikAligner performs pairwise alignment between every read in the read archive and a set of reference sequences (Figure 2.1). The program uses a hashing scheme similar to BLAT and BLAST and places all of the hashes (k-words or seeds) into a hash map or into a "jump database".

When presented with a new read, MosaikAligner hashes up the read in a similar fashion and retrieves the reference positions for each hash in the hash table. These hash positions are clustered together and then evaluated with a full Smith-Waterman algorithm. Alignments are then screened according to the filter criteria specified by the user.

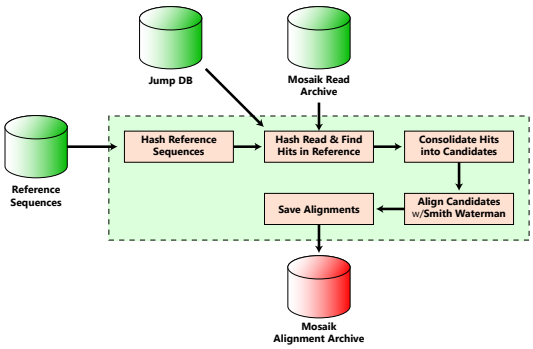


Figure 2.1 **MosaikAligner Illustrated.**

Hashing Strategy

The first incarnation of MOSAIK used a fast hashing strategy that stored one hash position per seed. If a seed was seen more than once in the reference sequence, it was marked non-unique and removed from subsequent use. When clustering hash positions, the longest contiguous set of hash was used to seed the full Smith-Waterman alignment. While speedy, this strategy (**FAST**) was vulnerable to microrepeat structures in reference sequences and often produced read pile-ups in repeat regions.

Subsequently the hashing strategy was revised to store *n* number of hash positions per seed and all hash position clusters were used to seed the full Smith-Waterman alignment. As the number of hash positions per seed increased, so did alignment accuracy and the memory requirements.

Three newer strategies are available:

- **SINGLE** (stores 1 hash position per seed)
- **MULTI** (stores 9 hash positions per seed)
- **ALL** (stores all hash positions per seed).

We have standardized on the use of the most accurate version (**ALL**) for all of our current lab projects.

Using Jump Databases

The aforementioned hashing strategies involve using hash maps as their primary data structure. However, putting an entire mammalian genome into a hash map is not very efficient. *MosaikAligner consumes around 57 GB RAM when using the ALL strategy and aligning against the entire mouse or human genome!*

Alignment Qualities

Alignment qualities are a new addition to MOSAIK. Similar to base qualities, alignment qualities give the probability that a read has been misaligned.

Alignment qualities occur on a logarithmic scale of 0 to 99. An alignment quality of 20 indicates that there is a 1 % chance that the alignment was misaligned, whereas an alignment quality of 30 indicates that there is a 0.1 % chance of misalignment.

We are currently adjusting the Bayesian equations governing alignment qualities. In practice, unique reads are assigned the highest alignment quality (99) because of normalization.

In non-unique reads, the best alignment usually receives a quality between 10 - 60 and the remaining alignments receive an alignment quality of 0 because of normalization.

Using a "jump database" makes full mammalian genome alignment possible (Figure 2.2). A jump database is comprised of two main files: the *keys file* and the *positions file*. The *positions file* is a tightly packed file with all of the possible hash positions for each seed. The *keys file* is a sparse look up table that allows MOSAIK to find the relevant hash positions with only two hops in memory. This data structure guarantees no collisions, which is often prevalent in highly loaded hash maps.

What does this mean to the user? *Instead of 57 GB, we can now deliver the same data at 19 GB RAM*. Since there are no collisions, alignments are faster.

As an added bonus, the user can select whether the keys and positions files should remain on disk or be loaded into memory for additional performance. Technically, this means that a full genome alignment can be performed with as little as 4 GB RAM (used by the mammalian genome reference sequence). However the random access usage patterns in the "jump database" still make this alternative prohibitively slow. Flash-based memory such as solid state drives (SSDs) might make this feature more practical.

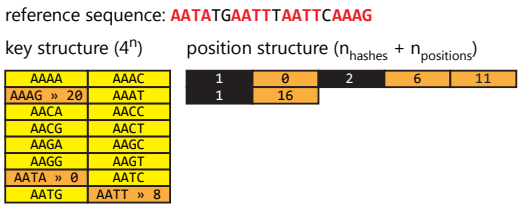


Figure 2.2 **The Jump Database Illustrated.**

In this example we are hashing the reference sequence with a hash size of 4. The hashes are store in 2-bit notation and thus have an equivalent integer value. e.g. AAAA=0, AAAC=1, AAAG=2, etc.

Using the integer value we can jump directly in the *keys file* to retrieve the *positions file* offset. e.g. if we look into the bucket corresponding to hash AAAG, we retrieve a file offset of 20.

Every record in the *positions file* starts with a number indicating the hash position count and is followed by the hash positions. e.g. at file offset 20, we discover one hash position at reference sequence offset 16.

The Art of Being Ambiguous and Masking

Many aligners convert reads and reference sequences into an efficient 2-bit format. While this may reduce the memory footprint, it discards valuable information contained in the IUPAC ambiguity codes.

MOSAIK uses the full set of IUPAC ambiguity codes during alignment (Figure 2.3). For example, if you use a reference sequence where all of the major confirmed SNPs from dbSNP are marked with the appropriate ambiguity codes, alignment bias for one allele over the other can be avoided. i.e. if the reference genome has an 'M' allele, a read position with an 'A' or 'C' at that location will be scored as a match.

The only ambiguity code that is not strictly implemented is 'N'. Many finished genomes uses large regions (often 50,000 bases) of N's to denote where two contigs are linked but the actual genomic data is missing. In order to prevent every single read from aligning to those regions, 'N' is interpreted as 'X' (meaning this does not align to any of the four nucleotides).

MOSAIK has no problems with reference sequences that are hard masked with X's.

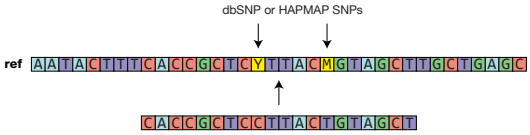


Figure 2.3 **Supporting IUPAC Ambiguity Codes.**

Bias against known SNPs can be reduced by using a reference sequence masked with IUPAC ambiguity codes. In the example, the alignment would result in zero mismatches.

Hash Size Selection (Speed vs. Sensitivity)

In general, using a larger hash size equates to a faster alignment speed. However this is often at the expense of sensitivity. e.g. using a hash size of 30 when aligning 35 bp reads will certainly be fast, but your reads will not be seeded if you have any internal sequencing errors or SNPs. In contrast, a hash size of 11 would guarantee that all reads with up to two mismatches are seeded but performance would suffer.

When aligning mammalian reads, a hash size of 15 provides a happy medium between speed and sensitivity.

Limiting Hash Positions

While scaling up to mammalian alignments, we added a feature (**-mhp**) that places a maximum number of hash positions per seed. Alignment representation bias is minimized by selecting a random subset of the hash positions.

When using a hash size of 15, each seed has an average of 5.25 hash positions in the human genome. Limiting the number of hash positions to 100 increases alignment speed significantly while having virtually no impact on alignment accuracy.

Setting a Minimum Cluster Size

A new feature that dramatically improves alignment speed with little impact on accuracy is the alignment candidate threshold (**-act**) (Figure 2.4). Normally all clusters are submitted for Smith-Waterman alignment. Initially we imposed a criterium (**-dh**) that two consecutive hashes had to be clustered before being aligned. This double-hit mechanism ensured that fewer spurious hash hits in the reference sequence would cause a full alignment to be performed.

The *alignment candidate threshold* extends this double-hit idea. An alignment candidate is simply the set of all seeds that form a cluster. The alignment candidate size is the length from the first base in the cluster to the last base in the cluster.

e.g. If a hash size of 11 is used and two hashes form cluster separated by a SNP, the alignment candidate size is 23. If the act paramater had been set to **-act** 20, this read would be submitted for Smith-Waterman alignment.

Uniqueness and Filters

MosaikAligner has a strict, but simple definition of what makes a unique read different from a non-unique read. If the read aligns to more than one location according to the user criteria, the alignments are non-unique. If the read aligns to a single location according to the user criteria, the alignment is unique.

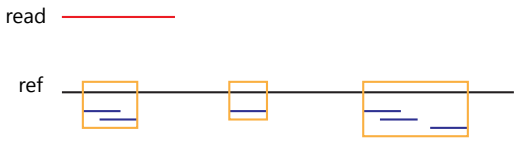


Figure 2.4 **Alignment Candidate Threshold.**

Before performing a pairwise alignment, MOSAIK hashes up each read and retrieves the hash positions for each seed in the reference sequence.

In this figure the hash positions are depicted as the horizontal blue lines. When we cluster these hash positions, three clusters are formed (orange boxes). Each cluster represents an alignment candidate that will be pairwise aligned.

Perhaps the middle cluster represents a spurious hit due to the repetitive nature of the reference sequence. To prevent spurious hits from eating up processing cycles, we can enforce that each cluster must have a specified length (the alignment candidate threshold) before being pairwise aligned.

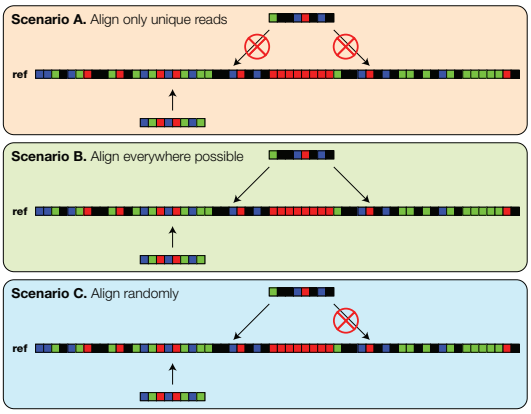


Figure 2.5 **Alignment Mode.**

MOSAIK has the ability to either place only the uniquely aligned reads (**-m unique**) or to place all reads (**-m all**). Other aligners choose a random location when placing non-unique reads, this benefit of this scenario confuses the author and therefore is not supported.

MOSAIK offers two different alignment modes: unique and all (Figure 2.5). When aligning in the unique mode (**-m unique**), alignment stops as soon as there is evidence that the read can be aligned to two places. Both alignments are saved to the alignment file which can be handled by downstream applications. When aligning in the all mode (**-m all**), the aligner finds all possible alignments. The all mode is suggested when trying to resolve paired-end reads.

When filtering reads, the mismatch threshold can be either a maximum number of mismatches allowed (**-mm**) or a maximum percentage of mismatches (**-mmp**) (w.r.t. read length).

Since Smith-Waterman is a local alignment algorithm, undesirable subsequences may be aligned. Normally, unaligned portions of the read are counted as mismatches, but this behavior can be disabled with the **-mmal** parameter. When using this parameter, it may be useful to force the alignments to have a minimum length with respect to the original read length. This threshold can be set with the **-minp** parameter.

We regularly allow four mismatches (**-mm** 4) for 36 bp Illumina reads and allow up to 5 % mismatches (**-mmp** .05) with variable length read technologies such as Helicos and 454.

Using MosaikAligner:

```
MosaikAligner -in myreads.dat -out h_sapiens_aligned.
dat -ia h.sapiens.dat -hs 15 -mm 4 -a all -m all -mhp
100 -act 20 -j h.sapiens_15 -km -pm -p 10
```

Here we specify an input read file (**-in**) and an output alignment file (**-out**) that stores all of the alignments. Additionally we specify a binary reference sequence file (**-ia**). All of the aforementioned parameters are required.

A hash size of 15 was specified (**-hs**) and a maximum of 4 mismatches is allowed (**-mm**).

All hash positions are initially stored (**-a all**) and all possible alignments will be reported (**-m all**). However, only 100 random hash positions will kept for each seed (**-mhp**). In each seed cluster, a minimum length of 20 bp is required (**-act**).

A jump database (**-j**) will be used instead of the normal hash map. Both the *keys file* (**-km**) and the *positions file* (**-pm**) will be copied into memory to increase performance.

A total of 10 processors (**-p**) will be used to increase alignment speed.

Box 2.2 **MosaikAligner Parameters.**

Input & Output

-in	specifies the input read archive.
-out	specifies the output alignment archive.
-ia	specifies the input reference sequence archive.
-rur	stores unaligned reads in a FASTQ file.

Essential Parameters

-a	specifies the alignment algorithm: fast, single, multi, all.
-m	specifies the alignment mode: unique or all.
-hs	specifies the hash size [4 - 32].
-p	uses the specified number of processors.

Filtering

-act	specifies the alignment candidate threshold.
-mm	specifies the number of mismatches allowed.
-mmp	specifies the maximum percentage of the read length are allowed to be errors. [0.0 - 1.0]
-mmal	uses the aligned read length instead of the original read length when counting errors.
-minp	specifies what minimum percentage of the read length should be aligned. [0.0 - 1.0]
-mhp	specifies the maximum number of hash positions to be used per seed.

Jump Database

-j	specifies the jump database filename stub.
-km	loads the *keys file* into memory.
-pm	loads the *positions file* into memory.

Pairwise Alignment Scores

-ms	the match score. Default: 10.
-mms	the mismatch score. Default: -9.
-gop	the gap open penalty. Default: 15.
-gep	the gap extend penalty. Default: 6.66.
-hgop	the gap open penalty used in homopolymer stretches when aligning with 454 reads. Default: 4.

2.3. MosaikSort

MosaikSort takes the alignment output and prepares it for multiple sequence alignment. For single-ended reads, MosaikSort simply resorts the reads in the order they occur on each reference sequence.

For paired-end reads, MOSAIK resolves the reads according to user-specified criteria before resorting the reads in the order they occur on each reference sequence.

Paired-End Read Resolution

When both mate-pairs are available, MosaikSort first samples the fragment lengths from uniquely aligned reads. Using this information, a minimum and maximum fragment length is calculated using the empirical 99.73% confidence interval.

When one mate is unique and the other is non-unique, the paired-end read is resolved if the program finds at most one alignment that occurs within the desired confidence interval (Figure 2.6). Similarly when both mate-pairs are non-unique, the paired-end read is resolved if the program finds at most one combination of alignments that fit the desired confidence interval.

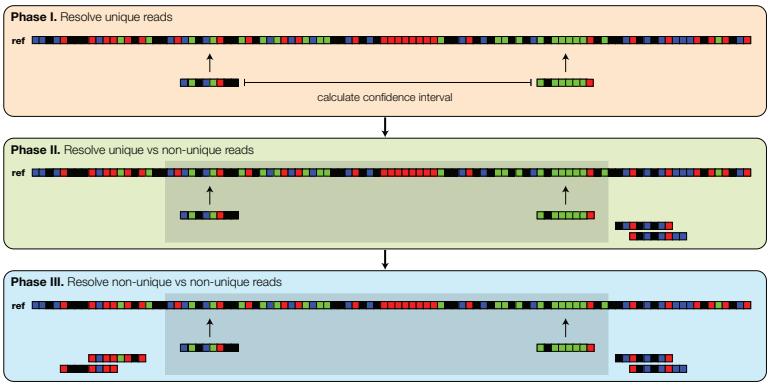


Figure 2.6 **Paired-End Read Resolution**

Using MosaikSort:

```
MosaikSort -in h_sapiens_aligned.dat -out h_sapiens_
resolved.dat -inu -uo
```

In the example above we specify an input alignment file (**-in**) containing paired-end reads and a sorted output alignment file (**-out**). Additionally we customize the paired-end resolution behavior by including orphaned unique reads (**-uo**) and by ignoring all reads where both mates are non-uniquely aligned (**-inu**).

Box 2.3 **MosaikSort Parameters.**

Input & Output

-in	specifies the input alignment archive.
-out	specifies the output alignment archive.

Single-end Options

-cs	specifies how many alignments to cache. Default: 6,000,000.
-u	use unique alignments only

Paired-end Options

-cs	specifies how many alignments to cache. Default: 6,000,000.
-afl	allows all fragments lengths when evaluating unique read pairs.
-ci	sets the fragment length confidence interval. Default: 0.9973.
-inu	ignores the resolution of non-unique vs non-unique reads.
-sap	samples fragment lengths from all unique read pairs.
-u	use unique mates only.
-uo	use uniquely aligned orphaned reads.

Paired-end Terminology

When discussing paired-reads, we use the term **mate** to mean one of the reads at each end of the fragment.

In MosaikAligner, each mate is aligned separately. Sometimes one mate aligns, but the other does not. We use the term **orphaned reads** to describe such an event.

2.4. MosaikMerge

MosaikMerge is not normally a part of the MOSAIK pipeline. It takes multiple sorted alignment archives from MosaikSort and merges them into a single alignment archive that can be processed by MosaikAssembler.

When aligning multiple runs belonging to different sequence libraries, the best practice is to use MosaikSort on each lane or run and then combine them with MosaikMerge. This practice avoids the unintended consequences of attempting to resolve paired-end reads when multiple fragment length distributions are present.

Another common use of MosaikMerge is to combine runs from different read technologies. This allows the researcher to combine 454, Illumina, Helicos, and Sanger capillary technologies into one co-assembly. At the moment, merging SOLiD reads with the other technologies is not supported because proper conversion from colorspace to basespace has not been added to MosaikMerge.

Using MosaikMerge:

```
MosaikMerge -in 454_aligned.dat -in helicos_aligned.
dat -in illumina_alignments -out coassembly.dat
```

In this example we combine the alignments from the two files, *454_aligned.dat* and *helicos_aligned.dat*, with all of the alignments contained in the directory *illumina_alignments*. The merged output will be saved in *coassembly.dat*.

Box 2.4 **MosaikMerge Parameters.**

Options

-in	specifies either an input alignment archive or a directory containing alignment archives. This parameter can be used multiple times.
-out	specifies the merged output alignment archive.
-cs	specifies how many alignments to cache. Default: 6,000,000.

2.5. MosaikAssembler

MosaikAssembler takes the sorted alignment file and produces a multiple sequence alignment which is saved in an assembly file format. At the moment, MosaikAssembler saves the assembly in the phrap ace format and the GigaBayes gig format, but other formats such as the gapped SAM format will be available in the future.

MosaikAssembler has been completely rewritten to take advantage of the sorted alignment files. As a result, assembly file creation is now orders of magnitude faster than before and limited only by sequential hard disk transfer speeds rather than slower random-access transfer speeds.

By default MosaikAssembler will assemble each reference sequence where reads aligned. Since the sorted alignment archives incorporate an index, a specific reference sequence can be assembled quickly with the region of interest (**-roi**) parameter.

Using MosaikAssembler:

```
MosaikAssembler -in h_sapiens_resolved.dat -ia
h.sapiens.dat -out h_sapiens -roi chrX
```

In this example, a sorted alignment file (**-in**) and a binary reference sequence file (**-ia**) are specified as input. MosaikAssembler will use the provided filename stub (**-out**) when generating an assembly specifically for the X chromosome (**-roi**).

Box 2.5 **MosaikAssembler Parameters.**

Input & Output

-in	specifies the input alignment archive.
-out	specifies the assembly output filename stub.
-ia	specifies the input reference sequence archive.

Options

-f	specifies the assembly file format: ace or gig
-roi	specifies the name of the reference sequence to assemble.

3. Utilities

MOSAIK contains three additional utility programs to facilitate data analysis: MosaikText, MosaikCoverage, and MosaikJump.

3.1. MosaikText

MosaikText converts alignments to different text-based formats. Currently it supports the BLAT axt (**-axt**) format, the UCSC Genome Browser bed format (**-bed**), the SAM format (**-sam**), and the Illumina ELAND (**-eland**) format. Alternatively alignments can be dumped directly to the screen (**-screen**) in an axt-like format.

In addition to examining alignment archives, MosaikText supports dumping the contents of read archives.

Using MosaikText:

```
MosaikText -in h_sapiens_aligned.dat -axt h_sapiens_
aligned.axt -u
```

In this example, the supplied alignment archive (**-in**) is exported to an AXT file (**-axt**). By enabling the **-u** parameter, only unique alignments are exported.

3.2. MosaikCoverage

MosaikCoverage is a handy program for investigating representational bias. This utility parses the alignment file and produces a base-accurate coverage plot (no binning) for each reference sequence that has coverage. In addition to the coverage plot, a simple space delimited coverage file is produced (Figure 3.1). MosaikCoverage uses *gnuplot* to generate the coverage graphs in PostScript. If available, it will call *ps2pdf* to convert the graphs into more portable pdf files.

Using MosaikCoverage:

```
MosaikCoverage -in h_sapiens_aligned.dat -ia
h.sapiens.dat -u -od graphs -cg
```

In the example above, the coverage is calculated from all of the reads in the alignment file *h_sapiens_aligned.dat* (**-in**) for each sequence specified in the reference sequence file *h.sapiens.dat* (**-ia**). By enabling the **-u** parameter, only unique alignments will be used to calculate coverage. All resulting files will be placed in the output directory *graphs* (**-od**) and pdf graphs will be generated (**-cg**).

Box 3.1 **MosaikText Parameters.**

Read Archive

-ir	specifies the input read archive.
-fastq	stores the data in the specified FASTQ file.
-screen	displays the reads on the screen.

Alignment Archive

-in	specifies the input alignment archive.
-u	limits the output to show only uniquely aligned reads.
-axt	stores the data in the specified BLAT AXT file.
-bed	stores the data in the specified UCSC Genome Browser bed file.
-eland	stores the data in the specified Illumina ELAND file.
-sam	stores the data in the specified SAM file.
-screen	displays the alignments on the screen in an AXT-like format.

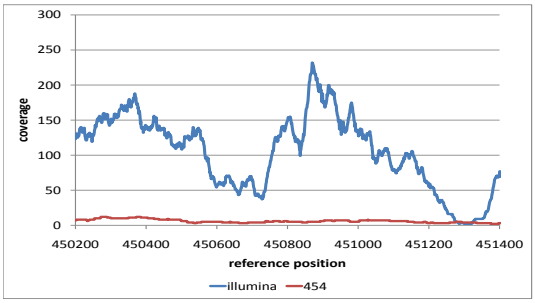


Figure 3.1 **Using MosaikCoverage in Excel.**

This graph depicts the observed coverage in *S. cerevisiae* chromosome 12 from both an Illumina and a 454 data set.

Box 3.2 **MosaikCoverage Parameters.**

Input & Output

-in	specifies the input alignment archive.
-ia	specifies the input reference sequence archive.
-u	limits the output to show only uniquely aligned reads.
-od	specifies the output directory.
-cg	creates coverage graphs if *gnuplot* is installed.

3.3. MosaikJump

MosaikJump is a tool that converts a reference sequence archive into a "jump database". As discussed earlier in section 2.2, the jump database is a custom data structure that separates seeds and hash positions in a manner that is fast and collision-free. As such, it works as a drop-in replacement for the traditional hash maps that MOSAIK normally uses.

An added benefit of using the jump database is that the user can choose which components should remain on disk and which should be loaded into memory. Maximum performance is realized when the entire jump database is loaded into memory.

The *positions file* is directly proportional to the number of hash positions in the reference sequences as well as the number of seeds present. e.g. Using a hash size of 15, there are roughly 2.9 billion hash positions in the human genome and 550,000 unique seeds. Assuming an integer is used to store each seed grouping and hash position, the positions file will be 12.7 GB.

The *keys file* stores the offsets for each seed grouping in the *positions file*. The *keys file* grows exponentially with each increasing hash size. e.g. Using a hash size of 15, the required file size is 5 GB ($5 * 4^{15}$). However if a hash size of 13 is used, the resulting file is only 313 MB. Due to the exponentially growing file size, the upper practical bounds for the jump database is a hash size of 15.

Box 3.3 **MosaikJump Parameters.**

Input & Output

-ia	specifies the input reference sequence archive.
-out	specifies the output filename stub for the jump database.

Options

-km	keeps the *keys file* in memory during jump database creation.
-mem	specifies the amount of RAM (GB) to use when sorting the hashes.
-hs	specifies the hash size [4 - 32].
-mhp	specifies the maximum number of hash positions to be used per seed.

Using MosaikJump:

```
MosaikJump -in h.sapiens.dat -out h.sapiens_15 -hs 15
-km
```

In the example above, MosaikJump will use the reference sequence file (**-in**) to generate the jump database files for hash size 15 (**-hs**) that start with the specified filename stub (**-out**). To improve performance during jump database creation, the keys file will be kept in memory (**-km**).

4. Performance

All statistics were measured on a lightly loaded computer with two 3.0 GHz Intel Xeon X5450 quadcore processors and 64 GB memory.

There is a direct correlation between MOSAIK alignment speed and the aggregate length of the reference sequences (the target genome). MOSAIK is extremely fast when processing yeast or roundworm-sized genomes and is faster than many aligners when processing mammalian genomes.

4.1. Speeding Up Alignments

There are a number of steps you can take to increase alignment speed.

Algorithmically speaking, the parameters that will help increase alignment speed are the following: hash size (**-hs**), alignment candidate threshold (**-act**), and the maximum hash position threshold (**-mhp**). The crux is that you want to maximize the improvement in speed while minimizing any negative impact on alignment accuracy and the percentage of reads aligned.

The larger the hash size, the more likely that hash will be unique in the reference sequence. Therefore increasing the hash size, reduces the number of spurious hash hits in the reference sequence. The consequence of increasing the hash size is that it also increases the number of bases that must exactly match the reference sequence. Our lab uses a hash size of 15 for most of our current analysis projects.

The alignment candidate threshold effectively dictates the minimum size of hash clusters before being submitted for pairwise alignment. Increasing this threshold reduces the number of spurious alignments being aligned. The consequence of increasing the threshold is that a read with a combination of sequencing errors and polymorphisms may also be filtered if the threshold is not met.

The maximum hash position threshold speeds up the alignment by reducing the number of hash positions that need to be clustered. In the human genome, each seed has an average of 5.25 hash positions when using hash size 15. Setting the threshold to 100 in such a case has minimal affect on the alignments while increasing the alignment speed considerably. The consequence of setting the threshold too low is that alignments in highly repetitive regions might no longer be seeded.

Perhaps the most obvious way to speed up alignments is to use more processors. MosaikAligner is fully multi-threaded and can handle any number of processors. The caveat is that the input/output and memory bandwidth may become saturated. e.g. On some of our 16 core systems, we have discovered that using 12 cores is usually faster than using 16 processor cores. You may have to perform a bioinformatics titration experiment where you align a subset of reads with an increasing number of processor cores in order to choose the configuration that works best for your system.

4.2. Yeast Alignments

MosaikBuild converted a SOLiD run of 61,516,412 reads (35 bp) in 7.4 minutes (139,000 reads/s).

MosaikAligner aligned the run in **9.5 minutes** (**109,000 reads/s**) to the entire *pichia stipitis* genome (15.4 Mbp) using 8 processor cores. We used the hash size 13 jump database for this experiment, so our total memory usage was 468 MB. 57.1 % of the reads aligned in this unfiltered data set.

MosaikSort sorted the uniquely aligned reads in **3.0 minutes**.

MosaikAssembler produced a GigaBayes assembly file with 27,592,793 Illumina alignments in **64 seconds** (**431,000 reads/s**).

Box 4.1 **Yeast Alignment Parameters**

-a all **-m** unique **-hs** 13 **-act** 20 **-mm** 6 **-mhp** 100 **-p** 8 **-j** jumpdb/p.stipitis_13cs **-km** **-pm**

4.3. Roundworm Alignments

MosaikBuild converted an Illumina run of 11,386,260 reads (35 bp) in 38 seconds (308,000 reads/s).

MosaikAligner aligned the run in **4.8 minutes** (**40,000 reads/s**) to the entire *C. elegans* genome (100 Mbp) using 8 processor cores. We used the hash size 15 jump database for this experiment, so our total memory usage was 5.8 GB. 91.6 % of the reads aligned in this data set.

MosaikSort sorted the output in **1.6 minutes**.

MosaikAssembler produced a GigaBayes assembly file with 12,323,293 Illumina alignments in **31 seconds** (**398,000 reads/s**).

Box 4.2 **Roundworm Alignment Parameters**

-a all **-m** unique **-hs** 15 **-act** 20 **-mm** 4 **-mhp** 100 **-p** 8 **-j** jumpdb/c.elegans_15 **-km** **-pm**

4.4. Human Alignments

MosaikBuild converted a lane of 6,563,762 Illumina paired-end reads (36 bp) in 2.9 minutes (53,000 reads/s).

MosaikAligner aligned the lane in **45.9 minutes** (**2,390 paired-end reads/s**) to the entire human genome (2.9 Gbp) using 8 processor cores. We used the hash size 15 jump

Box 4.3 **Human Alignment Parameters**

-a all **-m** all **-hs** 15 **-act** 20 **-mm** 4 **-mhp** 100 **-p** 8 **-j** jumpdb/h.sapiens_15 **-km** **-pm**

database for this experiment, so our total memory usage was 20 GB. 95.1 % of the mates aligned in this data set.

MosaikSort resolved the paired-end reads and sorted the output in **37 minutes** on a slower computer (2.0 GHz AMD Opteron 270).

MosaikAssembler produced a GigaBayes assembly file with 2,589,439,852 Illumina mates in **4.1 hours** (**174,000 mates/s**).

4.5. Aligner Settings By Sequencing Technology

By analyzing the performance characteristics and alignment accuracy, we have compiled a list of aligner settings (Box 4.4) that we typically use for each sequencing technology.

Box 4.4 **Aligner Settings.**

454 GS20 & FLX

-hs 15 **-mm** 0.05 **-act** 26

454 Titanium

-hs 15 **-mm** 0.05 **-act** 55

Illumina GA1 36 bp

-hs 15 **-mm** 4 **-act** 20

Illumina GA2 51 bp

-hs 15 **-mm** 6 **-act** 25

Illumina GA2 76 bp

-hs 15 **-mm** 12 **-act** 35

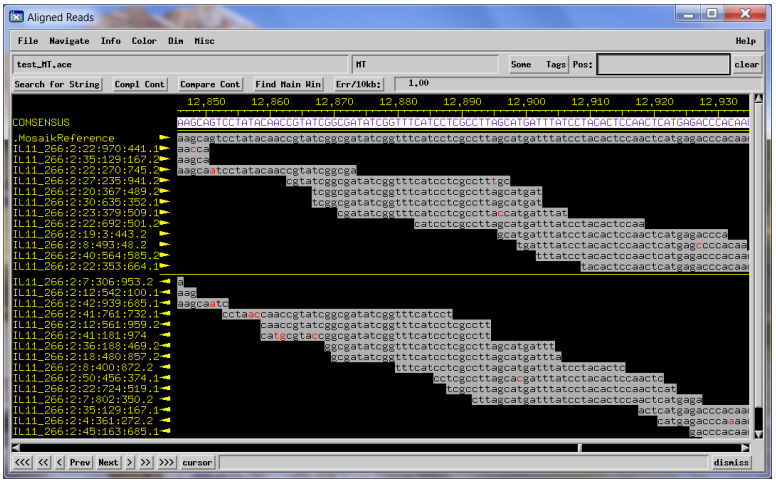
SOLiD 35 bp

-hs 15 **-mm** 4 **-act** 20

5. Visualization

MOSAIK works well with two visualization programs: consed and EagleView. Both of these programs reads ACE assembly files. An ace file is easily created by running the MosaikAssembler with the *-f ace* parameter on your sorted alignment archive.

5.1. consed



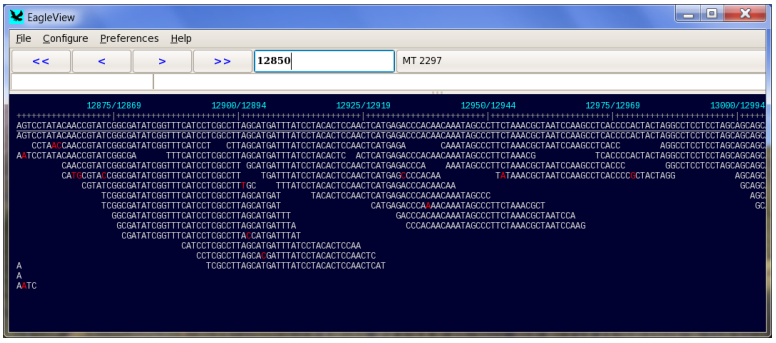
Box 5.1 **consed**

Consed can be downloaded from Phil Green's webpage at the University of Washington:

http://bozeman.mbt.washington.edu/consed/consed.html

When using consed with ace files created in MOSAIK, you should run it with the ***-nophd*** switch to prevent it from loading the non-existent phred sequence files.

5.2. EagleView



Box 5.2 **EagleView**

EagleView was developed in our lab by Weichun Huang. It can be downloaded from Marth Lab webpage:

http://bioinformatics.bc.edu/marthlab/EagleView

Compared with consed, EagleView shows a more compact view of the read assembly. EagleView's strength is that it can show any number of annotation tracks to help provide context to the read data.

6. Data Access (ReadMosaik)

A common problem when evaluating bioinformatics tools is data access. Users are often forced to write parsers for various file formats or output log files. MosaikText provides alignment data in several known text formats, but the ReadMosaik API provides even faster and easier access to the aligner results.

The API is written in C++ and can be found in the **ReadMosaik directory** contained in the MOSAIK package. By using the SWIG (*http://www.swig.org*) package, a version that is accessible in Perl is also available.

Here is a sample program that loads each read in a MOSAIK alignment archive:

```
#include <iostream>
#include "AlignmentReader.h"

using namespace std;

int main(int argc, char* argv[]) {

    // open the MOSAIK alignments file
    Mosaik::CAlignmentReader reader;
    reader.Open("myreads_aligned.dat");

    // get some basic statistics
    uint64_t numBases = reader.GetNumBases();
    uint64_t numReads = reader.GetNumReads();

    cout << "# of bases: " << numBases << endl;
    cout << "# of reads: " << numReads << endl;

    // keep reading all of the sequences
    Mosaik::AlignedRead ar;
    while(reader.LoadNextRead(ar)) {

        // do something with each read
        cout << "read name: " << ar.Name << endl;
    }

    // close the alignments file
    reader.Close();

    return 0;
}
```

Box 6.1 **Read Data Structures**

The *AlignedRead* data structure contains all of the mate 1 and mate 2 alignments for one read. If the read is single-ended, only the *Mate1Alignments* vector will contain alignments.

```
struct AlignedRead {
    string Name;
    vector<Alignment> Mate1Alignments;
    vector<Alignment> Mate2Alignments;
};
```

In the *Alignment* data structure, *ReferenceBegin* and *ReferenceEnd* contain the start and end locations on the reference sequence. *ReferenceName* contains the name of the reference sequence.

QueryBegin and *QueryEnd* contain the start and end locations on the read.

The gapped pairwise alignment is contained in the strings *Reference* and *Query*.

Each character in the *BaseQualities* string contains a base quality for each pairwise aligned base in the same orientation as the alignment. Extra base qualities have NOT been introduced for gaps in the alignment. There is no offset between each character and the intended base quality.

Quality contains the alignment quality which measures the probability that an alignment has been misaligned.

IsReverseComplement is set to true if the alignment is on the 3' or Crick strand.

```
struct Alignment {
    unsigned int ReferenceBegin;
    unsigned int ReferenceEnd;
    unsigned short QueryBegin;
    unsigned short QueryEnd;
    unsigned char Quality;
    bool IsReverseComplement;
    char* ReferenceName;
    string Reference;
    string Query;
    string BaseQualities;
};
```

Appendix 1: Crib Sheet

MosaikBuild

```
MosaikBuild -fr myreads.fasta -fq myreads.fasta.qual -out myreads.dat

MosaikBuild -fr myreference.fasta -oa myreference.dat
```

-fr, **-fr2** specifies the FASTA files containing the bases for the first mate (-fr) and the second mate (-fr2).
-fq, **-fq2** specifies the FASTA files containing the base qualities for the first mate (-fq) and the second mate (-fq2).
-q, **-q2** specifies the FASTQ files for the first mate (-q) and the second mate (-q2).
-cs translates the reference sequence from base space to colorspace.
-st specifies the sequencing technology: 454, helicos, illumina, sanger, solid
-out specifies the output file when converting reads.
-oa specifies the output file when converting the reference sequence.

MosaikAligner

```
MosaikAligner -in myreads.dat -out myreads_aligned.dat -ia myreference.dat -hs 15 -mm 4 -a
all -m all -mhp 100 -act 20 -j myjumpdb -km -pm -p 8
```

-in specifies the input read archive.
-out specifies the output alignment archive.
-ia specifies the input reference sequence archive.
-rur stores unaligned reads in a FASTQ file.
-a specifies the alignment algorithm: fast, single, multi, all.
-m specifies the alignment mode: unique or all.
-hs specifies the hash size [4 - 32].
-p uses the specified number of processors.
-act specifies the alignment candidate threshold.
-mm specifies the number of mismatches allowed.
-mmp specifies the maximum percentage of the read length are allowed to be errors. [0.0 - 1.0]
-minp specifies what minimum percentage of the read length should be aligned. [0.0 - 1.0]
-mhp specifies the maximum number of hash positions to be used per seed.
-j specifies the jump database filename stub.
-km loads the jump database *keys file* into memory.
-pm loads the jump database *positions file* into memory.

MosaikSort

```
MosaikSort -in myreads_aligned.dat -out myreads_sorted.dat -inu -uo
```

-in specifies the input alignment archive.
-out specifies the output alignment archive.
-ci sets the fragment length confidence interval. Default: 0.9973.
-inu ignores the resolution of non-unique vs non-unique reads.
-u use unique mates only.
-uo use uniquely aligned orphaned reads.

MosaikAssembler

```
MosaikAssembler -in myreads_sorted.dat -ia myreference.dat -out myassembly
```

-in specifies the input alignment archive.
-out specifies the assembly output filename stub.
-ia specifies the input reference sequence archive.
-f specifies the assembly file format: ace or gig
-roi specifies the name of the reference sequence to assemble.