

# **Hapler Version 1.53**

Shawn T. O'Neil and Scott J. Emrich

University of Notre Dame, Notre Dame, IN 46556, USA  
`{soneil,semrich}@nd.edu`

# Table of Contents

Hapler Version 1.53 .....	1
<i>Shawn T. O'Neil, Scott J. Emrich</i>	
1 Overview .....	4
2 Changelog .....	5
3 Requirements and Installation .....	6
4 Example Usage .....	7
5 Input and Options .....	9
5.1 --input <filename or -> .....	9
5.2 --alignment-type < <i>tigr</i> or <i>sam</i> > .....	9
5.3 --help .....	9
5.4 --maximize-one-read-haps < <i>true</i> or <i>false</i> > .....	9
5.5 --random-repetitions <i>auto</i> or <integer> .....	9
5.6 --max-repetitions <integer> .....	10
5.7 --snp-caller <simple, <i>simplestrict</i> , or 454> .....	10
5.8 --snp-list <filename> .....	10
5.9 --ground-truth <filename> .....	10
5.10 --evaluate-contigs <filename> .....	11
5.11 --compute-reconstructions < <i>true</i> or <i>false</i> > .....	11
5.12 --human-readable < <i>true</i> or <i>false</i> > .....	11
5.13 --show-alignments .....	11
5.14 --version .....	11
6 Output .....	12
6.1 Consensus Sequence Output Lines .....	13
Consensus (> prefixed) Column 1 - Unique Consensus Identifier .....	13
Consensus (> prefixed) Column 2 - Alignment Name .....	13
Consensus (> prefixed) Column 3 - Consensus Type .....	13
Consensus (> prefixed) Column 4 - Minimum Hapler Crossovers .....	13
Consensus (> prefixed) Column 5 - Maximized SNP Allele Support .....	13
Consensus (> prefixed) Column 6 - Minimized Unique Haplotype Regions Used .....	13
Consensus (> prefixed) Column 7 - Hapler Haplotype Regions Used .....	13
Consensus (> prefixed) Column 8 - Consensus Sequence .....	14
Consensus (> prefixed) Column 9 - Ground Truth Minimized Crossover Number .....	14
Consensus (> prefixed) Column 10 - Ground Truth Minimized Unique Haplotypes Used .....	14
Consensus (> prefixed) Column 11 - Ground Truth Haplotypes Used .....	14
6.2 Haplotype Region Output Lines .....	15
Haplotype (@ prefixed) Column 1 - Unique Haplotype Identifier .....	15
Haplotype (@ prefixed) Column 2 - Alignment Name .....	15
Haplotype (@ prefixed) Column 3 - Alignment Length .....	15
Haplotype (@ prefixed) Column 4 - Haplotype Block Number .....	15

Haplotype (@ prefixed) Column 5 - Minimum Number of Haplotypes Supported By Block .....	15
Haplotype (@ prefixed) Column 6 - Number of Repetitions Completed .....	15
Haplotype (@ prefixed) Column 7 - Haplotype Region Number .....	15
Haplotype (@ prefixed) Column 8 - Number of Non-Redundant Sequences ...	15
Haplotype (@ prefixed) Column 9 - Number of Redundant Sequences .....	16
Haplotype (@ prefixed) Column 10 - Number of Defined SNPs Covered .....	16
Haplotype (@ prefixed) Column 11 - Number of Defined SNPs Not Covered ..	16
Haplotype (@ prefixed) Column 12 - Start Position of Haplotype Region .....	16
Haplotype (@ prefixed) Column 13 - End Position of Haplotype Region .....	16
Haplotype (@ prefixed) Column 14 - Length of Haplotype Region .....	16
Haplotype (@ prefixed) Column 15 - Number of Pieces Haplotype Is In .....	16
Haplotype (@ prefixed) Column 16 - Average Sequence Coverage of Haplotype	17
Haplotype (@ prefixed) Column 17 - Base Coverage at SNP Positions .....	17
Haplotype (@ prefixed) Column 18 - Alignment Coverage .....	17
Haplotype (@ prefixed) Column 19 - Haplotype Region Assembly at SNP Positions .....	17
Haplotype (@ prefixed) Column 20 - Full Haplotype Region Assembly .....	17
Haplotype (@ prefixed) Column 21 - Reads In Haplotype .....	17
Haplotype (@ prefixed) Column 22 - Ground Truth Sequences Exactly Matched .....	17
Haplotype (@ prefixed) Column 23 - Ground Truth Sequence Mismatch Counts .....	18
7 Maximizing Quality .....	19
7.1 Read Length and Coverage .....	19
7.2 SNP Calling and Error Correction .....	19
7.3 Sampling Number .....	20
8 How it works .....	21
A Example TIGR Input Format .....	25
B Example SAM Input Format .....	26
C Example SNP Input Format .....	27

## 1 Overview

Suppose you are given a alignment of sequencing reads (such as those that contribute to a consensus contig in an assembly, or a mapping of reads against a reference, see Section 5) that represent an unknown number of haplotypes. For example, perhaps you have pooled material for a number of (non-clonal) individuals and you sequenced from the ND5 gene of each and aligned them to a reference. Hapler is a tool that attempts to assemble these haplotypes individually. If there is enough information (high enough coverage, long enough reads, low enough error rates, high enough differentiating diversity in the dataset), there is a good chance this can be done. If not, Hapler takes care to only assemble shorter haplotype regions it is “sure” of.

Once these haplotype regions are assembled, Hapler also reconstructs an overall consensus that attempts to minimize and identify points of possible chimerism (points in the consensus where different SNPs must be supported by different haplotype regions).

For a quick introduction into the basic use of Hapler see Section 4. For considerations on maximizing quality, see section 7.

## 2 Changelog

### 1.53

- Tweaked the output format and warnings format (again) for ease of description and parsing.
- Implemented custom SNP loading from a text file (finally).
- Removed the `--allow-gaps` option; by default gaps are split and mate-pair information is ignored; a warning is output when this happens.

### 1.52

- Adjusted the output format to be more parseable, hopefully.
- Added an optional `--human-readable` option to decrease output size for large datasets.
- Fixed a bug where the program would crash when given an alignment with no SNPs.
- Code cleanup and fixes.

### 1.51

- Fixed bug with parsing SAM format, if cigar string is “\*”, the read is skipped.

### 1.5

- Internal logic cleanup
- Added `auto` option for `--random-repetitions`.
- Added `--max-repetitions` option.
- Added minimum-chimerism consensus reconstruction, evaluation of given contig consensus and majority vote.
- Fixed an error in the 454 SNP caller where SNPs could be called even if the majority vote was a gap.

### 1.01

- Added ability to have ~’s in TIGR formatted sequences, so that `--allow-gaps` (split) can be used with them).

### 1.0

- Initial release.

### 3 Requirements and Installation

Hapler is written in Java (as a single .jar file), and has been tested with Java 1.6.0\_22 (on OSX 10.6.6). It is released under the LGPL. To install and use Hapler, one simply needs to acquire the .jar file and run `java -jar Hapler.jar` on the command line.

## 4 Example Usage

The default parameter settings are such that Hapler will assemble haplotype regions and reconstruct a minimum-chimerism consensus in a conservative fashion for low diversity, low-coverage data (e.g., an arthropod EST sequencing project). Hapler reads TIGR formatted assemblies on standard input by default and uses the built in “simplestrict” SNP caller (requiring 2x coverage of the minority allele for calling SNPs, see section 5).

Thus, good results from Hapler should be attainable by running something as simple as:

```
cat pz_p450_CP6B6_pop.tigr | java -jar Hapler.jar
```

Reading on standard input isn’t required, of course:

```
java -jar Hapler.jar --input pz_p450_CP6B6_pop.tigr
```

Looking at the result of the above by piping into `less -S`, we can see that Hapler returns a lot of information in column/row format, with informative comments and warnings (lines starting with `#` for easy filtering) describing each column. See Section 6 for more detail about what is returned. In this case, because the data was generated by 454, we may wish to use the built-in 454 SNP caller (though for best results SNPs should be called with external software and imported with `--snp-caller`).

```
cat pz_p450_CP6B6_pop.tigr | java -jar Hapler.jar --snp-caller 454
```

Because Hapler outputs on standard out, we can easily do things such as filter down to haplotype assemblies (lines starting with `@`) with high coverage, which our experiments show tend to be correct:

```
cat pz_p450_CP6B6_pop.tigr | java -jar Hapler.jar --snp-caller 454
                             | grep -v '^@'
                             | awk '{if($16 > 2) print $0}'
```

Perhaps we are only interested in the minimum-chimerism consensus reconstructions produced (lines starting with `>` with ‘Hapler’ in the third column), and we want to create a fasta file from them:

```
cat pz_p450_CP6B6_pop.tigr | java -jar Hapler.jar --snp-caller 454
                             (run Hapler)
                             | grep '^>'
                             (Filter to consensus reconstructions)
                             | awk '{if($3 == "Hapler") print $0}'
                             (Isolate Hapler Consensuses)
                             | awk '{print $1,$2,$3,$4,$5,$6,$7"\n"$8}'
                             (Print in fasta format)
> pz_hapler.fasta
```

Because Hapler can read on standard input, one can easily run Hapler on contigs in an Amos Bank. For example, if the file `eidlist.txt` contains a number of contig identifiers associated with the Amos Bank `bank`, one can simply run:

```
bank2contig -E eidlist.txt bank | java -jar Hapler.jar --snp-caller 454
```

Or, using command line sub-shells, two specified contigs:

```
bank2contig -E <(echo 7180000019110\\n7180000019111) bank  
| java -jar Hapler.jar --snp-caller 454
```



## 5 Input and Options

Hapler reads multiple alignments in TIGR format (produced by the Amos tools `bank2contig` utility) or SAM format. For examples of these two formats, see Appendix A and B. If you find bugs in inputting these formats, please let us know. Note that Hapler does *not* do the assembly or alignment step for you: you need to produce a TIGR or SAM formatted multiple alignment file (either via de-novo assembly or mapping to a reference).

Hapler assumes that `~` characters in either SAM or TIGR sequences are gaps. Otherwise, Hapler is alphabet agnostic: any other non-whitespace non-`~` character can be used in the multiple alignments (for example, one could “haplotype” protein sequences).

Although TIGR format does not describe mate-pair information (that we are aware of), SAM format may. Because gaps (e.g. mate-pair information) are not compatible with the minimum coloring process (in polynomial time), Hapler will split sequences containing `~` characters into multiple sequences and will ignore all mate pair information. If this happens, a warning is printed to the output.

Hapler takes the following options:

### 5.1 `--input <filename or ->`

The name of the alignment file (SAM or TIGR) to use. If `-` is used (which is the default), Hapler reads on standard input.

### 5.2 `--alignment-type <igr or sam>`

Values can be either `igr` or `sam`, depending on input type. The default is `igr`, note that input type is *not* autodetected.

### 5.3 `--help`

Shows a brief description of Hapler and options.

### 5.4 `--maximize-one-read-haps <true or false>`

As mentioned in Section 8, Hapler samples from minimum colorings of each connected component of the conflict graph. By default, Hapler searches for minimum colorings which also maximize the number of reads that get colors to themselves: we argue that since some reads are likely to contain sequencing errors, we have a better chance of isolating them by searching for haplotypes that can consist of a single read within the parsimony framework of minimum coloring.

This can be turned off by setting this value to false.

### 5.5 `--random-repetitions auto or <integer>`

As mentioned in Section 8, Hapler samples from minimum colorings of each connected component of the conflict graph. By repeating these samples and only putting reads together into the same haplotype if they *always* appear together in all sampled colorings, we can be more sure of the quality of the inference. By increasing the number of samples, we can increase the robustness, usually at the cost of haplotype length.

By default (`auto`), Hapler runs 20 repetitions, and continues running repetitions until either 1) the last 50% of repetitions has not resulted in more haplotype reconstructions, or 2) the maximum number of repetitions is reached (default 100).

### 5.6 `--max-repetitions <integer>`

The maximum number of repetitions to run, even if `random-repetitions` is `auto`, default 100.

### 5.7 `--snp-caller <simple, simplestrict, or 454>`

For best results, one should use the `-snp-list` option, which overrides this one. Nevertheless, Hapler provides several simple SNP calling methods, outlined below:

- **simple**: Any variant locus regardless of allele frequency is treated as a SNP locus.
- **simplestrict**: Any variant locus where a minority allele is present at least twice or is covered by less than 10 reads is treated as a SNP.
- **454**: A variant locus is a SNP if 1) the majority allele is not a ‘–’ and 2) either a) the number of non-‘–’ alleles is at least two, or b) there is any non-‘–’ allele that is not part of a homopolymer run of length  $\geq 3$ .

Because the results are fairly dependent on SNP calling and/or error correction accuracy (both in terms of false positives and false negatives), it would be wise to see section 7.

### 5.8 `--snp-list <filename>`

Rather than use the simple built in SNP callers that Hapler provides, the user can provide a list of (0-indexed) loci that describes positions to use as SNP loci for each multiple alignment.

This is useful when using external SNP callers, such as QualitySNP or PyroBayes.

For an example of the format taken, see Appendix C.

### 5.9 `--ground-truth <filename>`

If the actual full length haplotypes the reads are sourced from are known (as in simulation or testing), they can be provided to Hapler. In this case, all reconstructed and evaluated consensus sequences (Hapler generated, Majority Vote, and Contigs given for evaluation), as well as all assembled haplotype regions, will be evaluated for chimerism and agreement with the ground truth haplotypes. This information will result in extra columns being added to the output.

This option requires a fasta file containing the haplotypes, all of which need to be the same length as the alignment. Note that these haplotypes are used for comparison in all alignments. Thus, this option is only really useful when the input describes a single alignment.

### 5.10 `--evaluate-contigs <filename>`

For each alignment, Hapler can evaluate a given consensus contig (which must be the same length as the alignment) for chimerism. Hapler estimates the minimum number of crossover points needed reconstruct the contig using Hapler assembled haplotype regions (where a crossover point is a pair of SNPs that cannot be supported by a single haplotype region). If the `--ground-truth` option is used, the contig is also evaluated against the ground truth haplotypes.

The input for this option should be a fasta file with sequences corresponding to alignments having the same ID. Contigs must the same length as the alignments they represent.

### 5.11 `--compute-reconstructions <true or false>`

By default, Hapler reconstructs a minimum-chimerism consensus sequence from the haplotype regions and evaluates the chimerism of it against the haplotype regions as well as against ground truth haplotypes (if given). It also reconstructs the majority vote consensus and evaluates the chimerism against Hapler haplotype regions as well as against ground truth. For long alignments with many SNPs, this process can be computationally expensive ( $O(\#SNPs \times \#Haplotypes^2)$ ): this option allows the user to turn it off.

### 5.12 `--human-readable <true or false>`

By default, Hapler outputs haplotype region assemblies in human readable format: aligned against each other and padded with `~` characters, including a line indicating where SNPs occur. For long alignments with many short haplotype regions, this results in the output containing mostly padding, which can take a long time to output and takes up a lot of space. This option removes the padding and the SNP identification line.

### 5.13 `--show-alignments`

Mostly for debugging, this option reports the sections of the multiple alignment contributing to each haplotype reconstruction. Currently not very user-friendly.

### 5.14 `--version`

Output version information and exit.

At the start of every run, hapler outputs comment lines (prefixed with `##`) that describe the further output of Hapler in short form. Hapler also outputs running information to standard out.

Specifically, for each alignment, hapler outputs

1. Warnings, if appropriate, prefixed with **#!**. (These usually refer concerns about SNP calling, for example, if an alignment appears to have no SNPs.)
2. A number of lines corresponding to consensus sequences and their chimerism evaluation, prefixed with **>** (Hapler generated, Majority vote, and provided contigs if the `--evaluate-contigs` option was used).
3. A line describing the positions of called SNPs, prefixed with **#\***
4. A line representing the “universal” haplotype region, containing reads that are consistent with all haplotypes, prefixed with **@@**.
5. A number of lines each corresponding to an assembled haplotype region, prefixed with **@**.

[illegible]

12

## 6.1 Consensus Sequence Output Lines

**Consensus (> prefixed) Column 1 - Unique Consensus Identifier** This column gives a unique identifier for each reconstructed haplotype region, which is merely a concatenation of the alignment name and a code for the consensus type (H for Hapler generated, M for majority vote, C for evaluated contig).

**Consensus (> prefixed) Column 2 - Alignment Name** The name of the alignment this consensus is associated with.

**Consensus (> prefixed) Column 3 - Consensus Type** “Hapler” for Hapler generated consensus, “MajVote” for the majority vote, “Contig” for evaluated contigs given by `--evaluate-contigs`.

**Consensus (> prefixed) Column 4 - Minimum Hapler Crossovers** This column gives the minimum number of crossovers necessary to reconstruct the consensus sequence from Hapler assembled haplotype regions. Here, a crossover is defined as an adjacent pair of SNPs that are not both supported by some haplotype region. Note that Hapler optimizes its consensus sequence according to this (and the following) criterion, so it will necessarily have the lowest value. Also note that any allele in a consensus at a SNP position that is not represented in any haplotype region (i.e., is novel, and is likely an error), is treated as being supported by a “unique” haplotype specific to that locus, and hence incurs two crossovers: one into and one out of the unique haplotype (unless it is at the first or last SNP).

This column thus represents Hapler’s estimate of the “chimerism” of the consensus sequence. Note that because Hapler is conservative in assembling haplotype regions, this number almost always overestimates the true chimerism (unless Hapler is made less conservative by using a very small number for the `--random-repetitions` options).

**Consensus (> prefixed) Column 5 - Maximized SNP Allele Support** After finding the minimum number of crossovers necessary to reconstruct a consensus, Hapler adjusts these crossovers to maximize the total number of bases (in redundant and non-redundant reads) supporting the consensus in haplotype regions. More formally, if haplotype  $s(i)$  is chosen to support a SNP locus  $i$ , and haplotype  $s(i)$  has coverage of  $c(s(i))$  at this locus, this column maximizes  $\sum_{i \in SNPs} c(s(i))$ .

**Consensus (> prefixed) Column 6 - Minimized Unique Haplotype Regions Used** After minimizing chimerism and maximizing support of a consensus, Hapler seeks to minimize the number of unique haplotype regions used to reconstruct a consensus. This tie-breaking criterion will seldom discriminate between two reconstructions in practice.

**Consensus (> prefixed) Column 7 - Hapler Haplotype Regions Used** Hapler optimizes several criteria (above) to determine which haplotype regions should be used at SNP positions to reconstruct a consensus sequence (non-snp positions are assumed to be supported by the majority vote or universal haplotype). This column lists which haplotype regions are chosen, where they start in the alignment (0-indexed), and what their average coverage is.

The start positions in this column are useful for determining where possible crossover points may occur. One may wish to mask these positions when designing PCR primers or microarray probes, for example.

**Consensus (> prefixed) Column 8 - Consensus Sequence** The consensus sequence, as reconstructed from the haplotype regions used (for the Hapler generated consensus), the majority vote (for the MajVote consensus), or as given in the `--evaluate-contigs` option.

**Consensus (> prefixed) Column 9 - Ground Truth Minimized Crossover Number** This column is only output if the `--ground-truth` option is used.

Hapler can evaluate its own consensus (or any consensus) not only against its own assembled haplotype regions, but also against a given set of ground truth sequences. As above, this column represents the minimum number of crossovers necessary to reconstruct the consensus, but out of the ground truth sequences.

This column thus represents the “true” level of chimerism.

**Consensus (> prefixed) Column 10 - Ground Truth Minimized Unique Haplotypes Used** This column is only output if the `--ground-truth` option is used.

First, note that each ground truth haplotype, being a single sequence, has a “coverage” of 1. Thus, the summed support is equal for all reconstructions and this secondary optimization is not done.

Instead, Hapler skips directly to minimizing the number of unique ground truth haplotypes needed to reconstruct the consensus.

**Consensus (> prefixed) Column 11 - Ground Truth Haplotypes Used** This column is only output if the `--ground-truth` option is used.

After optimizing the above criteria to determine which ground truth haplotypes are needed to reconstruct the consensus, Hapler outputs which ones are used as well as their start positions for reference. These are the of “true” (minimum number of) chimerism points.

## 6.2 Haplotype Region Output Lines

**Haplotype (@ prefixed) Column 1 - Unique Haplotype Identifier** This column gives a unique identifier for each assembled haplotype region, which is merely a concatenation of the next three columns.

**Haplotype (@ prefixed) Column 2 - Alignment Name** Each assembled haplotype region is associated with its input multiple alignment name (from the TIGR or SAM file) through this column.

**Haplotype (@ prefixed) Column 3 - Alignment Length** The length of the input alignment (also the length of the output consensus sequences).

**Haplotype (@ prefixed) Column 4 - Haplotype Block Number** Non-variant regions separate connected components in the conflict graph (see Section 8). When there are such non-variant regions (multiple connected components), we have no information to attach haplotypes from one variant region to another. Thus, each haplotype region is associated with a connected component of the conflict graph, or “Haplotype Block,” it belongs to through this column. These are numbered, for each multiple alignment, beginning at 1.

This value is ‘U’ for the “non-variant” Haplotype, which is associated with a unique “universal” block. (But note that this block *can* be associated with all other blocks, as we know that non-variant sequences can be phased with all haplotypes).

**Haplotype (@ prefixed) Column 5 - Minimum Number of Haplotypes Supported By Block** Parsimony suggests that we should minimally color the conflict graph (see Section 8), equivalent to reconstructing a minimum number of haplotype regions supported by the data. However, because we repeat and sample the coloring process, we usually end up with a larger number of haplotype reconstructions. This number, which is the same for all haplotypes in a single block (it’s really associated with the haplotype block, rather than each haplotype region), gives the minimum number that would be constructed by pure parsimony ( i.e., this is the number of haplotype regions that would be created for this block if `--random-repetitions` was set to 1).

**Haplotype (@ prefixed) Column 6 - Number of Repetitions Completed** By default, for each haplotype block, Hapler repeats the coloring process in pseudo-random until no new haplotype regions are reconstructed (see input section `--random-repetitions`). This column outputs the actual number of repetitions used for each haplotype block.

**Haplotype (@ prefixed) Column 7 - Haplotype Region Number** Within each haplotype block, each haplotype region is given a number, counting upward from 1. The “non-variant” haplotype is given the special designator ‘U’.

**Haplotype (@ prefixed) Column 8 - Number of Non-Redundant Sequences** In order to minimally color the conflict graph, Hapler must “mask” redundant reads (those whose covered SNP loci are a subset of some other non-conflicting read). This masking is

done by the reads themselves: if read A makes read B redundant, and read B makes read C redundant, then A must also make C redundant: so A “masks” B and C and these are removed from the haplotyping process.

This column then gives the number of non-redundant (non-masked) reads associated with each haplotype reconstruction. (Note that for the universal haplotype, all sequences are non-redundant.)

**Haplotype (@ prefixed) Column 9 - Number of Redundant Sequences** As a complement to Column 6, we also give the number of redundant reads associated with reads contributing to each haplotype reconstruction.

**Haplotype (@ prefixed) Column 10 - Number of Defined SNPs Covered** A read is said to *cover* a SNP locus if it has a known allele (non-‘ ’ character) there. The number of SNPs defined by a haplotype region is the union of those covered by all reads within it.

**Haplotype (@ prefixed) Column 11 - Number of Defined SNPs Not Covered** The total number of defined SNPs for the alignment minus the number of defined SNPs covered.

**Haplotype (@ prefixed) Column 12 - Start Position of Haplotype Region** This is the 0-indexed position of the first non-‘ ’ character in the haplotype region assembly (i.e., the first position of the first read in the haplotype). If there are no reads in the haplotype region (this can only happen with the universal ‘U’ haplotype region), this will be -1.

**Haplotype (@ prefixed) Column 13 - End Position of Haplotype Region** This is the 0-indexed position of the last non-‘ ’ character in the haplotype region assembly (i.e., the last position of the last read in the haplotype). If there are no reads in the haplotype (this can only happen with the universal ‘U’ haplotype), this will be -1.

**Haplotype (@ prefixed) Column 14 - Length of Haplotype Region** End Position - Start Position + 1, unless no sequences are in the haplotype region (this can only happen with the universal ‘U’ haplotype), in which case 0.

**Haplotype (@ prefixed) Column 15 - Number of Pieces Haplotype Is In** Because of the minimum coloring formulation, it is possible for a haplotype to consist, for example, of only two reads that are far apart in the alignment and don’t actually overlap. If these two always share a coloring, we can be confident of their association, even though there is no “positive” information (shared SNP alleles) linking them (see Section 8).

Pieces represent association not only via coloring but also via positive shared SNP allele information: pieces are defined as connected components in an overlap-at-SNP loci graph within haplotype regions. Thus, for any two reads which are part of the same piece, there is a path over overlapping and agreeing SNP allele reads between them.

Multiple pieces indicate a lack of such positive information linking reads together within a reconstructed haplotype, even though the repeated coloring may suggest they belong to the same haplotype. Usually, however, the repetition of colorings forces haplotypes down to a single piece.



**Haplotype (@ prefixed) Column 16 - Average Sequence Coverage of Haplotype**

The is similar to the “average gapped coverage” used in assembly. We take the total number of bases represented in reads belonging to the haplotype (*including* redundant/masked reads), and divide it by the length of the haplotype (see above). Note that if a haplotype doesn’t cover some internal section (i.e., has internal ‘ ’ characters, which can only happen if it is in multiple pieces), then this value can be below 1.0.

**Haplotype (@ prefixed) Column 17 - Base Coverage at SNP Positions** Summing over SNP loci covered, we sum the number of reads belonging to the haplotype region that cover the locus.

**Haplotype (@ prefixed) Column 18 - Alignment Coverage** The number of non-‘ ’ characters in the haplotype region assembly. Note that this can be different than the haplotype region length (above) if there are ‘ ’ characters in the middle of the haplotype region (which is rare).

**Haplotype (@ prefixed) Column 19 - Haplotype Region Assembly at SNP Positions** This column shows the assembled haplotype regions, with all non-SNP loci removed. It provides a good summary of the haplotype regions assembled and their length/coverage at the important (variant) loci. These are shown aligned for all variant loci: where a haplotype doesn’t determine an allele a ‘ ’ character is used.

**Haplotype (@ prefixed) Column 20 - Full Haplotype Region Assembly** The column shows the assembled haplotype regions, including non-variant loci, all aligned against each other, padded with ‘ ’ characters (unless `--human-readable` is set to false). Where a haplotype isn’t determined, ‘ ’ characters are used.

Consensus bases for each SNP locus are called as the majority vote of reads within each haplotype (which in fact always agree, so this majority vote is unanimous). Consensus bases for all other loci are called as the majority vote of all reads in the multiple alignment: this helps filter out sequencing errors which have not been called as SNP loci, since per-haplotype coverage is often low.

**Haplotype (@ prefixed) Column 21 - Reads In Haplotype** For each reconstructed haplotype, Hapler also reports the reads contributing to it. Read IDs that are prefixed with a ‘+’ are redundant and are masked by the preceding non-‘+’ read. For each read, the start position in the multiple alignment (0-indexed) is also given in parentheses.

**Haplotype (@ prefixed) Column 22 - Ground Truth Sequences Exactly Matched** This column is only displayed when using the `--ground-truth` option.

If a set of ground truth haplotypes is given, then for each reconstructed haplotype region, hapler will count the number of ground truth haplotypes for which it is a *perfect* match, at all loci. Here, ‘ ’ characters in the haplotype region may match any character.

**Haplotype (@ prefixed) Column 23 - Ground Truth Sequence Mismatch Counts**

This column is only displayed when using the `--ground-truth` option.

For each ground truth haplotype, Hapler indicates the number of mismatching bases that keep an assembled haplotype region from being a perfect match. Thus, 0s indicate which ground truth haplotypes are perfect matches to the assembled haplotype region.

## 7 Maximizing Quality

In general, for haplotyping applications, poor data can result in 1) possible sequencing error being included in haplotype assemblies, and 2) chimeric haplotype reconstructions. Hapler uses a repeated, weighted bipartite matching method to try and maximize the robustness of the returned haplotype regions (see Section 8 and [3]): although sequencing errors can still get through (these are often represented as single read haplotypes, and sometimes as longer haplotypes with low coverage) and chimeras can still exist, this usually results in shorter haplotypes being returned, representing uncertainty in the data.

Thus, for best results, one should still consider the quality of data fed to Hapler.

### 7.1 Read Length and Coverage

If no read spans a non-variant region, Hapler will not attempt to build haplotypes across that region. Thus, it is necessary to 1) use long enough read lengths that non-variant regions can be spanned by reads, and 2) have high enough coverage that this does actually happen. An estimation of read length and coverage necessary to do this can be found in [1]. Ideally, one would want high enough coverage that all regions of all haplotypes are covered to some degree. One of the benefits of Hapler is that if such data isn’t available, it will work with what you do have and not return answers that aren’t well-supported.

### 7.2 SNP Calling and Error Correction

Our experiments with Hapler indicate that sequence quality and locating true SNPs is of quite high concern if one wants accurate haplotype reconstructions. Hapler determines errors from non-errors through the calling of SNP-loci: variations at SNP-called loci define the conflict information, while other variations (which are presumably errors) are ignored, though left in the original data.

Notably, both false-positive and false-negative SNP calls can adversely affect haplotyping performance. A false-negative SNP (a true SNP that is mistaken for a sequencing error) means that some genetic variation will be ignored. This can result in a haplotype missing from the output, or a possible chimera because that diversity may have allowed for differentiation that is consequently ignored.

False-positive SNPs (calling errors as SNPs) are troublesome as well—these can be viewed as erroneous, rare haplotypes supported by only a single read. This is variation that Hapler will attempt to isolate through the “maximizing single-read haplotypes” mechanism. Even so, it is possible for these erroneous reads to “stick to” other, correct reads that they share correct alleles with, resulting in longer (though usually low average coverage) haplotypes. Although a coverage-cutoff often identifies these (see [3]), this is a waste of the correct reads that are stuck to the erroneous reads, reducing haplotype information amongst correct reads.

Finally, we note that calling SNPs is general much harder when coverage is low, as near the ends of assembled contigs. Thus, one may want to “trim” alignments so that coverage is high throughout. Further, even if errors are not called as SNPs, if they sit in regions with locally low coverage (e.g., if there is a sequencing error in a locus only covered by two reads), Hapler may use the errors as the base call of the overall alignment, causing the error to be present in all haplotypes spanning that region.

### 7.3 Sampling Number

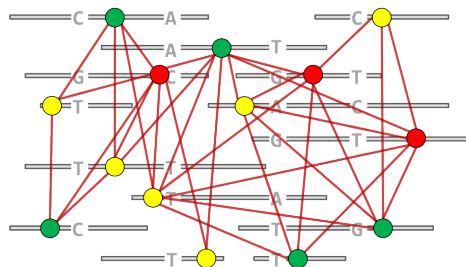
As described in Section 8, Hapler samples in a pseudo-random fashion from minimal coloring solutions, keeping only common haplotype information: an increase in repetitions with the `--random-repetitions` parameter results in higher confidence of the reconstructions. As shown in [3], for a single coloring many long but chimeric haplotypes are created. Increasing the parameter to 5 rapidly decreased the number of such chimeras (at a cost of haplotype reconstruction length), and increasing to 10 further improved the results but at a slower rate, becoming nearly asymptotic.

By default, Hapler runs a minimum of 20 coloring repetitions, and continues running repetitions until the last 50% of them have not increased the number of haplotype regions reconstructed. However, in any case, Hapler will stop when `--max-repetitions` have been run (default 100).

## 8 How it works

The following is a quick, intuitive overview. More details can be found in [3]. We are grateful for the work done by Eriksson et al. ([1]): Hapler expands on the initial graph-theoretical formulation proposed there for increased robustness on low-coverage, low-diversity data.

Haplotyping is done primarily by creating a graph with each read corresponding to a node such that two nodes are connected by an edge if they conflict. If we minimally color this graph, each color consists of nodes that don't conflict and may represent data sourced from the same haplotype. Note that by this formulation reads which don't overlap can be given the same color, thus haplotype reconstructions can return "gaps" in the middle. For example the "red" reads in Figure 2 leave an undefined section between the first and second read. This is infrequent in normal usage: *usually* this indicates uncertainty caused by lack of coverage or genetic diversity which Hapler is designed to avoid (see below).

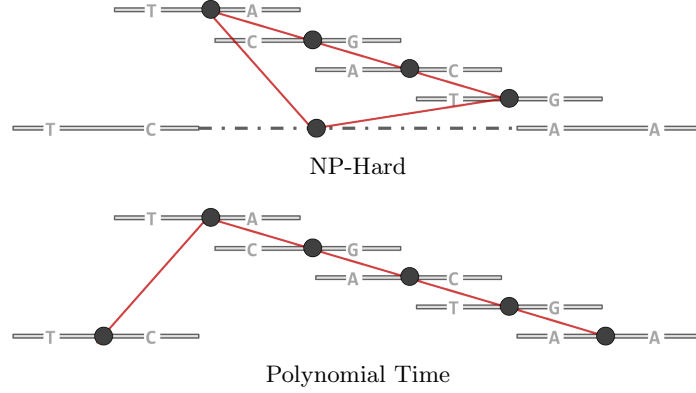


**Fig. 2.** Given an alignment of reads (here showing only positions defined as SNPs), we want to create a conflict graph and minimally color it.

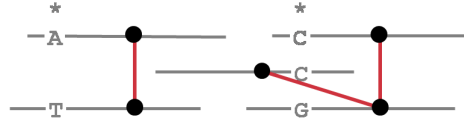
Minimum coloring is generally NP-hard. However, if "gaps" (unknown sequence) are not allowed in the reads (where a gap in a read precludes conflicts at the unknown alleles), then this problem is polynomial time. Note that mate-pairs could be considered as single a single sequence with a large gap in the middle: ideally, we'd like to use a single graph node to represent both ends of the mated read. Unfortunately, to ensure polynomial time and correct answers, we have to use two nodes, thus each end may end up in different haplotype assemblies.

Hapler executes a number of steps, which can be summarized as follows:

1. SNP-call. We may wish to only consider some variations as relevant for haplotyping, to ignore sequencing errors, for example. Hapler can take a list of positions to use as SNPs, or can find its own using a variety of methods (see Section 5).
2. Identify reads that don't conflict with anything. These reads are common to all haplotypes, are collected together into a "universal" haplotype that may have several gaps (where variation occurs).
3. Identify haplotype blocks. If reads are too short to span invariant regions, then there is no way to associate a haplotype segment from one variant region to another. (More formally, we only perform haplotyping on connected components of the conflict graph).
4. Minimally color the haplotype block conflict graphs. Hapler by default finds minimum colorings that also maximize the number of nodes (reads) given colors all to themselves (reads that define a haplotype all by themselves). Since sequences containing errors



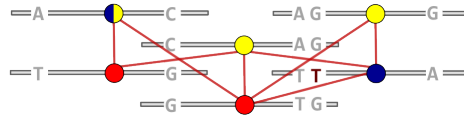
**Fig. 3.** The minimum coloring problem is polynomial time in this case if we ensure that reads are contiguous, without “gaps.” This, however, means that mate-pair information cannot be fully used: we must associate mated reads with different graph nodes, such that they can possibly be associated with different haplotypes.



**Fig. 4.** If there is more than one connected component in the conflict graph, no information exists to associate haplotypes from one connected component to another. Thus, Hapler only reconstructs haplotype regions for reads that are part of the same connected component, which form contiguous “haplotype blocks.” Reads which occur between these and conflict with nothing are part of invariant regions; these are associated with a special “universal” haplotype for each multiple alignment (see Section 6).

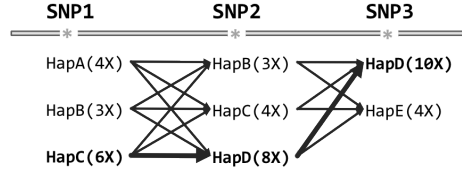
could be viewed as “haplotypes unto themselves,” this is an effort to isolate sequencing error. (See Figure 5).

5. Repeated coloring. Because multiple such colorings (haplotypings) may be possible, Hapler repeats this process a number of times, and only puts two reads into the same haplotype if they always are put together by the colorings. Increasing this parameter increases the confidence of haplotypes, avoiding chimeric reconstructions.



**Fig. 5.** There are many ways to minimally color a conflict graph; Hapler by default finds ways that also maximize the number of reads which are left as colors (haplotypes) unto themselves. In this example, the upper left read may be given two possible colors: Hapler forces the use of yellow. This can be turned off by using `--maximize-one-read-haps false` (see Section 5). Even so, in general multiple solutions may still be possible: Hapler pseudo-randomly samples from these and only infers haplotypes for reads which always appear similarly colored. The number of such samples is controlled with the `--random-repetitions` option.

6. Haplotype region assembly. For each haplotype an “assembly” is created, which is only defined for regions that are covered by reads assigned to the haplotype. Undefined regions are left as ‘ ’. For SNP-called loci, the majority allele within the haplotype (which will



**Fig. 6.** Dynamic programming representation used for reconstructing minimum chimerism consensus sequences. At each SNP locus, the haplotype regions overlapping that locus are identified, and a minimum chimerism, maximum coverage path is identified. Shown are example haplotype regions overlapping for three SNPs, as well as the locus-specific coverages of each haplotype region. The bold path shows the optimal reconstruction; there is one crossover necessary which starts at the locus of SNP2 (HapC to HapD), the total maximized SNP coverage is  $6 + 8 + 10 = 24$ , and two unique haplotype regions are used.

be invariant, i.e. a “unanimous” vote) is used for the call. For Non-SNP-called loci, the majority vote of the alignment as a whole is used—this usually prevents sequencing error from getting into the haplotype reconstructions, since coverage will be high overall but low for each individual haplotype.

7. Minimum-chimerism consensus reconstruction. In most situations, the genetic diversity and coverage is not complete enough to create haplotype regions which span the entire alignment with high confidence. However, we still want a “consensus” sequence that represents the alignment, so we try to build such a sequence that minimizes the possibility of chimerism. One possible way of doing this is by looking at a minimum tiling path over the haplotype regions, however, this is complicated by the existence of gaps in haplotype region assemblies as well existence of the universal haplotype region (crossing into this isn’t a real crossover and shouldn’t be penalized). Thus, we aim to reconstruct the full consensus only at SNP positions, filling in non-SNP loci with the majority vote of the overall alignment.

Given a consensus sequence  $\mathcal{S}$ , we define a “possible crossover” as any two adjacent SNP loci  $i$  and  $j$ , where no haplotype region supports (covers and agrees with)  $\mathcal{S}$  at both  $i$  and  $j$ .

Finding a consensus with minimal crossovers can easily be computed via dynamic programming: for each SNP locus  $i$ , we identify which haplotype regions overlap it. We can then consider pairs of adjacent SNP loci in order, considering each possible configuration of haplotype usages between them, and keeping track of the minimum chimeric path to that point—see Figure 6 for a representation.

In fact, Hapler optimizes the consensus sequence  $\mathcal{S}$  based on several criteria of decreasing importance:

- (a) Minimizing the number of possible crossover points in  $\mathcal{S}$ .
  - (b) Maximizing the total supporting read coverage (both redundant and irredundant) of alleles used at SNP positions in  $\mathcal{S}$ .
  - (c) Minimizing the number of unique haplotype regions used.
8. Evaluate the majority vote consensus and contigs given with `--evaluate-contigs` against haplotype regions. While Hapler can reconstruct a consensus minimizing possible chimerism, it can also evaluate any given consensus by computing the minimum amount of chimerism needed to reconstruct it from the haplotype regions. The process

is the same as above, except here for each SNP we only identify those haplotype regions overlapping *and* agreeing with the consensus before finding the optimal path. Also, in this case if the consensus has an allele at a SNP locus that isn't supported by any haplotype region, a unique "error haplotype" is created specifically to cover that locus, incurring two crossovers (one into and one out of), unless that is the first or last SNP.

9. Evaluate the Hapler consensus, majority vote, and given contigs against ground truth haplotypes. When the `--ground-truth` option is used, Hapler also evaluates all consensus sequences against the ground-truth haplotypes to determining the "true" chimerism. This process is the same as above, except here ground-truth haplotypes are the bases for the path created rather than haplotype regions.



## A Example TIGR Input Format

The TIGR alignment format describes a multiple alignment of short reads against one or more longer consensus sequences. It is output by the Amos tool `bank2contig` when working with an Amos Bank. For example, if the file `eidlist.txt` contains a number of contig identifiers associated with an Amos Bank `bank`, one can simply run:

```
bank2contig -E eidlist.txt bank | java -jar Hapler.jar
```

The TIGR format describes each multiple alignment starting with a `##` line describing each consensus, followed by a number of `#` lines describing the placement of each read. Here is an example:

```
##7180000000490 62 1491 bases, 00000000 checksum.
TCGGCGCTTGC GG- TTTTGGAAATCCATGGATTCTTTGAGCAATGAAATCTGAATTC
AGAAGATTAGGTAAAAGAATATTTCAACGAACGCCTAAAGATGCTTTGCATGCAGCTCTG
AAA- TTATTATTTCCGGAATTATGCAAAAATATTAATTTCTTAGATCCACAACCTGAGAA
ATCTATGACTCTCTTGGTTCAAACGTGTTATGAG- AGATAGAAATTATAAGCCTCGGGTA
AAAACGATTTTATCGATCTCATGTTAGAACTTAGAGAAAAAGGTACAATAATGGTGAAT
CTATTGAGAGTAGAAATAGCGATGGATCTCCAAAAGTAG- TTACATTGGAAATGACAGAC
ATGATAATGATAGCTCAAGTC- TTTG- TATTCTTTGGAGCCGG- TTTGAGACATCATCT
ACAGCATCCAGTTATACATTGCATCAATTAGC- TTTTAACCCAGAATATCAATTAAGT
- G- C- AAGAAGAAATT- GATCAAGT- GTT- AAAAAATAT- GATAATAAAATTACTTATGA
AGCAGTAAATGAAATGACCTATCT- AGAAAAAGCTTTTATGAAGCAATGAGAAATGTACC
CATCGGTAGCCTATATTGTTAGAAATGTACATCACCATAATATACCATTCCTGAAATCG
GTGTT- ACAAATAAACGAAGGT- GT- AAAAGTTATGATTCCAATCCAAGCTATGCATAACG
ATATGAAGTACTTCGAAGAACCGG- AAAAATTC- AACCCGAAAGATTTAATTTGGGAAG
AAAACAATTT- AAG- AATGTGTTTCTACC- TTTTGG- TGAAGGTCCAAGAGCATGTGTCG
GTG- AAAG- AC- TGGGACA- GATGCAAGTCCATGGCAGGACTTGACGAGTTTACAGAAG
- TTTTCCGTGGAACCTGCT- AAATGTAGCTGAGATATCCGAAGCCGAGCCACAGCGA
T- CGTTGCCG- AAAG- TTTTGTAGG- CGGATTG- CCT- CTT- AAAATAAGGAAGAGAGC-
ATC- ATAA- TATTATGTTAAC- AGG- ----- A- T- - TA- - - TTAATACGAAAGT- AATG
CTTAAGTTAAGGTACTTTCTGTTAAAGTAA- TTATAAA- --- TTTTAAATATTTTACT
TG- CC- A- TTGAGACATCG- AG- AATG- TTGACTCCA- GGGTCCCC- TTTTCAACCTTG
GGTCCGACAGGTCCCGGTGTACCCT- GGGGACCAAGTGTCTGTCGTCCCTAAACATT
AATC- AAAATGAGACAATTACATAGTATTGTAATTAA- ACAATACAATATTTATTA- TT
TTTAATAGTTTATTACCTTGACATTCTTATT- CGG- AGG- CGT- G- AAGAAGGAAATAG
TAT- AAGTGTGTT- AAATTATGTTGTGACAAC- AAATTTATAAA- C- TTAACAACATTAC
ATATATTGTGATACATT- GTAAATTAAGAATAAAGATATTTGTATAAATC
#FQVI7FG02F5V01(0) [ ] 239 bases, 00000000 checksum. {1 237} <1 236>
TCGGCGCTTGC GG- TTTTGGAAATCCATGGATTCTTTGAGCAATGAAATCTGAATTC
AGAAGATTAGGTAAAAGAATATTTCAACGAACGCCTAAAGATGCTTTGCATGCAGCTCTG
AAA- TTATTATTTCCGGAATTATGCAAAAATATTAATTTCTTAGATCCACAACCTGAGAA
ATCTATGACTCTCTTGGTTCAAACGTGTTATGAGTAGATAGAAATTATAAGCCTCGGGT
#FQVI7FG02FW4R(0) [RC] 452 bases, 00000000 checksum. {446 1} <1 445>
TCGGCGCTTGC GG- TTTTGGAAATCCATGGATTCTTTGAGCAATGAAATCTGAATTC
AGAAGATTAGGTAAAAGAATATTTCAACGAACGCCTAAAGATGCTTTGCATGCAGCTCTG
AAA- TTATTATTTCCGGAATTATGCAAAAATATTAATTTCTTAGATCCACAACCTGAGAA
ATCTATGACTCTCTTGGTTCAAACGTGTTATGAGTAGATAGAAATTATAAGCCTCGGGT
```

Fig. 7. Screenshot of example TIGR formatted input.

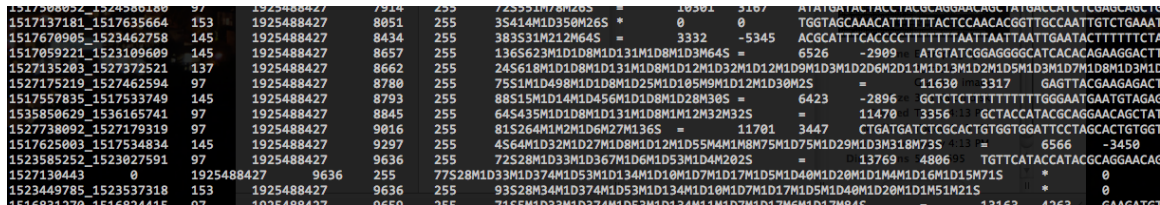
Hapler currently only uses the following information from this format: Contig IDs (for the multiple alignment names, 7180000000490 in the above), Read Names (FQVI7FG02F5V01 etc.), gapped read sequences, and read placement (numbers in ()): 239 and 452 in the above).

## B Example SAM Input Format

SAM is a relatively recent ASCII-formatted description of assembly/alignments. Each line represents a single read (which may be mated with another read described in another line). Formal documentation can be found in [2]. The general form for each line is whitespace separated columns describing:

1. Read Name
2. Flag
3. Scaffold Name
4. Start Position (0 indexed, IIRC)
5. Map Quality
6. CIGAR alignment string (\* if the read is not mapped and should be ignored)
7. Mate Name
8. Mate Position
9. Insert Size
10. Ungapped Sequence
11. Quality Sequence

SAM format specifies that mated reads will be given the *same* name: this is how Hapler determines the existence of mate-pairs. If mate pairs are present, each read of a mate pair will be given a unique name by appending a number to the name: `_0`, `_1`. (If there are multiple reads with the same name, this numbering scheme can continue).



The screenshot displays a portion of a SAM file. Each line represents a read. The fields are: Read Name (e.g., 1517368952\_1524586189), Flag (e.g., 97), Scaffold Name (e.g., 1925488427), Start Position (e.g., 7914), Map Quality (e.g., 255), CIGAR alignment string (e.g., 72S551M1078M20S), Mate Name (e.g., 1517368952\_1524586189\_0), Mate Position (e.g., 18381), Insert Size (e.g., 3187), Ungapped Sequence (e.g., ATATGTACTACTACGAGGAGGAGCTATATGCTCTCTGAGGAGCT), and Quality Sequence (e.g., TGGTAGCAACATTTTTTACTCCAACACGGTTGCCAATTGTCTGAAAT). The sequence and quality fields are truncated in the image.

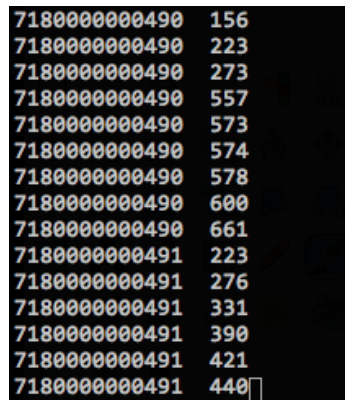
Fig. 8. Screenshot of example SAM formatted input.

Hapler uses the Read Name and Start Position columns when parsing data. For the CIGAR string, the **S** command necessitates a Soft Clip (sequence at the start or end that should be ignored), which is honored by Hapler. Hard clips (**H**) are assumed to have already been done to reads, so Hapler ignores these. Values of **N** are replaced with `~` characters, which will cause sequences to be later split into separate sequences at those positions. Insert characters (**I**) are not allowed: Hapler assumes that the multiple alignment is “gapped.” (If you are creating SAM files with `bank2contig` and run into this problem, try outputting in TIGR format instead). **D** characters, which indicate deletion alleles w.r.t. the reference, cause Hapler to insert `-` characters into the sequence. And of course, match characters (**M**) are used to reconstruct the full gapped sequence from the ungapped sequence.

Hapler ignores Quality information in its current form.

## C Example SNP Input Format

Rather than using it's own internal SNP callers, Hapler can use user-provided SNP call loci. This is described in a simple text file with each row corresponding to a SNPs and the first column representing the name of the alignment the SNP is associated with and the second column being the 0-indexed position of the SNP in the alignment. Note that any further columns in the file will be ignored. Duplicated SNPs and SNP calls which are non-variant in the actual alignment or are outside of the range of the alignment will produce warnings in the output, as will alignments for which no SNPs are called.



```
7180000000490 156
7180000000490 223
7180000000490 273
7180000000490 557
7180000000490 573
7180000000490 574
7180000000490 578
7180000000490 600
7180000000490 661
7180000000491 223
7180000000491 276
7180000000491 331
7180000000491 390
7180000000491 421
7180000000491 440
```

**Fig. 9.** Screenshot of example SNP description format.

## References

1. N. Eriksson, L. Pachter, Y. Mitsuya, S. Rhee, C. Wang, B. Gharizadeh, M. Ronaghi, R.W. Shafer, and N. Beerenwinkel. Viral Population Estimation Using Pyrosequencing. *PLoS Computational Biology*, 4(5):e1000074+, 2008.
2. H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078, 2009.
3. S. T. O’Neil and S. J. Emrich. Robust haplotype reconstruction of eukaryotic read data with Hapler. In *ICCABS ’11: Proceedings of the IEEE 1st International Conference on Computational Advances in Bio and medical Sciences*, pages 141–146, 2011.