# Supplementary Information: A practical guide to cancer subclonal reconstruction from DNA sequencing

## Contents

# Introduction

## About this guide

In this guide we will go through the steps of reconstructing a tumour's subclonal architecture (**Figure 1A**). For simplicity, we will be starting from one pair of patient-matched simulated tumour and normal bam files.

These BAM files have been simulated using **BAMSurgeon** to correspond to a pre-designed tumour phylogeny shown on **Figure 1B**. Knowing the true mutations, copy number aberrations and underlying subclonal structure will help illustrate and give maximal insight into the concepts at each step of the pipeline.

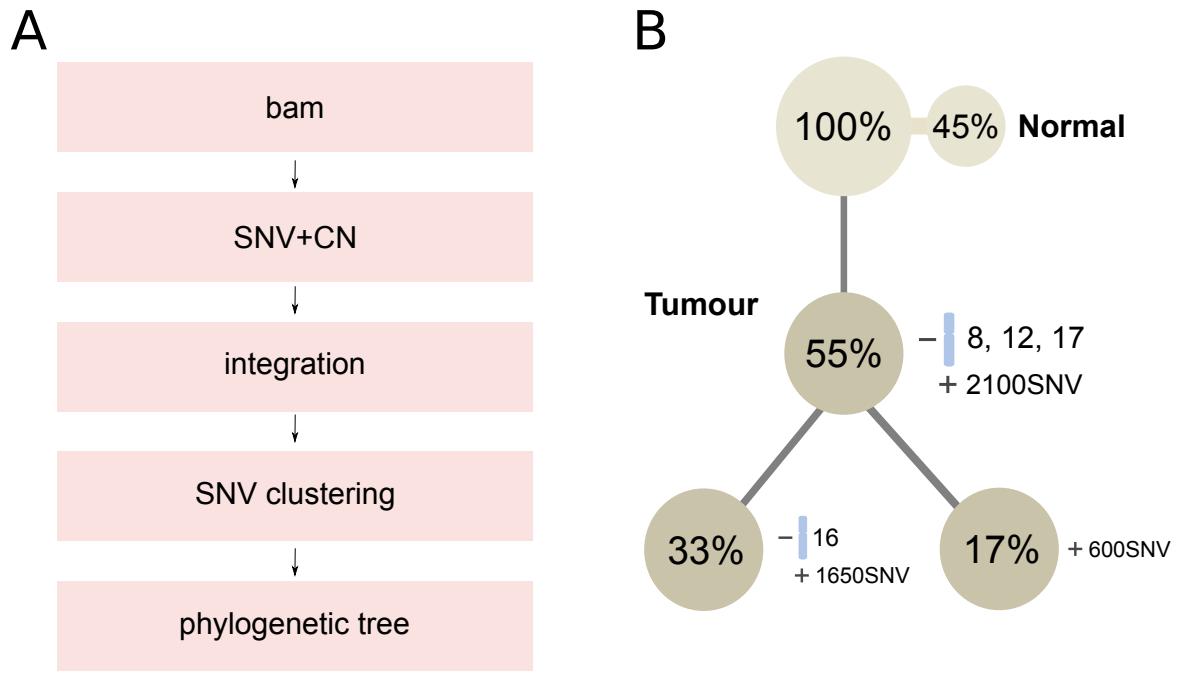


**Figure 1.** A. Pipeline outline. We will go through each step in this guide. B. Simulated tumour phylogeny. A few thousand SNVs, clonal loss of 3 chromosomes and subclonal loss of one chromosome. The simulated BAM, vcf and copy number outputs are available for download (see next section).

For each step of the subclonal reconstruction, we have chosen one tool for illustration purposes (alignment with BWA-mem, mutation calling with MuTect, copy-number calling with Battenberg and clustering with DPClust). However, the choice of the methods should be carefully considered by the user. Indeed, different methods will have different behaviours and might be better suited for the type of samples or project at hand. In the main text, we only recommend methods with specific characteristics, e.g. take into account copy number, Beta or Binomial models, picking one/the best method is not the subject of this guide.

Although we will cover some of the theoretical aspects of the different methods and subclonal reconstruction in this practical guide, we recommend a more in-depth book chapter on the principles of subclonal reconstruction by Dentro, Wedge, and Van Loo[1].

## Availability

This guide is also available on github: **https://github.com/galder-max/src_guide** (where SRC stands for 'subclonal reconstruction'). The data used here is available online, see next section for details. For each method, we provide a hyperlink to their online webpage.

# Download input data

Note: There is no need to start from the BAM file. If you would like to skip some of the steps upstream of the clustering and phylogeny inference, all the input files are already available online on synapse (see following section).

## Input BAM file

The input BAM files were generated as part of another project[2] and can be downloaded from EGA under study accession no. **EGAD00001003971**.

## VCF files

It will be less time consuming to start from vcf files. Mutation calling has already been performed with four different callers on these BAMs. In this guide we will use the "perfect" or "true" calls, i.e. the mutations as input to the simulator, and compare them to the MuTect calls. All vcf files are available on **synapse**. The vcf files accession number is syn21609786. Here we will be using T2 at 128X, the file is named *perfect_T2.T.128X_noXY.vcf*. We will compare the results with results obtained on downsampled BAM files at half the depth, i.e. 64X. These files are also available in the same tarball, together with 32X, 16X and 8X.

## Copy number aberration calling

CNA calling has been performed using **Battenberg**[3]. The output files can be downloaded in synapse with accession number syn21609785. Again, we are using T2 at 128X. The files are named *T2-128X_refit_cellularity_ploidy.txt* and *T2-128X_refit_subclones_noXY.txt*. The former contains the purity and ploidy values, while the latter contains the per segment copy number calls. Battenberg outputs for downsampled BAM files are also available at 64X, 32X, 16X and 8X.

# Step-by-step guide

## Step 1: Alignment with BWA-mem

We will assume that most quality checks have been performed and the first step after receiving your sequences fresh from the sequencing facility, is to align the fastq files to the reference genome. Here we have used **hg19** and aligned with **BWA-mem**[4].

You might also want to consider read trimming and further processing of the aligned BAM for variant calling such as realignment around indels and base recalibration. Mutation callers will usually include these steps in their pipeline documentation.

## Step 2: Mutation calling with MuTect

Mutation calling is a complex classification task where the whole tumour BAM file is compared to the normal BAM file, differences are picked up and characterised, and finally each difference has to be classified as either a true somatic change or a technical artefact.

For the clustering and subclonal reconstruction, we are starting from the substitutions, or single nucleotide variants (SNVs). Many algorithms have been proposed to call SNVs. Some perform better than others but the current good practice is to combine multiple callers into a consensus[5], typically with a majority vote strategy, e.g. considering true variant if called by 2 out of 3 callers.

### Typical run with MuTect

Here, to illustrate a typical run, we show how a previous version of MuTect v1.1.4 was run to generate the **MuTect**[6] vcf files.

```
print(cmd)
```

```
## [1] "java -Xmx8g -jar muTect-1.1.4.jar -T MuTect --tumor_lod 10 --tumor_sample_name T2
--normal_sample_name HG002 --reference_sequence hs37d5.fa --cosmic cosmic.header.vcf
--dbsnp dbsnp.nochr.vcf --input_file:normal HG002.N.bam --input_file:tumor T2.T.128X.bam
--out outFile.out --vcf outVcf.vcf --coverage_file outCoverage.wig"
```

On new data, MuTect2, which also allows to run on multiple samples, should be preferred over MuTect.

### False positives at low VAF

Now, let's load the MuTect and perfect vcf files into R and compare them. We will compare the mutation calls as well as the distribution of variant allele frequencies (VAF), i.e. fraction of mutated reads.

We will use a simple read.table function for minimal dependencies, but more sophisticated read functions are available for vcf files (e.g. *readVcf* from package *VariantAnnotation*).

```
vcf_perfect <- read.table("input/perfect_T2.T.128X_noXY.vcf",sep="\t")
head(vcf_perfect)
```

```
##   V1      V2 V3 V4 V5  V6   V7                                      V8 V9
## 1  1  556789  .  T  A 100 PASS   SOMATIC;VAF=0.118279569892473;DPR=93 AD
## 2  1  763107  .  C  T 100 PASS               SOMATIC;VAF=0.25;DPR=180 AD
## 3  1  946288  .  C  T 100 PASS  SOMATIC;VAF=0.241610738255034;DPR=149 AD
## 4  1 1358216  .  C  T 100 PASS  SOMATIC;VAF=0.291666666666667;DPR=144 AD
## 5  1 1375339  .  C  T 100 PASS  SOMATIC;VAF=0.280701754385965;DPR=114 AD
## 6  1 2801021  .  T  C 100 PASS SOMATIC;VAF=0.0814814814814815;DPR=135 AD
##      V10
## 1  82,11
## 2 134,45
## 3 110,36
```

```
## 4 101,42
## 5  81,32
## 6 124,11
```

```
vcf_mutect <- read.table("input/mutect_T2.T.128X_noXY.vcf",sep="\t")
head(vcf_mutect)
```

```
##   V1      V2          V3 V4 V5 V6   V7                V8                V9
## 1  1  763080 rs189703925  G  T  . PASS DB;SOMATIC;VT=SNP GT:AD:BQ:DP:FA:SS
## 2  1  763107           .  C  T  . PASS    SOMATIC;VT=SNP GT:AD:BQ:DP:FA:SS
## 3  1  831651  rs28864374  A  G  . PASS DB;SOMATIC;VT=SNP GT:AD:BQ:DP:FA:SS
## 4  1  946288           .  C  T  . PASS    SOMATIC;VT=SNP GT:AD:BQ:DP:FA:SS
## 5  1 1358216           .  C  T  . PASS    SOMATIC;VT=SNP GT:AD:BQ:DP:FA:SS
## 6  1 1375339           .  C  T  . PASS    SOMATIC;VT=SNP GT:AD:BQ:DP:FA:SS
##                         V10                         V11
## 1       0:80,1:.:81:0.012:0  0/1:150,7:38:157:0.045:2
## 2        0:89,0:.:89:0.00:0 0/1:130,45:32:175:0.257:2
## 3 0:101,1:.:102:9.804e-03:0  0/1:148,7:37:155:0.045:2
## 4       0:98,0:.:100:0.00:0  0/1:98,30:31:128:0.234:2
## 5        0:75,0:.:75:0.00:0  0/1:99,41:34:140:0.293:2
## 6        0:96,0:.:96:0.00:0  0/1:78,32:34:110:0.291:2
```

As we can see, some of the fields look similar, and are the usual vcf fields, while others are formatted differently. This is documented in the vcf header.

First we look at the chromosomal positions and the associated base changes and compare them between the two set of calls.

```
mut_perfect <- do.call(function(...) paste(...,sep=":"),
                       lapply(c(1,2,4,5),function(x) vcf_perfect[,x]))
mut_mutect <- do.call(function(...) paste(...,sep=":"),
                      lapply(c(1,2,4,5),function(x) vcf_mutect[,x]))
list(count_perfect=length(mut_perfect),
     count_mutect=length(mut_mutect),
     count_intersect=length(intersect(mut_perfect,mut_mutect)))
```

```
## $count_perfect
## [1] 4062
##
## $count_mutect
## [1] 4374
##
## $count_intersect
## [1] 3827
```

So we see that MuTect has missed a few mutations while calling a few extra ones as well. Typically, these false positives display low VAF, so let's verify this.

The perfect calls store the tumour VAFs in the 8th column, while MuTect calls store the tumour VAFs in the 11th column (named *tumor*) in the FA field. There are many ways to go about extracting this information. Here, we will use simple regular expression searches.

```
vaf_perfect <- as.numeric(gsub("(.*);VAF=(.*);DPR=(.*)","\\2",vcf_perfect[,8]))
vaf_mutect <- as.numeric(gsub("(.*):(.*):(.*):(.*):(.*):(.*)","\\5",vcf_mutect[,11]))
```

```
head(vaf_perfect)
```

```
## [1] 0.11827957 0.25000000 0.24161074 0.29166667 0.28070175 0.08148148
```

We can then plot the histograms of VAF and compare MuTect to simulated (**Figure 2**).

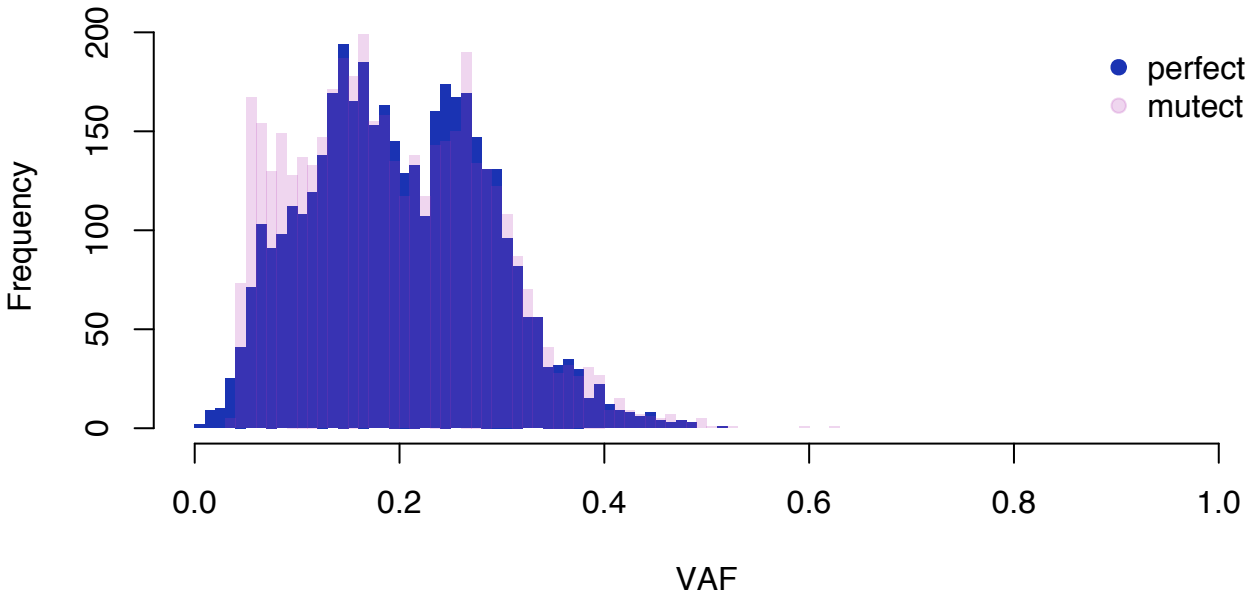


**Figure 2.** Variant allele frequency distributions: MuTect vs. simulated

As expected, MuTect has called extra mutations at low allele frequencies.

### Effect of read depth on the VAF distribution

One can downsample a BAM file, i.e. keep only a fraction of the reads and these need to be randomly sampled. SAMtools has a function for this that makes this really easy, see *samtools view -s*. Here, we will use the pre-computed vcf files on the downsampled BAM file. We load the vcf derived at 32X and the vcf derived at 128X and compare their VAF distribution (**Figure 3**).

```
vcf_perfect_32X <- read.table("input/perfect_T2.T.32X_noXY.vcf",sep="\t")
vaf_perfect_32X <- as.numeric(gsub("(.*);VAF=(.*);DPR=(.*)","\\2",vcf_perfect_32X[,8]))
```

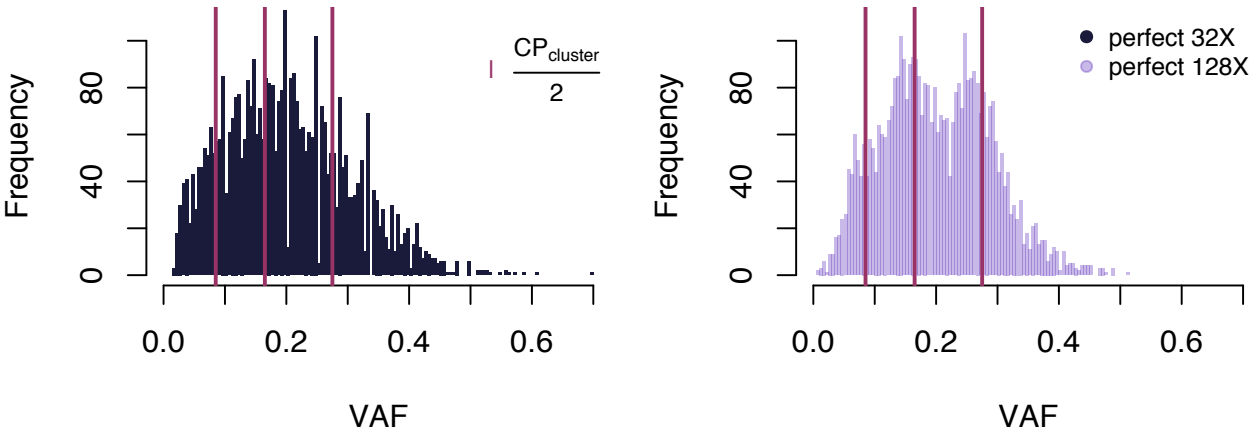


**Figure 3.** Variant allele frequency distributions: Simulated 32X vs. simulated 128X. Flattening of the VAF distribution at lower depth. Vertical lines show the cellular prevalence of the subclones divided by 2. That is where the VAF peak is expected for mutation falling in diploid regions.

## Step 3: Copy number calling with Battenberg

We have run **Battenberg**[3] to call copy number aberrations and flag subclonal segments. If you would like to run it as well, please follow the documentation on the **github page** to install and run it.

### How does it work

The Battenberg algorithm is based on the code and equations of ASCAT[7] to simultaneously derive purity and ploidy of tumour samples and call allele-specific clonal CNAs.

Briefly, to infer ploidy $\psi$ and purity $\rho$ of the samples, ASCAT looks at how the logR $r_i$ and the B allele frequency (BAF) $b_i$ of heterozygous SNPs are influenced by copy-number-induced allelic imbalances across the genome; as a consequence ASCAT will fail to report estimates for cancer genomes presenting no allelic imbalances. In most malignancies, apart from benign or less advanced tumours, we expect to observe allelic imbalances.

Then to call CNAs given a sample's purity and ploidy, ASCAT solves a system of two equations with two unknowns formed by writing LogR and BAF of heterozygous SNPs as a function of the number of copies of allele A, $n_A$ and number of copies of allele B, $n_B$:

$$\begin{aligned} r_i &= log_2\left(\frac{2(1-\rho)+\rho(n_{A,i}+n_{B,i})}{\psi}\right) \\ b_i &= \frac{1-\rho+\rho n_{B,i}}{2-2\rho+\rho(n_{A,i}+n_{B,i})} \end{aligned}$$

Where $\rho$ is the purity and $\psi$ is the average tissue ploidy.

To remove noise, the BAF track is segmented using piecewise constant fitting[7].

A grid search is then used to find the pair of realistic purity and ploidy values that minimises the genome-wide error between observed and expected $n_A$ and $n_B$ values. The expected $n_A$ and $n_B$ are approximated by rounding the observed $n_A$ and $n_B$ to the closest integers.

Battenberg runs this clonal version of ASCAT and then infers subclonal regions by looking at significant deviations of BAF of heterozygous SNPs from the expected BAF of clonal states. If deviations are significant, Battenberg flags the segment as subclonal. However, the subclonal states are not trivially obtained from the logR and the BAF, as there are now more unknowns than equations.

Using heuristics, Battenberg outputs a few possible solutions, i.e. up to 6 pairs of integer copy number states and their associated cancer cell fractions. For example, 1+1 in 70% cancer cells and 2+1 in 30% cancer cells would translate into: 70% of cancer cells carrying 1 copy of each allele, and 30% of cancer cells carrying 2 copies of the major allele and 1 copy of the minor allele. These pairs of states are chosen to be mixed states differing by one or two losses or gains from the closest clonal state that would correspond to the observed logR and BAF. The states are taken as combinations of integer states. Once the two mixed states are fixed, the cancer cell fractions (CCFs) ($\tau$ and $1-\tau$) of these states $(n_{A,1}+n_{B,1})$ & $(n_{A,2}+n_{B,2})$ can be inferred from observed average BAF across the genomic segment $b_s$:

$$\tau = \frac{1-\rho+\rho n_{B,2}-2b_s(1-\rho)-b_s\rho(n_{A,2}+n_{B,2})}{b_s\rho(n_{A,1}+n_{B,1})-b_s\rho(n_{A,2}+n_{B,2})-\rho n_{B,1}+\rho n_{B,2}}$$

Furthermore, Battenberg uses haplotyped SNPs to increase its sensitivity to detect the BAF deviations from expected BAF of clonal states. The haplotypes are imputed using impute2[8] and a two-step segmentation strategy of the BAF track helps take advantage of these imputed haplotypes. Briefly, haplotyped SNPs translate into phased BAF values, but also include switching errors. Therefore first the phased BAF track is segmented using a very sensitive segmentation. This identifies haplotypes and longer phased blocks. Then BAF values are mirrored per segment to have a segment average above 0.5, i.e. if(average(BAF)<0.5) BAF=1-BAF. Finally, a more specific segmentation is run to obtain the final segments.

The last step is to go through each segment's BAF and LogR to call the segment state(s), as described above.

### How does it look

First we note that the Battenberg profile is named *refit*. This indicates that the default fit was not a good one and manual inspection and refitting were necessary. Below we show the default fit (**Figure 4**) found by Battenberg and the refitted profile (**Figure 5**).

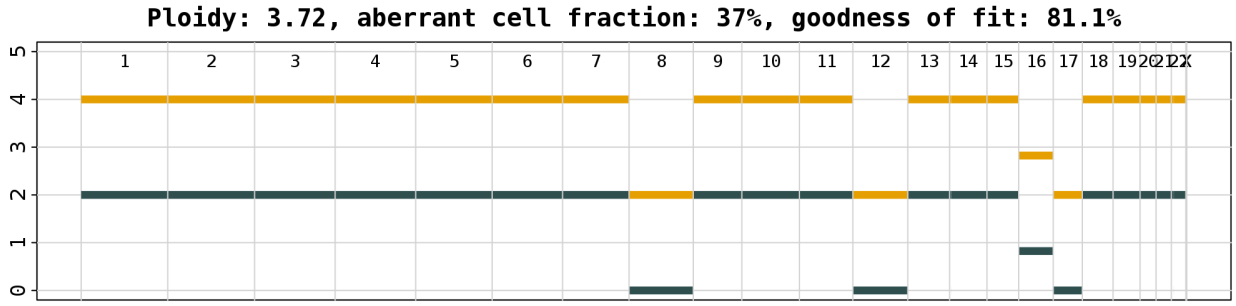


**Figure 4. Battenberg default profile.** Along the genome on the x-axis, copy number of the minor (dark blue) and total (orange) per chromosome.

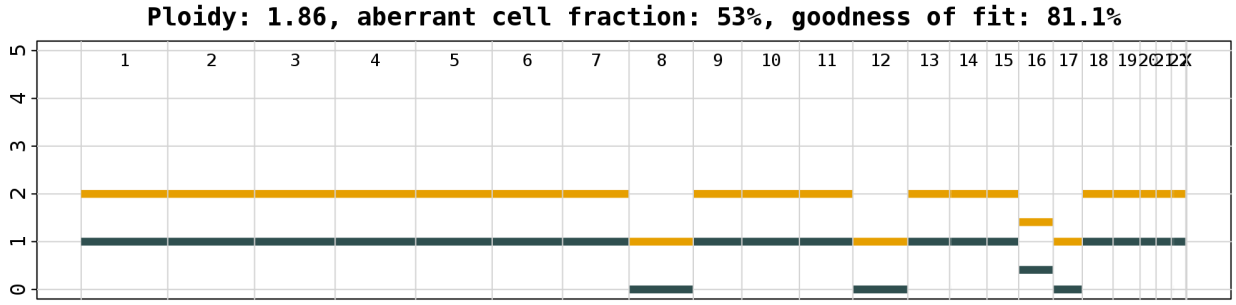


**Figure 5. Battenberg refitted profile.** Along the genome on the x-axis, copy number of the minor (dark blue) and total (orange) per chromosome.

This is a good illustration of the ambiguity of *in silico* ploidy estimates. The default fit proposes twice the actual ploidy. This is likely because of the subclonal loss on chromosome 16 which is almost at 50% of the main clone CP, which brings this segment closer to an integer in the whole-genome duplicated fit.

After the manual refit, the ploidy and the purity better match the simulation design.

The previous figures are generated by Battenberg. We can also derive them ourselves from the copy number profile. Now we load the Battenberg CNA profile - we only keep the first thirteen columns - which include only one of the possible solutions for subclonal segments (the most likely one). Then we plot the genome-wide profile.

```
cna_bb <- read.table("input/T2-128X_refit_subclones_noXY.txt",sep="\t",header=T)[,1:13]
head(cna_bb)
```

```
##   chr startpos    endpos       BAF pval       LogR     ntot nMaj1_A
## 1   1   811735 249223191 0.5013481    1 0.04334820 2.008582       1
## 2   2    23368 242985695 0.5011866    1 0.04252642 2.006444       1
## 3   3    60596 197846280 0.5011744    1 0.04301515 2.007715       1
## 4   4   109541 190781263 0.5010340    1 0.04298700 2.007642       1
## 5   5   859234 180742812 0.5013670    1 0.04378574 2.009720       1
## 6   6   203397 170898626 0.5012337    1 0.04301005 2.007702       1
```

```
##   nMin1_A frac1_A nMaj2_A nMin2_A frac2_A
## 1       1       1      NA      NA      NA
## 2       1       1      NA      NA      NA
## 3       1       1      NA      NA      NA
## 4       1       1      NA      NA      NA
## 5       1       1      NA      NA      NA
## 6       1       1      NA      NA      NA
```

The columns indicate the chromosome, start, end of the segments, their average BAF, p-value of being subclonal (i.e. non-integer given the purity and ploidy values), the LogR, the non-rounded total copies, the number of copies of the major allele, and the minor allele and *frac*, the fraction of cells in which these states are present. If the segment is detected as subclonal, this fraction is lower than 1 and a second copy number state is given together with the fraction of cancer cells in which it is present (1-*frac*). There is too much ambiguity in those states, and while these states are not consistent across CNA methods, CNA methods usually agree on whether the segment should be deemed subclonal.

Now, let's visualise the total and minor copies along the genome (**Figure 6**).

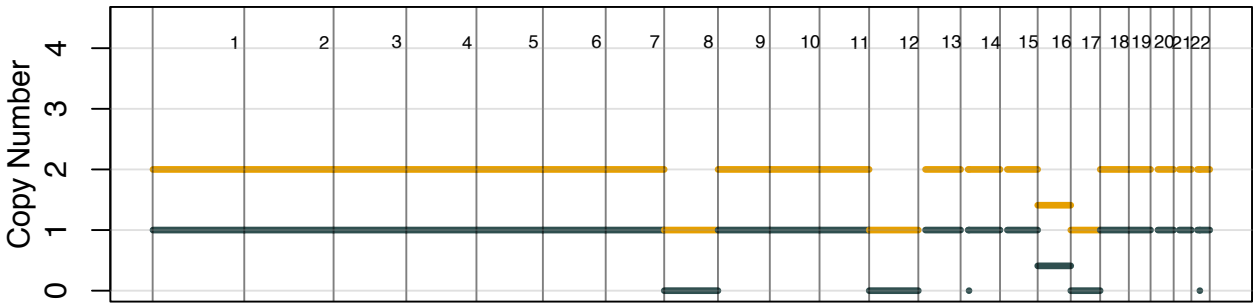


**Figure 6.** Battenberg plot from the input file. Along the genome on the x-axis, copy number of the minor (dark blue) and total (orange) per chromosome.

Importantly, this figure is showing genomic position on the x-axis while the Battenberg figures are showing SNP index. The latter is very useful to visualise relative number of SNPs per chromosome and identify characteristics of the run, such as long stretches of homozygosity, often seen in inbred population. These stretches are blindspots for segmentation with Battenberg, which is based solely on the BAF of heterozygous SNPs.

### Impact of read depth on CNA

The VAF of a given SNV is influenced by the tumour purity and its number of bearing copies. The noise on the VAF estimate is pretty much directly defined by the read depth, which can be a limiting factor.

For copy number, the length of the implicated segment comes into play as well. Large copy number events can be picked up in single cells with coverage as low as 0.1X.

So far, there has been no consistent benchmarking of methods for copy number calling. We only know that they agree reasonably well for single-sample bulk whole-genome sequencing around 60X[9].

## Step 4: Subclonal reconstruction with DPClust

### VAF to CCF

When clustering mutations, the assumption is that mutations present in the same fraction of cells are more likely to belong to the same most recent common ancestor of a large population of cells, i.e. a subclone. Therefore, we need to estimate in what fraction of cells these mutations are present, by deriving the cellular prevalence (CP) or cancer cell fraction (CCF), with

$$CCF = \frac{CP}{purity}$$

The CP or CCF is inferred from the VAF, which is influenced by the number of tumour DNA copies bearing the mutation, or "multiplicity" and the purity. While the purity is a global variable inferred mostly through CNA calling and can be further refined with SNV, the multiplicity is inferred on a per-mutation basis.

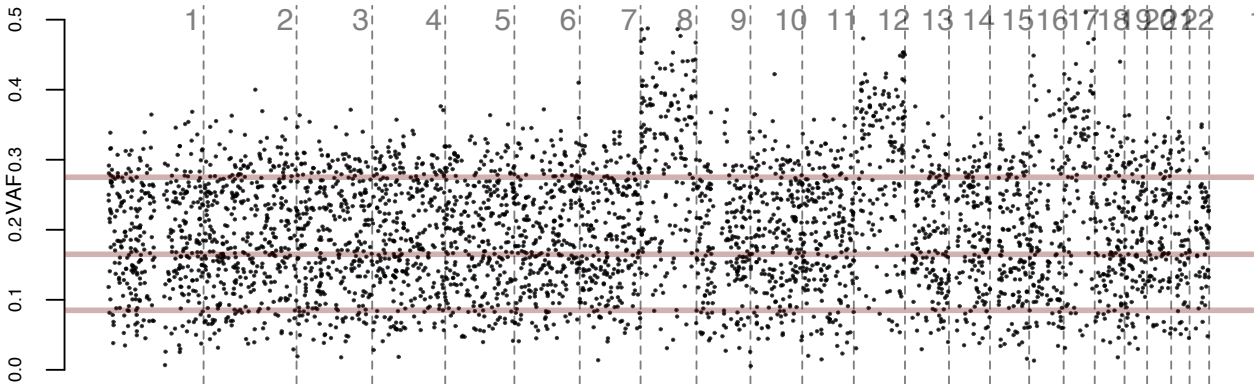


**Figure 7.** Illustrate the effect of copy number on mutation VAF

On **Figure 7**, we see that for the lost chromosomes, because there are now relatively more copies of sequenced DNA bearing the mutations falling in those regions, the VAF is higher.

The mutation copy number of a given SNV is computed as:

$$mutcopynumber = \frac{VAF}{\rho}(\rho N_{tumour} + (1-\rho)N_{normal})$$

Where $N_{tumour}$ is the number of DNA copies in the tumour. In our sample, $\rho$ equals 0.55. Let's look at mutations in 1+0 tumour regions (1 copy of the major, 0 copies of the minor). If their VAF is around 0.4 as in the figure, their mutation copy number is

$$\frac{0.4}{0.55}(1 \times 0.55 + 2 \times 0.45) \approx 1$$

The mutation copy number is an estimate of the multiplicity, which is an integer representing the number of DNA copies bearing the mutation. Dividing the mutation copy number by the multiplicity gives the CCF:

$$CCF = \frac{mutcopynumber}{multiplicity}$$

Most clustering methods infer the multiplicity and then the CCF, and for this, DPClust uses a pre-processing package dpclust3p.

We will now run the main function of this package on our data.

```
lociFile  <- "loci.txt"
vcfFile <- "input/perfect_T2.T.128X_noXY.vcf"
subcloneFile <- "input/T2-128X_refit_subclones_noXY.txt"
alleleCountsFile <- "alleleCountsFile.txt"
cellFile <- "input/T2-128X_refit_cellularity_ploidy.txt"
rhoPsiFile <- "rhoPsiFile.txt"
rhopsip <- read.table(cellFile,header=T)
dpInputFile <- "dpInput.txt"
writeRhoPsi(rhopsip[1],rhopsip[2],rhopsip[3])
createAlleleCountsFile(vcfFile=vcfFile, isperfect=T, outfile=alleleCountsFile)
suppressPackageStartupMessages(library(GenomicRanges))
suppressPackageStartupMessages(library(dpclust3p))
```

```
suppressWarnings(runGetDirichletProcessInfo(loci_file=lociFile,
                                            allele_frequencies_file=alleleCountsFile,
                                            cellularity_file=rhoPsiFile,
                                            subclone_file=subcloneFile,
                                            gender="male",
                                            SNP.phase.file="NA",
                                            mut.phase.file="NA",
                                            output_file=dpInputFile))
```

Let's have a look at the results, especially the mutation copy number vs. the copy number (**Figure 8**). In this sample, the multiplicities cannot be greater than 1, because there are no gained copy number segments.

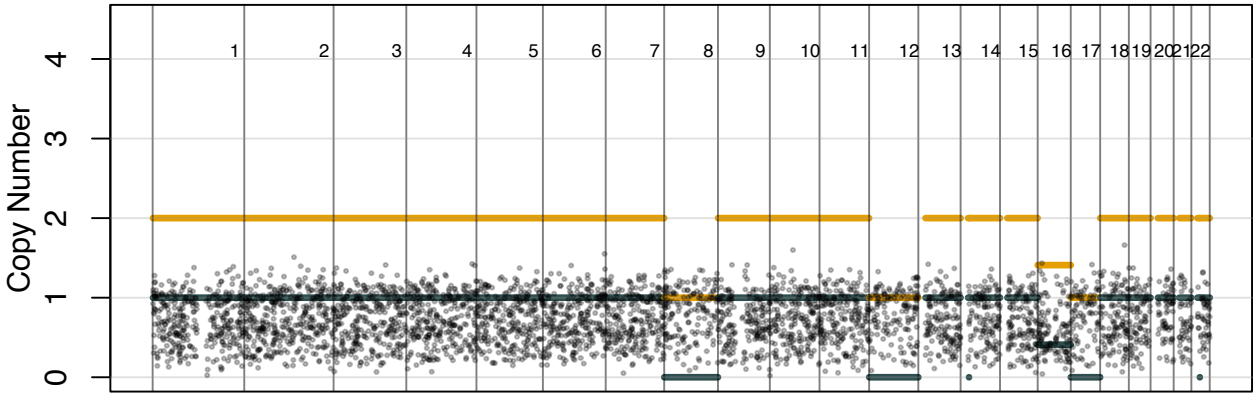


**Figure 8.** Mutation copy number vs. DNA copy number.

Because we have that $CCF = \frac{mutcopynumber}{multiplicity}$, in this sample, where $multiplicity = 1$, $CCF = mutcopynumber$.

Let's look at the distribution of the CCFs of mutations in non-subclonal copy number segments (**Figure 9**).

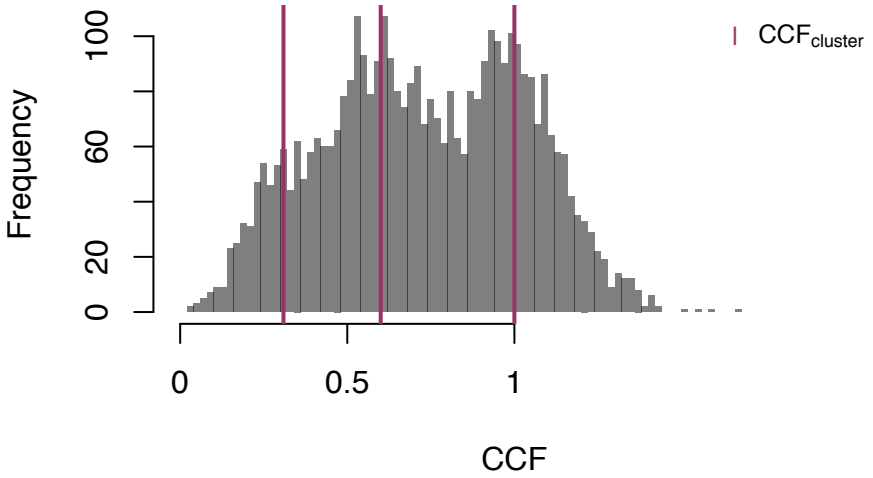


**Figure 9.** Distribution of CCF. Vertical lines indicate the expected CCF of the clones.

This looks like an easy clustering task, which is expected since the depth of coverage is so high.

### Clustering

DPClust is a clustering method that allows to automatically find the size, the number and the location of clusters of mutations at the same CCF value[3]. We will now run DPClust on the mutations, after integrating them with the Battenberg copy number profiles.

Next we run DPClust on the dpclust input file that we generated. This file contains all the necessary information for DPClust to run.

An important aspect of DPClust is that it has strong default prior on the concentration parameter, which should impact on the concentration/number of clusters. Not all methods put a strong prior on this parameter and allow for it to be influenced by the underlying data. In the original DPClust publication, the authors have tested the impact of this prior on the results and found that their clustering results were stable, even setting it to 1. We will try and replicate this result and run DPClust with a different value of this parameter.

The DPClust R package used here can be downloaded from **github**. DPClust has several dependencies, and there is a **Docker** file available to install DPClust and all its dependencies. If you have already installed R, the following packages are required:

BiocManager::install(c("VariantAnnotation", "mcclust","KernSmooth","ks","lattice", "ggplot2","gridExtra","reshape2"))

```
## load DPClust
suppressPackageStartupMessages(library(DPClust))
## load the data
cellularity <- read.table("input/T2-128X_refit_cellularity_ploidy.txt",header=T)[,"cellularity"]
conc_param <- 0.01 ## default value for the concentration parameter
dataset  <-  load.data(dpInputFile,
                       cellularity=cellularity,
                       Chromosome="chr",
                       position="end",
                       WT.count="WT.count",
                       mut.count="mut.count",
                       subclonal.CN="subclonal.CN",
                       no.chrs.bearing.mut="no.chrs.bearing.mut",
                       mutation.copy.number="mutation.copy.number",
                       subclonal.fraction="subclonal.fraction",
```

```
                        phase="phase",
                        is.male=T,
                        is.vcf=F,
                        ref.genome.version="hg19",
                        min.depth=1,
                        min.mutreads=0,
                        supported_chroms=as.character(1:22))
## run DPClust
clustering  <-
    DirichletProcessClustering(mutCount=dataset$mutCount,
                               WTCount=dataset$WTCount,
                               no.iters=1250,
                               no.iters.burn.in=250,
                               cellularity=cellularity,
                               totalCopyNumber=dataset$totalCopyNumber,
                               mutation.copy.number=dataset$mutation.copy.number,
                               copyNumberAdjustment=dataset$copyNumberAdjustment,
                               mutationTypes=dataset$mutationType,
                               samplename="T2",
                               subsamplesrun=c(),
                               output_folder="./output_0.01",
                               conc_param=conc_param,
                               cluster_conc=5,
                               mut.assignment.type=1,
                               most.similar.mut=NA,
                               min.frac.snvs.cluster=0.01,
                               max.considered.clusters=20)
writeStandardFinalOutput(clustering=clustering,
                         dataset=dataset,
                         most.similar.mut=NA,
                         outfiles.prefix="output_0.01/T2-0.01",
                         subsamplenames=c(),
                         assign_sampled_muts=T,
                         write_tree=F)
```

Let's look at the results. DPClust outputs a figure showing the location and size of the identified clusters in the CCF space (**Figure 10**).

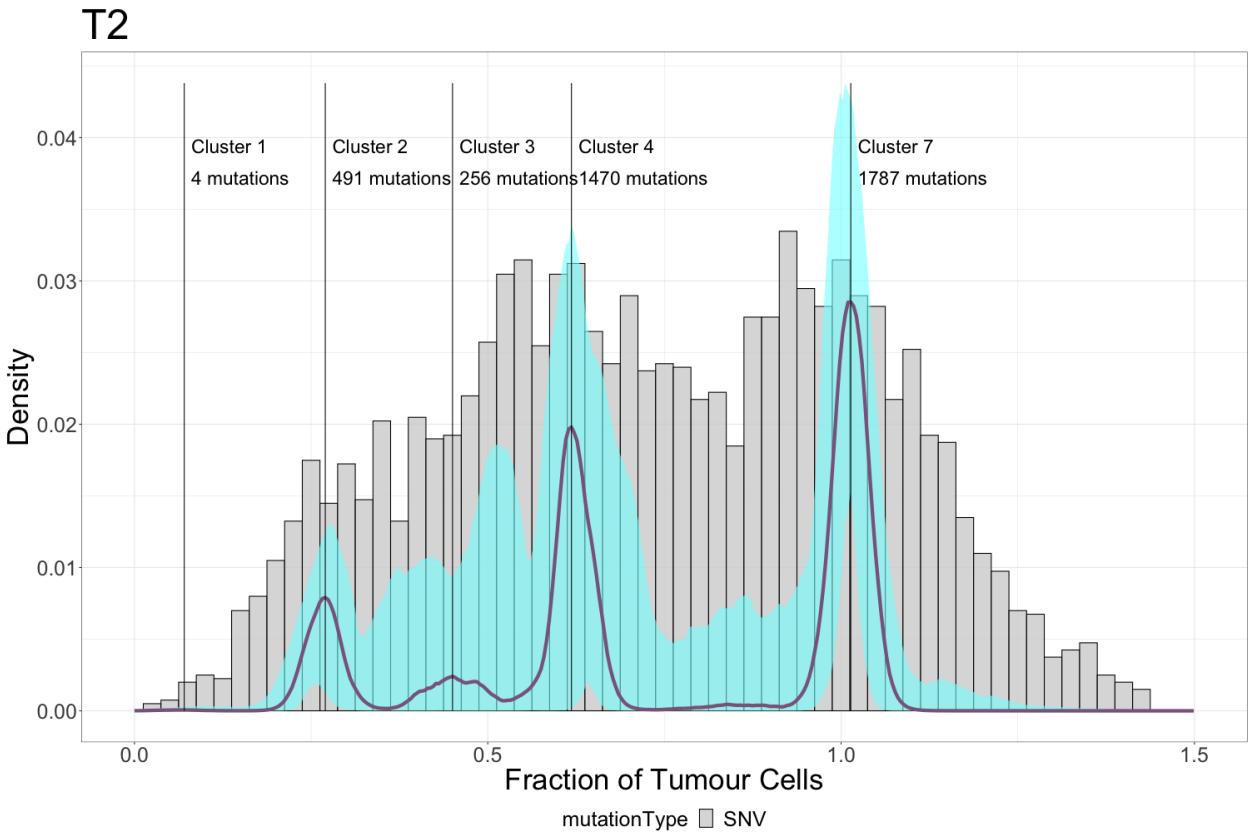


**Figure 10. DPClust main output figure for single sample.** The histogram of the CCF values is shown and the location and density distribution of the clusters inferred by DPClust is shown on top.

It looks like DPClust has found an extra cluster between the two subclones, and a substantial proportion of the clonal mutations have been assigned to subclones.

Next we will try running DPClust with a different concentration parameter (from 0.01 we change it to 1).

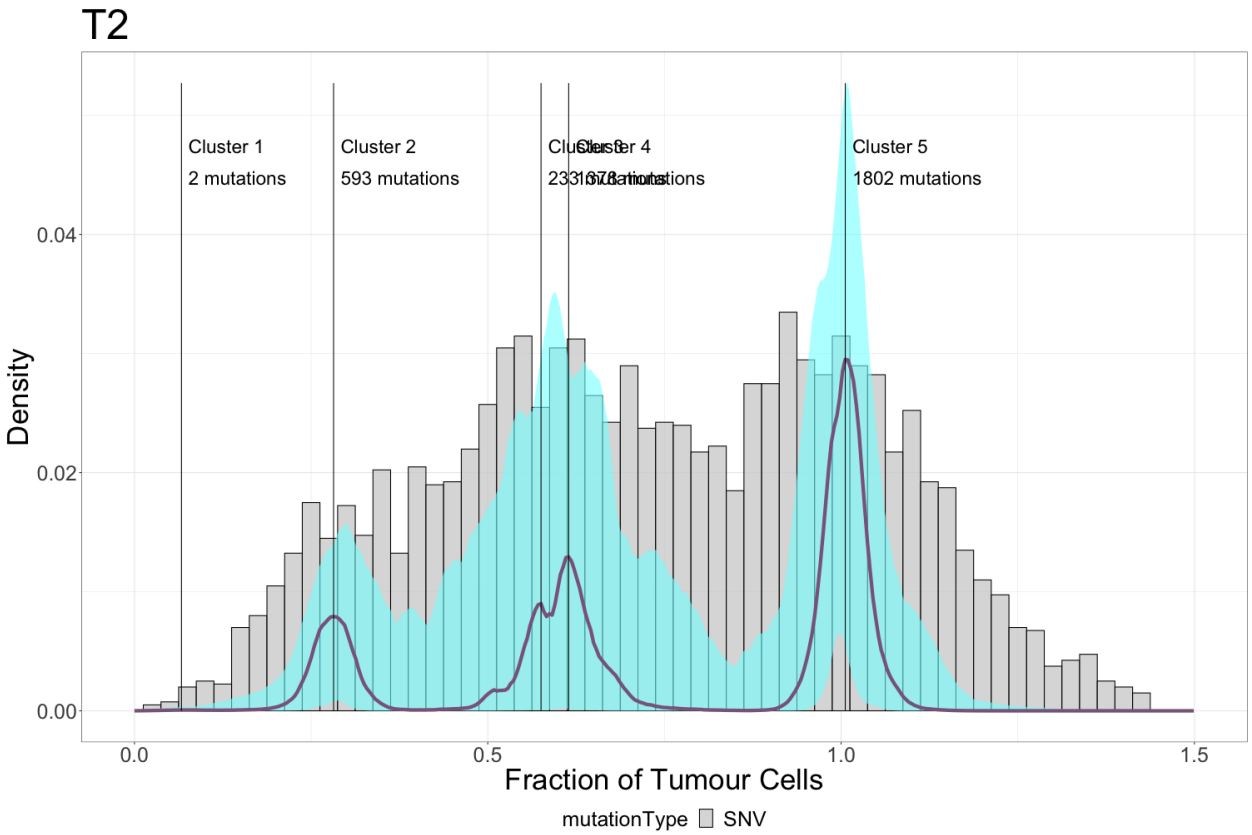


**Figure 11. DPClust main output figure for single sample (concentration parameter x100).** The histogram of the CCF values is shown and the location and density distribution of the clusters inferred by DPClust is shown on top.

In this very similar solution (**Figure 11**), we can see two very closeby clusters, which, if they were real, would need to be linear due to the pigeonhole principle (see Lexicon in main text). It is not unusual to see spurious clusters that are *too small* (e.g. <50 mutations or <1% of mutation load) as well as superclonal clusters (CCF>>100%). In single-sample studies, these do not add much to the subclonal reconstruction and we tend to merge them with the closest and larger cluster, as typically, the means of their distributions are not significantly different. For simplicity, we usually put hard thresholds on their mean CCF differences, e.g. $if(abs(CCF_{cluster_A} - CCF_{cluster_B}) > 0.1)\ merge(cluster_A, cluster_B)$, but one can also perform more formal statistical testing.

Thus, if we were to use this concentration parameter (1), the solution is actually pretty good after merging and removing the small cluster. The true number of clusters, their CCF and associated number of SNVs are:

| Subclone | CCF | N SNV |
|---|---|---|
| 1 | 1.00 | 2100 |
| 2 | 0.60 | 1650 |
| 3 | 0.31 | 600 |

In the previous run, DPClust has identified 3 clusters:

| Subclone | CCF | N SNV |
|---|---|---|
| 1 | 1.01 | 1787 |
| 2 | 0.62 | 1706 |
| 3 | 0.27 | 491 |

This is not far from the truth. What about mutation assignment? We can see that there are less mutations that have been clustered than there are in the design. This is partly because a few mutations are removed and re-assigned post hoc, so they are present in the assignment files, and partly because mutations with 0 alternate read counts are not present in the truth file at all. Let's have a look at the contingency table of assignment.

```
kable(contingency_table,"latex", booktabs=T) %>%
    kable_styling(position="center") %>%
    add_header_above(c("","Truth"=4))
```

| | Truth | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | NA |
| 2 | 9 | 95 | 403 | 0 |
| 3 | 5 | 159 | 58 | 0 |
| 4 | 231 | 1075 | 38 | 0 |
| 5 | 1642 | 127 | 8 | 0 |
| NA | 19 | 18 | 9 | 0 |

In the DPClust predictions, some mutations were not assigned to any cluster and are thus NA's in this table. These can be removed during the preprocessing step for various reasons, including low depth of coverage, missing data (e.g. mutations between copy-number segment SNP boundaries), or mutations falling on unsupported chromosome without copy number information.

The clustering contingency table looks relatively good, but this is not unexpected at such high depth. With such prediction, we can start characterising mutational signatures clonal vs. subclonal, and between subclone.

Now let's talk about the phylogeny.

### Phylogenetic tree

#### Single-sample tree

Once the clustering of SNVs is obtained, the tree building can start. Some methods have part or whole of the following process built in. For example, DPClust can take phasing of mutation as input to infer linear vs. branching subclones.

But here, we go through this step manually. Often the tree building from single samples is rather uninformative.

There are a few observations that can help disambiguate the phylogenetic relationship (i.e. linear vs. branching) between subclones:

- linear relationship
  - the pigeonhole principle: the sum of CCFs of branching subclones at the same tree height should be <=100% of the CCF of their most recent common ancestor. Indeed if it was >100%, this would mean that at least a few cells carry the same set of mutations. However, according to the infinite sites hypothesis, the same set of random mutations is highly unlikely to happen twice by chance alone. Therefore the smaller subclone must be a descendant of the bigger subclone.
  - phasing of two mutations assigned to different clusters A and B, resp. If two mutations are present on the same read pair, they must have been present in the same cell. This means that at least one cell of subclone A and one cell of subclone B carried the same mutation. Moreover, under the infinite sites hypothesis, each mutation appeared only once, hence, subclone A and B must be linearly related.
- branching relationship
  - two mutually exclusive mutations on haploid regions that are assigned to different clusters. On a clonally haploid region of the genome (i.e. 1 copy of allele A and 0 copies of allele B), only one copy of the region is left to be mutated in cancer cells. Given two mutations assigned to two linear subclones, located in this haploid region and close to each other (distance below the read length), it is expected that the reads carrying the mutation of the smaller subclone should carry the mutation of the larger subclone. If these two mutations are mutually exclusive, they must belong to branching subclones.

In our sample, there is no way to know if the two subclones are linear or branching. The sum of their $CCF_{subcloneA} + CCF_{subcloneB} < 1$, so it could be both linear or branching. Usually, it is easier to call linear than branching. But, if looking at mutation phasing in haploid regions only, the proportion of linear vs. branching subclone can be compared.

#### Note on multi-sample tree

With DPClust, multi-sample trees are also derived mostly manually. The clustering process generalises trivially to multiple dimensions, and the same code can be used to load the mutations from multiple samples.

An important step is to make sure the union of all mutations across samples has been *genotyped* if not called in all of them. Indeed, mutations called in a given sample might have been missed in another although the alternate read count is positive, i.e. $VAF > 0$. It is thus important to rederive the alternate and reference counts for all mutations in all samples before the clustering.

# Conclusion

In this guide, we have reviewed the steps of subclonal reconstruction one-by-one, running them on a simulated tumour for which we know the underlying phylogeny and subclonal structure. This has helped gain insight into the practicalities behind subclonal reconstruction.

We have seen that mutation callers can include false positives at low VAF, as well as false negatives. Copy number calling is ambiguous because the ploidy can always be doubled to fit the data to integers and manual refitting - which is often arbitrary - needs to be performed. Unless given the ploidy experimentally (here we know it from the truth design), there is no way to be 100% sure about the fit. There are hints in the data that can help diagnose bad copy number fits, for example wrong purity can be seen when reconstructing the CCF of SNVs and the highest peak is not at CCF=1 (i.e. the peak of clonal mutation).

We have illustrated the impact of read depth on the VAF distribution and CNA calling.

We have shown how some parameters of the clustering method can be tweaked to lead to slightly different results, especially the concentration parameter in DPClust, which can lead to different number of clusters and reconstruction.

Altogether, single-sample-based subclonal reconstruction is not very informative about the phylogeny and a lot of the sources of noise and ambiguities, such as clustering of mutation and inferring phylogenetic relationships, can be largely reduced using multi-sample strategies.

Unfortunately, multi-sample strategies are not always possible but single-sample studies remain informative about the subclonality of mutations and especially targetable drivers, their associated mutation processes and the timing of the MRCA.

Once the subclonal structure is obtained, post hoc assignment of other types of variants, such as indels and structural variants can be performed. Methods such as **MutationTimer** are designed to do this[10].

# References


1. Dentro, S. C., Wedge, D. C. & Van Loo, P. Principles of Reconstructing the Subclonal Architecture of Cancers. *Cold Spring Harbor Perspectives in Medicine* **7**, (2017).

2. Salcedo, A. *et al.* A community effort to create standards for evaluating tumor subclonal reconstruction. *Nature Biotechnology* **38**, 97–107 (2020).

3. Nik-Zainal, S. *et al.* The Life History of 21 Breast Cancers. *Cell* **149**, 994–1007 (2012).

4. Li, H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv:1303.3997 [q-bio]* (2013).

5. Ewing, A. D. *et al.* Combining tumor genome simulation with crowdsourcing to benchmark somatic single-nucleotide-variant detection. *Nature Methods* **12**, 623–630 (2015).

6. Cibulskis, K. *et al.* Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature Biotechnology* **31**, 213–219 (2013).

7. Van Loo, P. *et al.* Allele-specific copy number analysis of tumors. *Proceedings of the National Academy of Sciences* **107**, 16910–16915 (2010).

8. Howie, B. N., Donnelly, P. & Marchini, J. A Flexible and Accurate Genotype Imputation Method for the Next Generation of Genome-Wide Association Studies. *PLOS Genet* **5**, e1000529 (2009).

9. Dentro, S. C. *et al.* Portraits of genetic intra-tumour heterogeneity and subclonal selection across cancer types. *bioRxiv* 312041 (2018) doi:10.1101/312041.

10. Gerstung, M. *et al.* The evolutionary history of 2,658 cancers. *Nature* **578**, 122–128 (2020).