

1 HabLab Structural Variants

TODO: number of reads at processing steps

1.1 Description

This project focuses on illuminating the differences between two of the MPIP's mouse models. Model one is the high anxiety group (HAB) the other the low anxiety group (LAB). The data is generated using paired-end sequencing. Thus, in addition to calling single nucleotide variants (SNPs), we are able to call structural variants (SVs) by closely examining the orientation of a pair of reads after the alignment.

1.2 Raw Data

Samples for the sequencing were five (*true?*) animals from each group. DNA was extracted from tail tips (*true?*).

The data for this project was generated by two runs on our SOLiD 4 using two flow cells. The library is termed a Long Mate Pair (LMP) library and entails the sequencing of a forward primer (F3) and a reverse primer (R3). Run one finished on the 26th September 2011, while run two was completed on the 13th October 2011. As the underlying genetic data of the two runs is identical, the data can safely be obtained.

1.3 Work-flow

The raw data can be found at: `/net/PE1/raid1/NGS/LongMatePair_HABLAB/`, the SOLiD, however, does not ensure that R3 file is in sync with the F3 file. Thus, subsequent programs are bound to match the wrong pairs if this is not corrected. For syncing the files we used a custom C program (`/home/altmann/src/C/postprocessColreads/PEcheck.x86_64`), the resulting data can be found at: `/net/PE1/raid1/NGS/LongMatePair_HABLAB_sync/`. The data in this folder should be used as starting point for all paired alignments.

1.3.1 Quality Control (QC)

The first step of the processing work-flow was the quality control step. Here, this step is mainly reduced to the removal of reads with insufficient quality. The definition of insufficient quality in this experiment is a mean quality value (QV) below 10.0. We used a custom C program (`/home/altmann/src/C/postprocessColreads/`

PEqcTrimToDynLen.x86_64) to carry out this step. Moreover, in cases where only one mate of the pair passes the QC, it is written in a separate file and not considered for the paired-end analyses. These “single” reads can be used for other purposes, e.g., SNP calling, where the pairing information is not essential.

1.3.2 Alignment

Due to the color reads produced by the SOLiD platform we were a bit restricted with respect to the available alignment programs. Moreover, due to the large volume of data, we opted against the use of hash-based aligners and embraced BWT-based aligners. In particular, we used bowtie (version 0.12.7) and BWA (version 0.5.7). Bowtie has the advantage of being fast and it successfully pairs a large number of mate pairs. However, bowtie only aligns reads in the correct, expected orientation. Thus, with bowtie aligned read inversions cannot be detected. BWA is a bit slower, and produces less “correct” pairs, but it allows read pairs to be on different chromosomes, in reverse order, etc. In order to exploit both alignment programs, we implemented a sequential alignment scheme.

1. align the read pairs that survived QC with bowtie
2. align read pairs that could not be aligned with bowtie with BWA
3. align “single” reads (mate that passed QC) with BWA

For the aligners following settings were used:

- (a) *bowtie -C -I 200 -X 10000 -ff -k 1 -S -best -l 20 -e 140 -n 3 -p 20 reference-file -1 R3-file -2 F3-file*, while the reference was */net/PE1/raid1/NGS/references/mouse/bowtie/mm9_c*
- (b) *bwa aln -l 25 -t 20 -k 2 -n 4 -c -f out.sai reference infile*, while the reference was */net/PE1/raid1/NGS/references/mouse/bwa/unzipped/mm9_kartyp_col.fa*. Creating a paired end SAM file from the index files (.sai) was done by
- (c) *bwa sampe -a 10000 -n 5 -N 10 -s reference R3.sai F3.sai R3-file F3-file*, with the same reference as above.

Post-processing

The resulting alignments were post-processed, i.e., sorted, adding read group information, merged, etc. For adding the read group info we used the toolkit picard, which allowed in the same step to sort the SAM file with respect to the alignment coordinates. The output was a .bam file (binary SAM file). The sorted .bam files could be merged, first for the different runs, and then for the different alignment steps, resulting in one bam file for HAB and one for LAB. Sometimes, processing was not straight forward. For instance, reads that could not be aligned with BWA were removed from the bam file because they caused problems during the merging step with samtools. This filtering was carried out using samtools: *samtools view -F 4 -b in.bam > out.bam*.

For calling structural variants the bam files were converted into “hit” files (note: no official file type) stating the chromosome, position, insert size, and flag for each aligned read. The hit files were created using the samtools view command in combination with awk: `samtools view -F 4 -q 20 | awk '{ print $3" "$4" "$9" "$2}'`. Meaning, that from the bam file only aligned (-F 4) reads having a mapping quality of 20 or more (-q 20) are printed. The filtering on the mapping quality ensures also that we work only with unique alignments, i.e., alignments that were mapped to a single position in the genome. From the SAM output we use only the 3rd, 4th, 9th, and 2nd column (that is the awk command) corresponding to the information listed above.

The resulting hit file still contains PCR artefacts. We filtered PCR artifacts using a custom C program (`/home/altmann/src/C/CNVfinder/hitsDupRm.x86_64`). As can be seen from the following table, a substantial fraction of the reads were considered PCR duplicates - here defined as reads with the same mapping position and exactly the same insert size length.

group	input reads	output reads	% duplicates
HAB	1,368,358,387	834,196,576	39.0367
LAB	1,389,342,650	676,772,426	51.2883

Based on these cleaned hit files we conducted the structural variant calling.

1.3.3 Structural Variant Calling

There is quite a diversity of structural variants: copy number variants (CNVs), large deletions, large insertions, inversions, etc. For all types of structural variants (except CNVs) we use the information on how the two mate pairs were mapped (details below).

Copy Number Variants (CNVs)

In order to detect CNVs, i.e., regions in the genome of one mouse line that occur in multiple copies in the other mouse line, we compare the coverage, i.e., the number of aligned reads, between the two mouse lines. Changes in coverage may be caused by different copy numbers. For instance, if region A occurs twice in HAB and four times in LAB, then - on average - reads originating from region A are twice the times in LAB compared to HAB. This kind of analysis is often referred to as “depth of coverage” or DOC analysis.

In principle, it is not required to do paired-end sequencing for finding CNVs. However, the additional pairing information, in particular the insert size, helps to improve the coverage estimation. More details below (including an illustration).

Idea outline: the idea of the DOC analysis is simply to compare the coverage achieved in the two mouse lines. In this work we applied following strategy: we first split the whole genome into bins of 200 bp, in a second step the tags (reads) aligned to the bins are counted. This is carried out for each of the mouse lines. At this point, having pair-end sequences, we can improve our coverage substantially by counting also the bins that are covered by a read pair (see Figure 1.1). Here, we made use of this information to achieve a more accurate estimate.

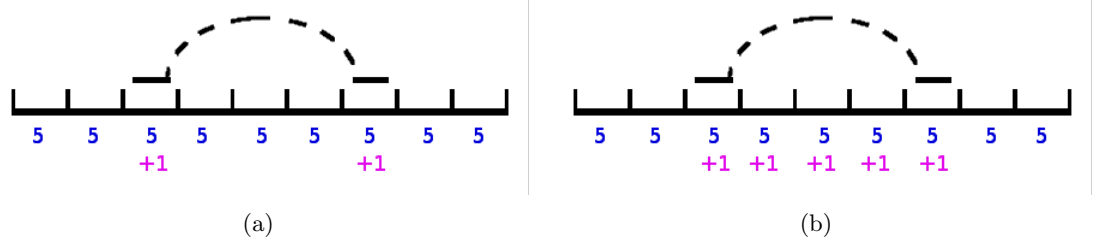


Figure 1.1: (a) Bin count only increased where the tags actually map. (b) Bin count also increased in all intermediate bins

The coverage computation was done by a custom python program (`/home/altmann/src/python/CNVfinder/depth2difference.py`). The program was called using the default parameters (i.e, window size of 200, counting region covered by a read pair, z-score cutoff of 3). Briefly, the tool first computes the mean and standard deviation (sd) of the insert size on the first 200,000 reads in the hits files. The observed values (see below) agree with the 2,000 bp insert size that was aimed at during the preparation of the samples. The z-score cutoff now ensures that regions between read pairs, which are extremely far away, are not counted (scenario (a) in Figure 1.1). Hence, for a read pair with insert size 3,000 bp the counts for the intermediate 15 bins ($3000 / 200$) are increased by 1 (scenario (b)), but for a read pair with an insert size of 6,000 bp the counts for the intermediate bins is not increased as the z-score exceeds the cutoff of 3 (scenario (a)).

group	mean	sd
HAB	2213.74	830.73
LAB	2050.91	850.0

In principle, the tag counts can be used to compute a difference in fold-change in coverage between the two mouse lines. This would hint at a possible presence of CNVs, however, the fold-change does not provide a nice statistic. In order to be more accurate, the tag counts within each bin were used to compute a value of statistical significance indicating the presence of a CNV. The following computations were all computed per chromosome. For each of our bins we counted the reads aligned in HAB and in LAB. Then we also counted the reads that did not align in this bin but on the same chromosome. This yields a 2x2 contingency table for which we can compute a p-value using Fisher's Exact test. The p-value is then converted into a q-value using the Bonferroni-Hochberg method. For simplicity we work with the $-\log_{10}$ of the q-value. This step is carried out by a custom python program (`/home/altmann/src/python/CNVfinder/hitsT0regions.py`). Q-value computation and further processing (i.e., the variant calling itself) was done by the same program. It was split into two steps so as to facilitate to re-use results from the (lengthy) q-value computation step. Q-values were computed using the following options: `-wq -c chr`. Since the test is a one-sided test, only q-values for the enrichment of, e.g., HAB over LAB were computed. In order to obtain q-values

for the other direction, the parameter *-r* has to be added, thus: *-r -wq -c chr*. In all cases *chr* is to be replaced with the chromosome, e.g., *chr5*. Execution generates *.PYqval* files of the following form:

chr7	3000001	39	65	-0.737	3.1169
chr7	3000201	118	175	-0.569	6.6629
chr7	3000401	171	199	-0.219	3.4834
chr7	3000601	203	223	-0.136	3.0312
chr7	3000801	244	251	-0.041	2.4241
chr7	3001001	280	275	0.026	1.9924
chr7	3001201	315	319	-0.018	2.7958
chr7	3001401	350	328	0.094	1.6746
chr7	3001601	374	344	0.121	1.4976
chr7	3001801	402	334	0.267	0.4511

The columns correspond to the chromosome, window start, tag count sample 1 (e.g., HAB), tag count sample 2 (e.g., LAB), $\log_2(\text{fold-change})$, $-\log_{10}(\text{q-value})$. The second step aims at finding “unusual” accumulations of q-values. This is done by screening for sequences of q-values exceeding a q-value limit. Here, we use a rather simple approach: we required all q-values within a window of *w* bins to exceed the initial finding threshold, then the bins left and right to that region were included into the CNV as long as they exceed an extension threshold. Here we used a window size of 8, a finding threshold of 12, and an extension threshold of 10. Hence, the program *hitsTOregions.py* was executed with the parameters: *-c chr -q -w 8 -t 12 -e 10 -v c file.PYqval*, the parameter *-q* tells the program to expect a *.PYqval* file, *chr* is, as above, set to the corresponding chromosome. The output file has following form:

CNV	chr7	3668801	3677601	8800	-2.0827842485	21.5523113636
CNV	chr7	3776801	3782201	5400	-2.14177162224	19.8247703704
CNV	chr7	4137001	4137601	600	-1.31221606668	14.4373333333
CNV	chr7	4366001	4368601	2600	-1.29890837437	14.6799461538
CNV	chr7	4431001	4432801	1800	-1.06777162224	13.7382222222
CNV	chr7	4872801	4873201	400	-1.26921606668	13.9773
CNV	chr7	5093801	5094801	1000	-1.82521606668	22.12238
CNV	chr7	5173201	5174601	1400	-1.46978749525	14.9794142857
CNV	chr7	5176801	5179001	2200	-1.05676152123	13.8987
CNV	chr7	5338201	5340001	1800	-1.76432717779	22.1546

The columns correspond to the type of structural variant (SV), the chromosome, SV start, SV end, SV size, mean fold-change, mean q-value. Thus, now one can filter the results using the q-value and fold-change columns.

Running the programs with the mentioned parameters led to 5,851 detected CNVs. Of these were 2,362 gains in HAB versus LAB, and the remaining 3,489 ones were losses in HAB versus LAB.

Large Deletions

For detecting large deletions we will rely on the same statistical model as with the CNVs. That is: counting tags within bins, computation of p-values, then q-values. And finally finding accumulations of extreme q-values using a window. The difference compared to the DOC CNV analysis was the way we fill the counts in the bins. Instead of counting the tags per bin in HAB and LAB, we counted (within one mouse line only) the pairs that were “regularly” aligned and those ones that showed an significantly increased insert size, which points towards a deletion. To this end we used a custom C program (`/home/altmann/src/C/CNVfinder/tagsCheckCover.x86_64`). The employed settings were as follows: *-f 10 (look for deletions) -z 3.0 (insert length with $z > 3.0$ is considered abnormal) -w 200 (window size) -m (mean as in table above for HAB/LAB) -s (sd as in table above for HAB/LAB)*. This generated again a table featuring chromosome, bin start, normal insert size length (i.e., $z < 3$), abnormal insert size length (i.e., $z > 3$), mean insert size:

chr7	3000001	32	0	1958
chr7	3000201	115	3	2118
chr7	3000401	169	4	2117
chr7	3000601	202	5	2236
chr7	3000801	240	6	2061
chr7	3001001	277	8	2452
chr7	3001201	313	8	2095
chr7	3001401	347	11	2487
chr7	3001601	367	18	2067
chr7	3001801	397	19	2167

On the resulting files - one per group - we applied again the `hitsTOregions.py` program for computing the q-values. Of note, now it was not necessary to compute the “right-sided” p-value, since we were only interested in accumulations of read pairs with abnormal insert size lengths. Hence, `hitsTOregions.py` was executed with the parameters *-wq -c CHR -v d hitsFile*, while `CHR`, again, takes the value of the chromosome to be examined. The result was a “.PYqval” file containing following information:

chr7	6998001	266	50	2245	7.7327
chr7	6998201	255	50	2133	8.265
chr7	6998401	260	50	2040	8.0193
chr7	6998601	283	50	2028	6.9704
chr7	6998801	300	50	2207	6.2739
chr7	6999001	298	50	2161	6.3526
chr7	6999201	304	52	2016	6.8517
chr7	6999401	286	50	2069	6.843
chr7	6999601	287	50	2043	6.8009
chr7	6999801	302	52	2325	6.9336

This is essentially the input file with an additional column: the $-\log_{10}(\text{q-value})$. Using this information, we again screened for accumulations of extreme q-values using the `hitsTOregions.py` program. The parameters were now a bit more extreme than in the CNV finding scenario, because most of the bins will not show any pairs with abnormal insert size lengths. Thus, we set the window size to 12 (corresponding to a minimum deletion length of 2,400 bp, of note: with our insert size distribution we would only be able to detect deletions larger than 4,000 bp), a discovery q-value threshold of 50, and extension q-value threshold of 3. Parameters: `-c CHR -q -w 12 -t 50 -e 3 -v d input.PYqval`. From the deletions detected in this fashion, the insert length was subtracted (2,000 bp currently hard-coded in the python program).

The resulting files present in their columns, the type of structural variant (del), the chromosome, the SV start, the SV end, the length, and the mean q-value (across all the bins contributing to the deletion). The q-values allow filtering and prioritizing deletions in the downstream analysis.

del	chr7	3738001	3755001	17000	38.1285552381
del	chr7	3798801	3807801	9000	17.4038523077
del	chr7	3836001	3845401	9400	54.585219403
del	chr7	3920601	3926801	6200	31.6307215686
del	chr7	3969001	3975801	6800	89.4092481481
del	chr7	4133001	4145401	12400	219.531287805
del	chr7	4178601	4186201	7600	71.1666931034
del	chr7	4228601	4242201	13600	73.6238579545
del	chr7	4340001	4344601	4600	69.0443953488
del	chr7	4407001	4412001	5000	140.423448889

In particular, using this parameters we obtained 8,035 large deletions in HAB and 7,208 large deletions in LAB. Between these deletions 6,451 partially overlapping large deletions were identified. The percentage of the overlap is depicted below (Figure 1.2 (a)). Here, it becomes obvious that the partially overlapping deletions detected in HAB and LAB actually overlap to an essential extent. Approximately 80% of the deletions showed an overlap of 80% or more.

Deletions exclusively detected in HAB were 1,621 and in LAB 782. As can be seen from Figure 1.2 (b) the deletions found only in one line tended to be a bit shorter.

Large insertions

Following the same principles for finding large deletions, i.e., by examining the insert size length, we could also identify large insertions that are manifested by significantly decreased insert size lengths. However, given our insert size lengths with mean 2,000 and sd 850 a significantly decreased insert size length ($z > 3$) equals to: -400. Thus, by just looking at the insert size length, we cannot find large insertions. One possibility, though, is to examine reads where only one mate was aligned. If these cases are enriched, one can assume a nearby insertion, since the unaligned mate could be originally sampled from the insertion (Figure 1.3).

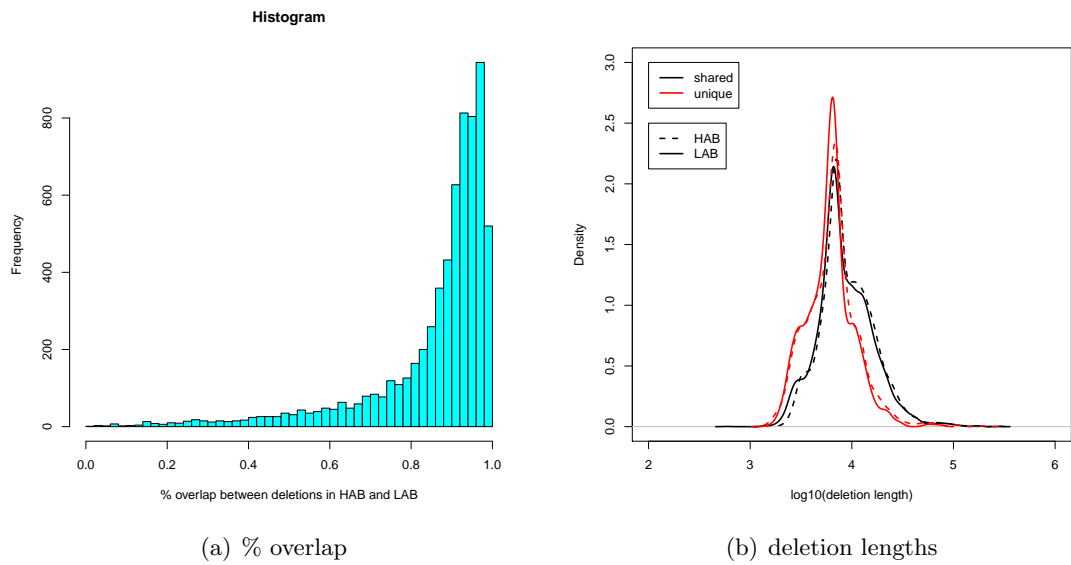


Figure 1.2: (a) Histogram showing the % overlap between the deletions detected in HAB and in LAB. (b) Length of the deletions shared by both lines (black) and unique to each line (red).

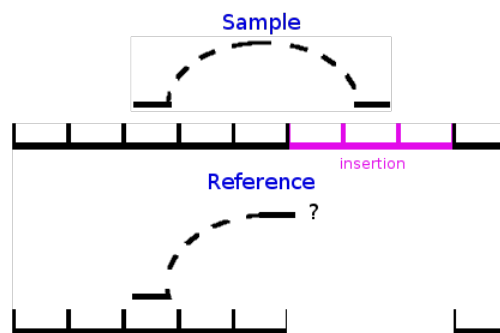


Figure 1.3: Only one mapped mate in case of insertions in the sequenced sample.

Thus, in case of an insertion, in theory the 10 bins (2,000 bp insert divided by 200 bp bins) up- and downstream from the insertion should exhibit an unusually high rate of reads for which one mate was not mapped. Consequently, after computing the q-values with the usual program, we screened for a sequence of 7 bins with q-values above 12 and applied an extension threshold of 3.

In order to obtain the sequence of the inserted region, one can apply methods from *de novo* assembly to the unaligned mates in the target region. Corresponding software tools have been published. However, we did not apply these tools, yet.

The screening for insertions yielded 81 and 36 insertions in HAB and LAB, respectively. Of those, a total of 33 (26) are overlapping (with more than 50%). Of note, the result of this processing gives the “range” in which the elevated q-values were observed. The insertion itself, is more likely to be in the center of the range.

Inversions

For detecting inversions we use the same workflow as for deletions. In the first step (`tagsCheckCover.x86_64`), however, we do not count pairs with abnormally large insert size (`-f 10`) but with the wrong mate direction (`-f 1`). Briefly, typically the R3 and F3 reads should be aligned in the “FF” direction of the “+” strand of the reference or, alternatively, in the “RR” direction on the “-” strand of the reference. Thus, if we observe read pairs that were aligned in a “FR” or “RF” manner something is different between the sequenced genome and the reference. More precisely, these pairs point at genomic inversions.

We follow the same work-flow as in the case of deletions. At the point we obtain the q-values in the .PYqval files we again apply `hitsTOregions.py` with a window size of 8 (`-w 8`), a discovery q-value threshold of 40 (`-t 40`), and extension q-value threshold of 3 (`-e 3`), and we set the SV type to inversion (`-v r`). With these parameters we obtained 169 inversions in the HAB line, and 118 in the LAB line. A total of 87 inversions was shared by the two mouse lines. As in the case of deletions, the overlaps tended to be rather large (Figure 1.4). A total of 82 and 31 detected inversions were exclusive to HAB and LAB, respectively.

Wrong Orientation

Yet another arrangement can be measured. It is a bit more complex and results in the wrong ordering of the reads. That is, usually the R3 is expected to be aligned before the F3 read (on the “+” strand, “-” strand vice versa). Complex structural variants may cause the F3 mate to be aligned before the R3 read (on the “+” strand, “-” strand vice versa). Using the same set of programs we can screen for these events as well. First step: `tagsCheckCover.x86_64` with parameter `-f 4` (wrong mate order) `-w 200` mean and sd as measured above.

After generating the .PYqval files we executed `hitsTOregions.py` with following parameters: `-w 8` (window size), `-t 30` (discovery q-value), `-e 3` (extension q-value). Using these parameters we found 263 and 190 SVs causing wrong mate order in HAB and

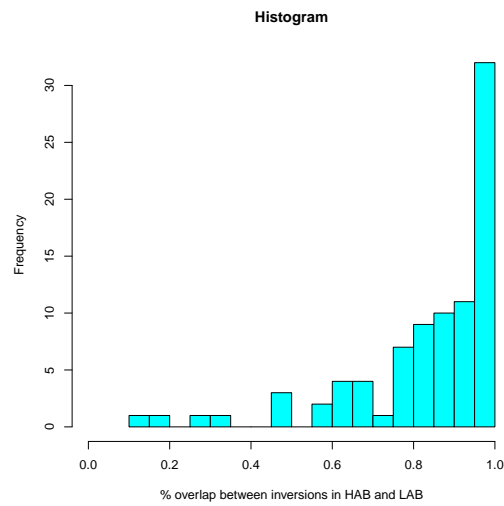


Figure 1.4: Histogram showing the % overlap between the inversions detected in HAB and in LAB.

LAB, respectively. A total of 140 are partially overlapping in HAB and LAB, 135 (144) show an overlap of 50% (85%) or more.

1.4 SNP calling

In progress ...