Tutorial: SNP Calling from Resequencing Data

Bernhard Haubold

February 16, 2012

1 Introduction

In this Tutorial I describe how to analyzed the SNPs contained in resequencing data. My example comes from work with mice, but the general principle ought to apply to any genome you care to investigate. The idea is that you, gentle reader, are sitting in front of a computer and repeat the commands I describe. We proceed in four steps: (i) Map the sequencing reads, (ii) call the SNPs, (iii) deposit the SNPs in a database, (iv) query the database.

2 Mapping

We map the reads using `bowtie` [1]. This rapidly locates a set of sequencing reads on a genome using an index of that genome. The mouse genome index provided on the `bowtie` web page contains the standard headers of the input FASTA file from which it was originally computed. This means that when you want to later interpret map positions, you need to remember that

`gi|149288852|ref|NC_000067.5|NC_000067`

refers to chromosome 1. To simplify matters we rebuild the downloaded index with more convenient names.

2.1 Rebuilding the `bowtie` Index

- Extract the index

```
bowtie-inspect m_musculus_ncbi37 > m_musculus_ncbi37.fasta
```

- Edit the names

```
perl -pe 's/^.+?chr.+? (.+?),.*/>chr$1/;s/^.+?mito.*/>mt/' m_musculus_ncbi37.fasta > tmp
mv tmp m_musculus_ncbi37.fasta
```

- Check the new names

```
grep '^>' m_musculus_ncbi37.fasta
```

- Reconstruct index

```
bowtie-build m_musculus_ncbi37.fasta m_musculus_ncbi3
```

Do not throw away the `fasta` file as we'll need it later at the SNP calling stage.

2.2 Mapping Reads

The important point here is to make sure that the output from the mapping procedure is in SAM format [2]:

```
bowtie -S -p 2 -m 1 m_musculus_ncbi37 sample.fastq > mapSample.sam
```

Apart from SAM output (`-S`), we are using 2 CPUs (`-p`) and only consider reads that map to a single position in the genome (`-m 1`).

3 SNP Calling

3.1 SAM→BAM

BAM is the binary version of SAM and is the starting format for many downstream analyses. But rather than standard BAM, we need sorted BAM, which we get by

```
samtools view -bS mapSample.sam  | samtools sort - mapSample.sorted
```

This generates the file `mapSample.sorted.bam`.

3.2 Inspect BAM File

It is often instructive to take a look at the raw read information.

- Index the BAM file

  ```
  samtools index mapSample.sorted.bam
  ```

- Look at data

  ```
  samtools tview mapSample.sorted.bam
  ```

- To navigate the alignment, enter `g` and then a coordinate, e.g. chr1:123,223,674.
- To exit press `q`

3.3 Variant Calling

Variant calling is carried out by first "piling up" the reads in the sorted BAM file and then submitting the piled up reads to `bcftools`. The pile up step strongly influences downstream results. In particular the "Base Alignment Quality" (BAQ) parameter seems to determine the number and reliability of the SNPs found. I obtained good results using the `-E` parameter, which according to the documentation initiates

> Extended BAQ computation. This option helps sensitivity especially for MNPs, but may hurt specificity a little bit.

The pileup program requires an index of the reference sequence, which we build by

```
samtools faidx m_musculus_ncbi37.fasta
```

Now we can run

```
samtools mpileup -E -C50 -uf m_musculus_ncbi37.fasta mapSample.sorted.bam |
bcftools view -bvgc - |
bcftools view - |
vcfutils.pl varFilter -D100 > var.flt.vcf
```

4 Database Construction

The file `var.flt.vcf` just generated has the format

```
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT mapSample.sorted.bam
chr1 3042482 . T C 16.9 . DP=2;AF1=1;CI95=0.5,1;DP4=0,0,1,1;MQ=46;FQ=-33 GT:PL:GHQ 1/1:48,6,0:4
```

The columns are explained in the first 16 rows of the file:

```
##fileformat=VCFv4.1
##samtoolsVersion=0.1.13 (r926:134)
##INFO=<ID=DP,Number=1,Type=Integer,Description="Raw read depth">
##INFO=<ID=DP4,Number=4,Type=Integer,Description="# high-quality ref-forward bases, ref-reverse, alt-forward and alt-reverse bases">
##INFO=<ID=MQ,Number=1,Type=Integer,Description="Root-mean-square mapping quality of covering reads">
##INFO=<ID=FQ,Number=1,Type=Float,Description="Phred probability that sample chromosomes are not all the same">
##INFO=<ID=AF1,Number=1,Type=Float,Description="Max-likelihood estimate of the site allele frequency of the first ALT allele">
##INFO=<ID=CI95,Number=2,Type=Float,Description="Equal-tail Bayesian credible interval of the site allele frequency at the 95% level">
##INFO=<ID=PV4,Number=4,Type=Float,Description="P-values for strand bias, baseQ bias, mapQ bias and tail distance bias">
##INFO=<ID=INDEL,Number=0,Type=Flag,Description="Indicates that the variant is an INDEL.">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=GL,Number=3,Type=Float,Description="Likelihoods for RR,RA,AA genotypes (R=ref,A=alt)">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="# high-quality bases">
##FORMAT=<ID=SP,Number=1,Type=Integer,Description="Phred-scaled strand bias P-value">
##FORMAT=<ID=PL,Number=-1,Type=Integer,Description="List of Phred-scaled genotype likelihoods, number of values is (#ALT+1)*(#ALT+2)/2">
```

Notice the composite data contained in columns `INFO` and `mapSample.sorted.bam`. Particularly the `INFO` column contains information that we may wish to access later on. The standard solution to this problem would be to split the two composite columns, but unfortunately they don't contain a constant number of items. Nor does the column `ALT`, which may consist of between one and three comma-separated nucleotides. To get around the problem of variable column number we expand every row to the maximum number of columns and fill empty fields wherever they occur with NULL values:

```
awk -f vcf2tab.awk var.flt.vcf > var.txt
```

where `vcf2tab.awk` is listed in Section 6.1. We now generate the corresponding table called `var`

```
mysql -D <dbname> -L < createVar.sql
```

where `createVar.sql` is listed in Section 6.2, and fill it

```
mysql -D <dbname> -L -e "load data local infile './var.txt' replace into table var; show warnings"
```

5 Database Querying

The database can now be queried using commands like

```
mysql -h <host> -D <dbname> -e "select count(pos) from var where chrom like 'chr1'"
```

This returns the number of SNPs on chromosome 1.

6 Code Listings

6.1 `vcf2tab.awk`

```
#!/usr/bin/awk -f
# vcf2tab.awk
# Purpose: Convert polymorphism files in VCF format to a
#   table that can then be loaded into a relational database
# Author: Bernhard Haubold, haubold@evolbio.mpg.de
# Date: 16th February 2012
# License: GNU General Public License
BEGIN{
}{
  if(!/^\#/){
    printf $1;
    for(i=2;i<5;i++)
```

```
      printf "\t" $i;
    # deal with ALT
    nf = split($5,arr,",");
    for(i=1;i<=3;i++)
      if(arr[i])
        printf "\t" arr[i];
      else
        printf "\t\\N";
    for(i=6;i<8;i++)
      printf "\t" $i;
    # deal with INFO
    nfInfo = split($8,arr,";");
    for(i=1;i<=6;i++){
      split(arr[i],a,"=");
      nf = split(a[2],b,",");
      for(j=1;j<=nf;j++)
        if(b[j])
          printf "\t" b[j];
        else
          printf "\t\\N";
    }
    split(arr[7],a,"=");
    split(a[2],b,",");
    for(i=1;i<=4;i++)
      if(b[i])
        printf "\t" b[i];
      else
        printf "\t\\N";
    # deal with last column
    split($10,arr,":");
    printf "\t" arr[1];
    split(arr[2],a,",");
    for(i=1;i<=10;i++)
      if(a[i])
        printf "\t" a[i];
      else
        printf "\t\\N";
    printf "\t" arr[3];
    printf "\n";
  }
}
```

6.2 createVar.sql

```
-- createVar.sql
-- Purpose: Set up table for data generated using the script var2tab
-- Author: Bernhard Haubold, haubold@evolbio.mpg.de
-- Date: 16th February 2012
-- License: GNU General Public License
drop table if exists var;
create table var(
        chrom varchar(5),
        pos int,
        id char(1),
        ref char(1),
```

```
        alt_1 char(1),
        alt_2 char(1),
        alt_3 char(1),
        qual float,
        filter char(1),
        dp int,
        af1 float,
        ci95_1 float,
        ci95_2 float,
        dp4_1 int,
        dp4_2 int,
        dp4_3 int,
        dp4_4 int,
        mq int,
        fq float,
        pv4_1 float,
        pv4_2 float,
        pv4_3 float,
        pv4_4 float,
        gt varchar(3),
        pl_1 int,
        pl_2 int,
        pl_3 int,
        pl_4 int,
        pl_5 int,
        pl_6 int,
        pl_7 int,
        pl_8 int,
        pl_9 int,
        pl_10 int,
        gq int,
        primary key(chrom,pos)
);
```

References

[1] B. Langmead, C Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10:R25, 2009.

[2] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25:2078–2079, 2009.