Filtering read alignments in BAM format: user guide

Tonatiuh Peña-Centeno
University of Greifswald

February 28, 2012

Abstract

This note documents ”filterBAM”, a program designed to clean alignments stored in BAM format. The filter is based on filterPSL, a perl script written by Prof. Mario Stanke as part of the AUGUSTUS software suite [Reference]. Both filterPSL and filterBam are designed for the cleaning of data that will subsequently be applied to the gene prediction problem. The filter produces also output that might later be used in visualisation tools, such as UCSC browser. The code should be modifiable rather easily, if it is to be applied to a different type of application (see the reference manual below for further information). filterBam is written in C++ and makes use of the Bamtools API of Barnett et al. (2011).

1 Installation

In this section we describe how to install filterBam on a Linux terminal.

1.1 Requirements

This software has been tested on a a Dell (x86_64) computer with Ubuntu 10.04 (lucid). Compilation of the code was done with GNU's C++ compiler, gcc version 4.4.3. Table 1.1 below lists the required software packages for compiling filterBam; while 1.1 shows the libraries necessary for compilation. In order to work with BAM files, it might be useful to have samtools installed, so this has also been included in the tables as an optional package.

Note that the flag -std=c++0x has been used given that some of the functionalities of the filter require some of the newest features of GNU's g++ compiler. It seems that c++11 has bundled such functionalities.

1.2 Compilation

1. Download the latest version of filterBam from [to define].
2. Edit the Makefile so that the BAMTOOLS_ROOT variable points to where Bamtools has been installed. [1]

[1] If you do not happen to know where Bamtools is, Makefile will look for it by calling the UNIX `find` command, but this may slow compilation time a bit.

Software Dependencies		
Name	version	Available at
Bamtools	2.1.0	`https://github.com/pezmaster31/bamtools`
Samtools (optional)	0.1.18	`http://samtools.sourceforge.net/`

Table 1: List of software required by filterBam

Library Dependencies			
Names	version	Available at	Notes
zlib	>= 1.2.2.1	`<http://www.zlib.net>`	For support of BGZF format

Table 2: List of libraries required by filterBam

3. Type 'make' to generate the binaries, which will be stored in the **bin** directory.

2 A couple of examples

In this section we show how the filter works through the application of a couple of examples. The first example documents the operation of the filter for single alignments, while the second example describes the operation of the filter in paired-alignment mode.

2.1 Test data

We have generated two data sets to show filterBam in operation.

2.2 Running

A run that will let pass most, if not all, readings:

./filterBam input.bam output.bam –minCover 0 –minId 0 –insertLimit 10000000
–nointrons

Note: that all options are provided at the very end.

3 Some notes about Samtools and Bamtools

We introduce some examples of how to use Samtools and Bamtools to make life easier when working with BAM files. Probably the main issue when working with BAM data is that alignments need to be sorted, but beign binary implies that

3.1 Samtools

3.2 Bamtools

Bamtools is a C++ wrapper API of the more well-known Samtools software. The latest version of Bamtools is 2.0 and is specificed on Table 1.1.

4 Technical stuff

Other relevant issues that might be well documenting go here.

4.1 Input data

The filter should work fine for data coming from 454 and Illumina technologies but not for colorspace data generated by SOLiD technology.

Filtering read alignments in BAM format: reference manual

Abstract

This note describes the operation of the filter and lists the main classes that were used to implement it. In this way, other developers might .

5 Introduction

RNA-seq data has become an important source of information for tasks such as differential expression analysis, transcript quantification and gene prediction. Given that this new technology produces millions of such short-reads ($\sim 30bp$), bespoke methods and tools are required to process such big amounts of information. For example, a single run of an RNAseq experiment will produce millions, if not, hundreds of millions of short reads [Ref: Wiki].

After sequencing and generation of an RNAseq dataset, a later step consists of utilising the short reads to obtain an approximate version of the transcriptome, typically via the alignment of the reads to a reference genome. Such alignments can be carried out with tools such as BLAT, Bowtie, GSNAP, among others. In particular, Bowtie and GSNAP are specifically designed to align reads as short as 50 and 14 bp, respectively. Very recently, the introduction of the Sequence AlignMent Format, or SAM, by Li et al. (2009), has meant that many of the aforementioned alignment tools now produce outputs in SAM format.

filterBam is a C++ code that cleans alignment files stored in BAM format, which is the binary version of SAM. The software package is based on filterPSL, a Perl routine written by Prof. Dr. Mario Stanke for the processing of alignment records stored in PSL format. filterPSL is part of a set of Perl scripts that accompany the distribution of the annotation software, AUGUSTUS [Ref]. filterPSL is mainly used as a preprocessing step for cleaning alignments obtained with softwares such as BLAT, and filterBam is supposed to supersede it by doing the same task but on RNAseq alignment data stored in a BAM file.

6 Main features

In a nutshell, assuming a BAM file given as input, filterBam by default cleans all those alignments that are either unmapped or do not satisfy any of the following conditions:

1. do not comply with a minimum coverage;
2. do not have a minimum value of percentage identitiy, or
3. (optionally), do not satisfy a minimum value of base inserts.

Table 3 above summarises the main features of the filter.

After this basic set of filters has been applied, the alignments are processed according to whether they originated from single- or paired- RNAseq reads. Single alignments are cleaned by droppping out all those that do not satisfy a score value that depends on the coverage and the percentage identity of the aligned read, i.e. $score(coverage, percId)$. Paired alignments are mated to other alignments according to the distance and insert length from their associated reads; the filter then drops out all those pairs of alignments that do not satisfy a score value that, once again, depends on coverage and percentage identity, ($score(coverage, percId)$).

The subsequent sections of this document describe in a step-wise manner how the filtering of single- and paired-alignments is done. The basic set of filters is

filterBam			
Action	Feature	Option	Default value
Every alignment			
Screens out	unmapped	–	–
	coverage level	minCover	80%
	pctge identity	minId	92%
	insert gaps	insertLimit	10bp
Single alingments			
Screens out	best	–	nore
	unique	uniqThresh	0.96
Paired alignments			
Screens out	best		
	uniq	uniqThresh	0.96
Writes to file	common target genes	commonGeneFile	false
	pairedness coverage info	pairBedFile	false

Table 3: Main features of filterBam.

described in Section 7. Then the filtering of single alignments is explained in Section 8, and finally the filtering of paired alignments is explained in Section 9.

7 Basic filters

Figure **??** below shows the schematics of the operation of the filter for single alignments. In the subsequent, we will assume an input BAM file is constituted by a series of records $i = \{1, \ldots, N\}$, each containing the information of an alignment. See (Li et al., 2009) for further reference. The filter first checks whether alignment i is mapped or not, and this is easily done by means of verifying the bit 0×4 of the alignment FLAG (SAM field number 2). As the specification suggests, this bit is the only source of reliable information to determine whether a read is mapped or not Li et al. (2009). This verification is achieved by using the isMapped method of BamTools. Unmapped reads are dropped, while mapped reads continue further processing. A counter keeps track of the number of unmapped reads that were dropped.

As a second step, alignments that passed the mapping test are appended with two additional but temporary string-tags. Tag 'co' and tag 'pi' are added to the binary alignment by the addTag method of BamTools. 'co' stands for *coverage* and is a measure of the amount of reads located at a given genomic position. 'pi' stands for *percentage identity* and is a measure of the number of basis that correctly identify a genomic position. Estimation of the coverage is done according to Equation 2, whilst estimation of the percentage identity is done following Equation 3, both in the Appendix.

If the estimated coverage value for the read in alignment i is less than that of the specified minCover, the alignment will be dropped and a counter keeping track of such types of events will be updated. In a similar way, if the value of percId for the read of alignment i, is less than that specified by minId, the alignment will be dropped and the corresponding counter will be updated. Default values for minCover and percId are shown in Table 3 respectively, but might be modified by using the options --minCover⟨value⟩ and --minId⟨value⟩.

8 Single alignments

Continuing with Figure **??**, we assume either options best or uniq are selected, but not option paired. The core issue to understand in the operation of the single-alignment mode filter, is that batches of alignments belonging to a common query QNAME_1 will be processed independently from alignments belonging to a different query name QNAME_2.

Alignments that passed the mapping test are appended with two additional but temporary string-tags. Tag *co* and tag *pi* are added to the binary alignment by means of the addTag method of BamTools. Tag *co* stands for *coverage* and is a measure of the amount of reads located at a given genomic position. Meanwhile *pi* stands for *percentage identity* and is a measure of the number of basis that correctly identify a genomic position. Whereas estimation of the coverage is done according to Equation 2, estimation of the percentage identity is done according to Equation 3, both in the Appendix. Table 4 below shows a series of alignments with the *co* and *pi* tags added.

If the estimated coverage value of alignment i is less than that of minCover, the alignment will be dropped and a counter keeping track of such types of drops will be updated. In a similar way, if the value of percId for alignment i, is less than that specified by minId, the read will be dropped and the corresponding counter will be updated. Default values for minCover= 80 and percId= 92 might be modified by using the options --minCover⟨value⟩ and --minId⟨value⟩, respectively.

An optional value, the number of inserts to the base reference (baseInsert), is computed optionally if the '–noIntrons' option is used. The number of insertions to the reference is computed through the application of Equation 4. This filter depends on the insertLimit value that has been specified, and which by default has a value of 10. The insertLimit parameter might be modifed by applying the --insertLimit⟨value⟩ option.

8.1 Uniq and Best criteria

Further cleaning can be achieved by means of selecting the mutually exclusive unique or best options. If such is the case, Figure 1 shows how an alignment record continues throughout the process path. Options best and unique stand for the filter selecting the *best* group of alignments, or the single-top alignment (i.e. unique), in terms of a cost function, which in this case is given defining the expression

$$\text{score} = \text{percId} + \text{coverage}. \tag{1}$$

Thus, after an alignment has passed through the mapping, coverage, percentage identity and intron-gap filters, the information from coverage and percentage identity is be combined into the figure score. Such value is added to the alignment as the tag *sc*.

After a group of alingments belonging to the same query has been scored, the group is sorted by such score value; as illustrated in Tables 4 and 5 below.

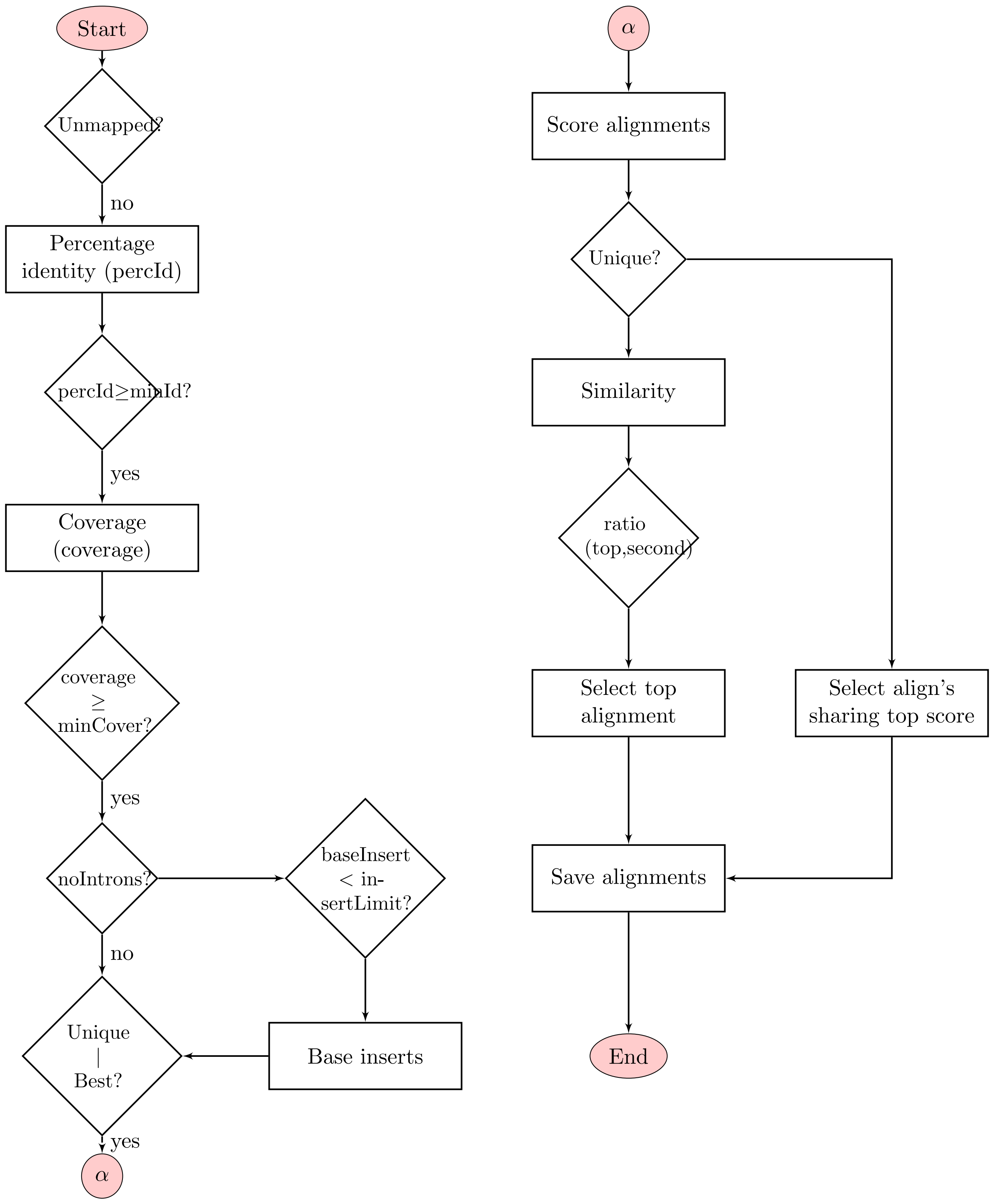

Figure 1: Flow diagram of the operation of the single-read filter

QNAME	RNAME	startPOS	endPOS	pi	co	sc
r2/1	chr17	27698729	27698778	98	100	198
r2/1	chr17	20320140	20320189	94	100	194
r2/1	chr19	1364	1413	98	100	198
r2/1	chr17	8038458	8038507	96	100	196
r2/1	chr17	24524223	24524271	94	100	194
r2/1	chr17	30676704	30676750	96	96	192
r2/1	chr17	16894327	16894376	94	100	194
r2/1	chr17	5031882	5031931	96	100	196
r2/1	chr18	0	49	98	100	198

Table 4: SAM alignments with added tags: percId, coverage and score

QNAME	RNAME	startPOS	endPOS	pi	co	sc
r2/1	chr17	27698729	27698778	98	100	198
r2/1	chr19	1364	1413	98	100	198
r2/1	chr18	0	49	98	100	198
r2/1	chr17	8038458	8038507	96	100	196
r2/1	chr17	5031882	5031931	96	100	196
r2/1	chr17	20320140	20320189	94	100	194
r2/1	chr17	24524223	24524271	94	100	194
r2/1	chr17	16894327	16894376	94	100	194
r2/1	chr17	30676704	30676750	96	96	192

Table 5: SAM alignments after sorting by score

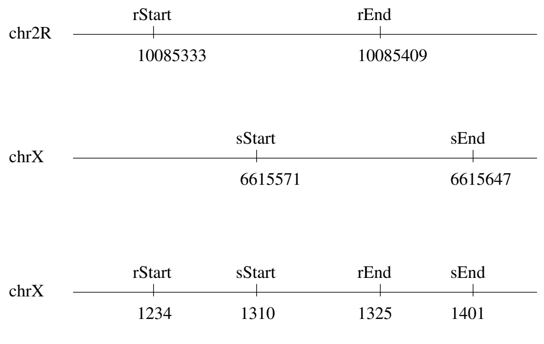

Figure 2: The similarity function checks whether mate paired reads are overlapping or not.

The difference between the uniq and best criteria is that the former will select only the top-scored alignment and will write it into file. Table 5 shows the same set of alignments as in Table 4, but after ranking by score. According to the best criterion, only the alignmnents sharing the optimal score will be preserved, while the rest of suboptimal alignments will be dropped. In the example of Table 5, the alignments sharing the score=198 will be preserved while the rest will be discarded.

The option uniq is based also on the sorting of alignments by its score. The main difference however is that only one alignment is preserved, provided that it is one of those sharing the top score, but also provided that it is is elligible to be preserved. However, in case a group of alignments happen to share the same score, filterBam checks whether such alignments are similar; whereby similarity refers to two reads to have been aligned on overlapping positions.

8.2 Similarity function

A function that tests whether alignments (or two alignment pairs) are similar has been included within filterBam. Testing for similarity is required given that by handling separately spliced and unspliced alignments, there is the possibility that very similar alignments are reported, *an unspliced read going approximately up to an intron and a spliced read with a few base pairs on one exon.* Such type of cases should not be considerd ambiguous when uniq is specified.

Figure 2 below shows two scenarios. In scenario one, a pair of reads were aligned to overlapping ranges of the reference genome; both reads are deemed similar. In scenario two, the reads are aligned to non-contiguous ranges of the reference genome, thus are considered not-similar.

Thus to finalise, a top-scored alignment will be let pass by the filter, if and only if, the second ranked alignment is not all too-similar to the top alignment.

9 Paired alignments

This section describes the filtering of paired alignments. This feature is enabled by selecting the option paired. By doing so, filterBam will compare a set of alignments belonging to a common query and determine which alignments are paired with which others. Such matching is done by examining the distance and insert length between candidate pairs. A more thorough explanation follows, nevertheless it is worth pointing out that before alignments are processed as paired alignments, they are subjected to the basic filters described in Section

Mate 1	Mate 2	Strand 1	Strand 2	dist	insLen	score
rs1/1 (70)	rs1/1 (71)	false	false	–	–	–
rs1/1 (70)	rs1/2 (138)	false	true	57	77	3.8
rs1/1 (70)	rs1/2 (201)	false	true	120	138	3.875
rs1/1 (70)	rs1/2 (499)	false	false	–	–	–
rs1/1 (71)	rs1/2 (138)	false	true	58	76	3.55
rs1/1 (71)	rs1/2 (201)	false	true	121	137	3.625
rs1/1 (71)	rs1/2 (499)	false	false	–	–	–
rs1/2 (138)	rs1/2 (201)	true	true	–	–	–
rs1/2 (138)	rs1/2 (499)	true	false	–	–	–
rs1/2 (201)	rs1/2 (499)	true	false	–	–	–

Table 6: Candidate mate pairs in the example presented in Section 9

7. A flow chart of the operation of filterBam for paired alignments is shown in Figure 3 below.

9.1 Mate pairs

Figure 4 shows a diagram in which four reads have been aligned: rs.1 (71), rs.2 (72), rs.2 (139) and rs.1 (202); with starting positions between parentheses. Bare in mind that query names have been made to coincide, in order to facilitate the understanding of the matching process. We recall that filterBam accepts inputs with '/1', '/2' suffix when the option paired has been selected.

Within a group of alignments, presented in the example as queries *rs*, filterBam defines a list of candidate mate pairs, as shown in Table X below. If one pair of alignments belongs to different mates 1,2 and come from different strands +,-, then their distance and insert-length is computed. If a pair of alignments has $dist \geq 0$ and $insLen \leq maxInsertLimit$, the alignments are considered a valid mate-pair.

9.2 Uniq and Best criteria

Figure 3 shows the flow chart of operation of filterBam for paired alignments. As it can be seen, the filter operates under very similar tenets to those of the filter for single alignments, the main difference being that under the paired option, alignments are processed in pairs. Thus after scoring of the alignments of forming of the mate-pairs, the uniq selects the top-ranked pair of mates; where the rank is given by a function that makes use of the *coverage* and *percId* in very similar terms to those of Equation 1. Analogously as well, the option best, lets pass the set of mate-pairs that share the maximum score. It is important to remark that alignments that were not paired are dropped.

10 Coverage, percent of identity and insert length

The coverage is computed as the sum of the alignment matches (sequence matches or mismatches) and the insertions to the reference. Both figures, alignment matches and insertions to the reference, correspond to CIGAR string operations M and I, respectively. Thus the following is done

$$\text{coverage} = \frac{\sum \text{CIGAR}\,(M, I)}{qLength} \tag{2}$$

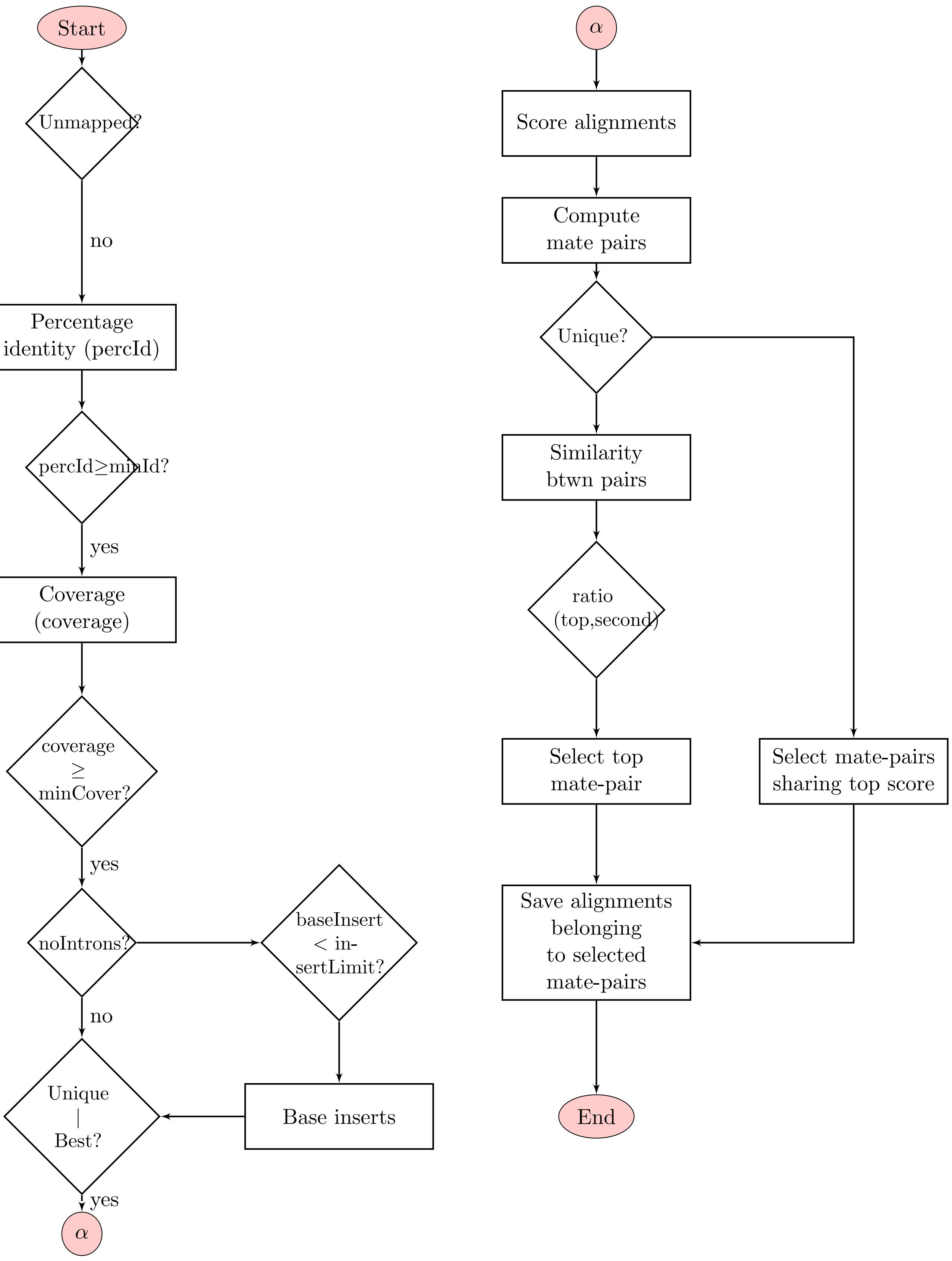

Figure 3: Flow diagram of the operation of the single-read filter

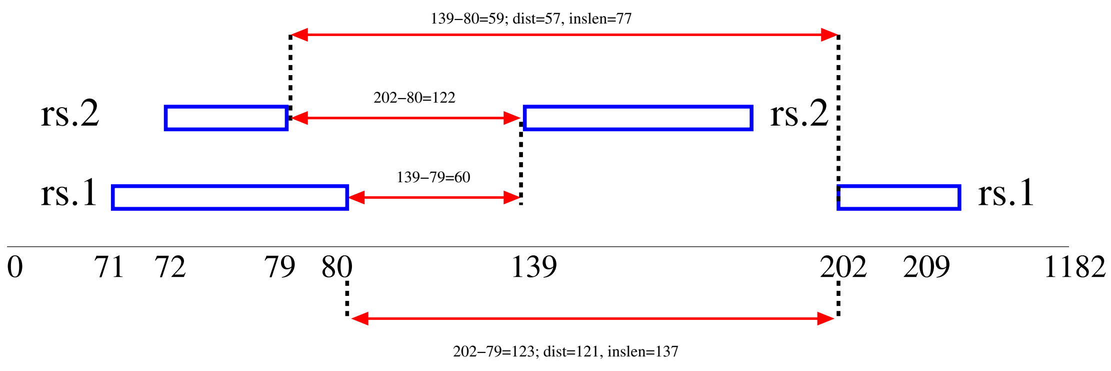
139−80=59; dist=57, inslen=77
202−80=122
139−79=60
rs.2
rs.2
rs.1
rs.1
0
71
72
79
80
139
202
209
1182
202−79=123; dist=121, inslen=137

Figure 4: Paired reads

An approximation to the percentage of identity is given by computing the query length and subtracting the so-called edit distance to the reference (tag "NM" in SAM jargon), i.e.

$$\text{percId} = \frac{qLength - \text{Tag}(NM)}{qLength} \tag{3}$$

The length of inserts is estimated by summing CIGAR operations "M" and "I", which correspond to alingment matches and deletions from the reference. In other words, we do the following

$$\text{InsertSize} = \frac{\sum \text{CIGAR}\,(D, I)}{qLength} \tag{4}$$

References

H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Math, G. Abecasis, R. Durbin, and . G. P. D. P. Subgroup. The sequence alignment/map format and samtools. *Bioinformatics Applications Note*, 25(16): 2078–2079, 2009.

D. Barnett, E. Garrison, A. Quinlan, M. Strmberg, G. Marth. BamTools: a C++ API and toolkit for analyzing and managing BAM files. *Bioinformatics*, 27(12):1691-1692, 2011.