Introduction to *Palimpsest*

***Benedict Monteiro*[*1], *Jayendra Shinde*[†1], *Quentin Bayard*[1], *Théo Hirsch*[1], *Sandrine Imbeaud*[1], *Feng Liu*[2], *Victor Renault*[2], *Jessica Zucman-Rossi*[1], *and Eric Letouzé*[‡1]**

[1]INSERM UMRS1138, Génomique Fonctionnelle des Tumeurs Solides, Equipe Labellisée Ligue Contre le Cancer, Institut Universitaire d'Hématologie, Paris, France.
[2]Laboratory for Bioinformatics, Fondation Jean Dausset – CEPH, Paris, France.

*benedictmonteiro1@gmail.com †jayendra.shinde91@gmail.com ‡eric.letouze@gmail.com

1 July 2019

Abstract

Cancer genomes are altered by various mutational processes and, like palimpsests, bear the signatures of these successive processes. The *Palimpsest* R package provides a complete workflow for the characterisation and visualisation of mutational signatures and their evolution along tumour development. The package includes a wide range of functions for extracting single base substitution (SBS), double base substitution (DBS) and indel mutational signatures as well as structural variant (SV) signatures. *Palimpsest* estimates the probability of each mutation being due to each signature, which allows the the analysis of the clonality of each alteration, and the prediction of the mechanism at the origin of each driver event. In short, *Palimpsest* is an easy-to-use toolset for the reconstruction of the natural history of a tumour using whole exome or whole genome sequencing data. This document, paired with the "Palimpsest_test_script.R" demo script, outlines the typical workflow for the analyis of the genomic data from a series of tumours.

Package

Report issues at www.github.com/FunGeST/Palimpsest

Contents

1 Introduction

This document presents a typical mutational signature analysis of whole genome sequencing data using *Palimpsest*. The corresponding script ("Palimpsest_test_script.R") and genomic data (LiC1162) are provided in the package GitHub repository. This script (1) extracts *de novo* and known mutational signatures (the COSMIC reference signatures) from the example dataset, (2) estimates the probability of each mutation being due to each process, (3) infers the clonality of each mutation and thus compares early and late signatures, (4) estimates the timing of chromosome duplications using somatic mutations, (5) analyses structural variant signatures and (6) integrates all these results in a schematic tumour history plot.

Palimpsest 2.0, like the previous version, has single base substitution (SBS) and structural variant (SV) signatures extraction capabilities. Since the initial release of the package, two new types of mutation signatures, double base substition (DBS) and small insertion and deletion (indel), have been defined (Alexandrov et al., 2018). Functions for the extraction of these two mutation types are now included in *Palimpsest*, making it a comprehensive, but simple tool for the analysis of all genomic alterations present in the tumours of a cancer series. For more information on the latest definitions of mutational signatures please see the original paper and the COSMIC website.

2 Installation Instructions

The latest version of the package can be installed from the FunGeST GitHub repository using `devtools`:

```
----------------------------------------------------------------------
> install.packages("devtools")
> library(devtools)
> devtools::install_github("FunGeST/Palimpsest")
----------------------------------------------------------------------
```

3 Dependencies

To add indel mutation categories we use a python script provided by the Borad Institute from the ICGC pan cancer genome analysis (Alexandrov et al., 2018), which is embedded in the R function `annotate_VCF()`. For this to work the function must be run in a Unix environment (i.e. Mac or Linux) with python 2.7 installed. The other aspects of the `annotate_VCF()` function, and indeed all other functions, work on a Windows operating system. The indel aspect of this function also requires you to have a FASTA file compatible with the input VCF genome (including position and chromosome names) accessible in your local environment. If you only wish to work on SBS/DBS/SV signatures you can skip this step.

The R package *bedr* is required to perform structural variant signature analysis. The bedr API gives access to "BEDTools" and offers additional utilities for genomic region processing. To gain the functionality of bedr package you will need to have the BEDTools program installed and in your default PATH.

4 Input Data

One input file is necessary to perform the core Palimpsest analyses:

1. *vcf*: Somatic mutation catalogue of a tumour series (can contain SNVs, indels or both).

The study of somatic mutations can be extended to include an analysis of clonality, and/or structural variation signatures (both optional). Corresponding input files:

2. *cna_data*: Segmented copy-number data.
3. *annot_data*: Minimal sample annotation data (i.e. includes gender and tumour purity).
4. *sv_data*: File listing structural variants information

Please refer to the example files provided with the package for the correct format. You can also check out the README file for further information about the formats of the input files.

4.1 Loading genomic data and reference genome

Once installed, load the package and the reference genome and you're ready to go! *Palimpsest* works with the choice of reference genomes available via *BSgenome*. Ensure that you have the *BSgenome* library installed and then load the appropriate reference genome for your data.

```
-------------------------------------------------------------------
> # Load Palimpsest package
> library(Palimpsest)
> library(BSgenome.Hsapiens.UCSC.hg19) # Reference genome of choice
-------------------------------------------------------------------
```

Next we define the directory containing input data, and the desired output directory (resdir).

```
-------------------------------------------------------------------
> # define input directory containing example dataset
> datadir <- "Palimpsest/RUNNING_PALIMPSEST_EXAMPLE/LiC1162/"
>
> # define parent output directory
> resdir_parent <- "~/Results/"
> if(!file.exists(resdir_parent)) dir.create(resdir_parent)
-------------------------------------------------------------------
```

We provide example input datasets with this package in the *datadir*, from our paper Mutational signatures reveal the dynamic interplay of risk factors and cellular processes during liver tumorigenesis (Letouzé et al., 2017), which can be loaded as follows:

```
-------------------------------------------------------------------
> # Loading the example data.
> load(file.path(datadir,"vcf.RData"))
> load(file.path(datadir,"cna_data.RData"))
> load(file.path(datadir,"annot_data.RData"))
> load(file.path(datadir,"sv_data.RData"))
-------------------------------------------------------------------
```

4.2 Preparing input data for mutational signature analysis

Annotating the VCF prepares it for analysis in *Palimpsest*. The following function annotates each mutation with its corresponding COSMIC mutation category (be it SBS, DBS or indel), the gene in which it occurs (if any) and whether or not it occurs on the transcribed strand. The genes are added from a table of Ensembl genes (ensgene) that is provided with the package in both hg19 and hg38 formats. You may choose which mutation type categories are added (see the help documentation at `?annotate_VCF()` for more information on this). If you wish to add indel mutation categories, you must supply a filepath to a FASTA file compatible with your input data and be working in a Unix environment (see Dependencies).

```
-----------------------------------------------------------------
> # Annotate VCF with categories of all mutation types
> vcf <- annotate_VCF(vcf = vcf, ref_genome = BSgenome.Hsapiens.UCSC.hg19,
                      ref_fasta = "~/Homo_sapiens_assembly19.fasta")
>
> # Annotate VCF with SBS & DBS categories only
> # (Windows friendly & no FASTA dependency)
> vcf <- annotate_VCF(vcf = vcf, ref_genome = BSgenome.Hsapiens.UCSC.hg19,
                      add_ID_cats = FALSE)
-----------------------------------------------------------------
```

The following function produces the input for NMF extraction, which consists of a list of two matrices: *mut_nums*, the number mutations of each category in each sample, and *mut_props*, the proportion of mutations of each category in each sample. The function calculates the input for one mutation type at a time, i.e. while the *Type* argument is set to "SBS" it will output the number and proportions of each SBS category in each sample in the VCF.

```
-----------------------------------------------------------------
> SBS_input <- palimpsest_input(vcf = vcf, Type = "SBS")
> DBS_input <- palimpsest_input(vcf = vcf, Type = "DBS")
> ID_input  <- palimpsest_input(vcf = vcf, Type = "ID")
-----------------------------------------------------------------
```

5 Mutational Signatures

Palimpsest offers two approaches to mutational signature analysis. The first option is to perform a *de novo* extraction. In this case, non-negative matrix factorization (NMF) is used to estimate the number of different processes operative in the data, the signature of each process and its activity in each tumour. This approach can identify new mutational signatures not present in the COSMIC reference signatures. The other option is to extract known signatures (e.g. the reference signatures from COSMIC database provided with the package) from the input data. In this case NMF is only used to estimate the activity of each signature in each tumour.

For simplicity the following examples describe the extraction of SBS mutational signatures, but all of the functions work in the same way for DBS and indel signature analysis. For more information on their use with of DBS and indel signatures please see the relevant section.

5.1 *De Novo* mutational signature analysis using NMF

The *de novo* mutational signature extraction is based on the use of non-negative matrix factorisation using the *NMF* package (Gaujoux & Seoighe, 2010), which defaults to the use of the standard algorithm from Brunet et al. (2004). The factorisation rank (i.e. the optimal number of signatures) can be manually defined uisng the *num_of_sigs* parameter, or estimated automatically using the cophenetic correlation coefficients and residual sum of squares (RSS) as described in the original article (Gaujoux & Seoighe, 2010). For larger datasets, it is advisable to increase the number of iterations (*nrun*) parameter to avoid local minima and obtain a stable number of mutational signatures (We recommend 10 iterations for the 44WGS input and 20 iterations for larger datasets). The resulting signatures (Fig. 2) are plotted in the *resdir*.

```
-----------------------------------------------------------------
> SBS_denovo_sigs <- NMF_Extraction(input_matrices = SBS_input,
                                    range_of_sigs = 1:10, nrun = 10,
                                    resdir = resdir)
-----------------------------------------------------------------
```

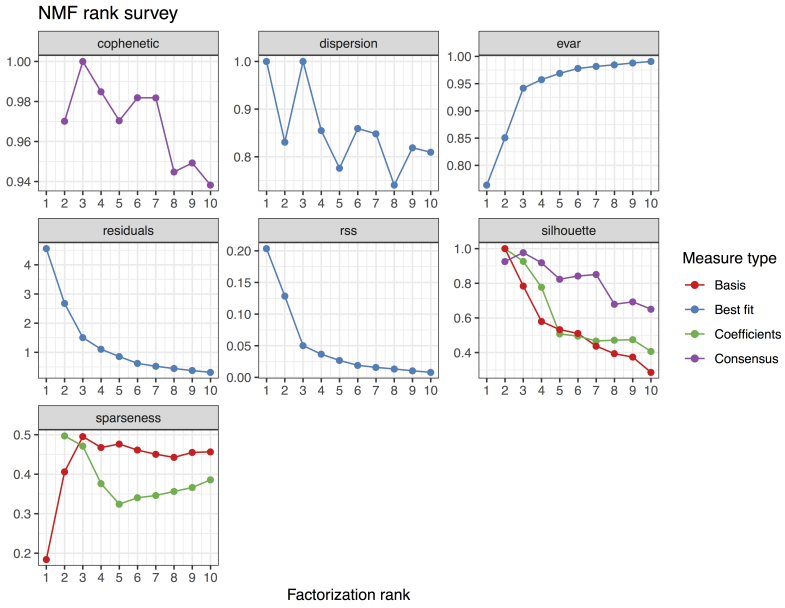

Figure 1: NMF rank estimation (plotted in the *resdir*). These outputs from the *NMF* package can be used to determine the optimal number of signatures. Alternatively, *Palimpsest* can estimate this automatically by taking the last rank before the cophenetic distance starts reducing dramatically (more than 0.02), which in the case of the example 44 samples is 7.

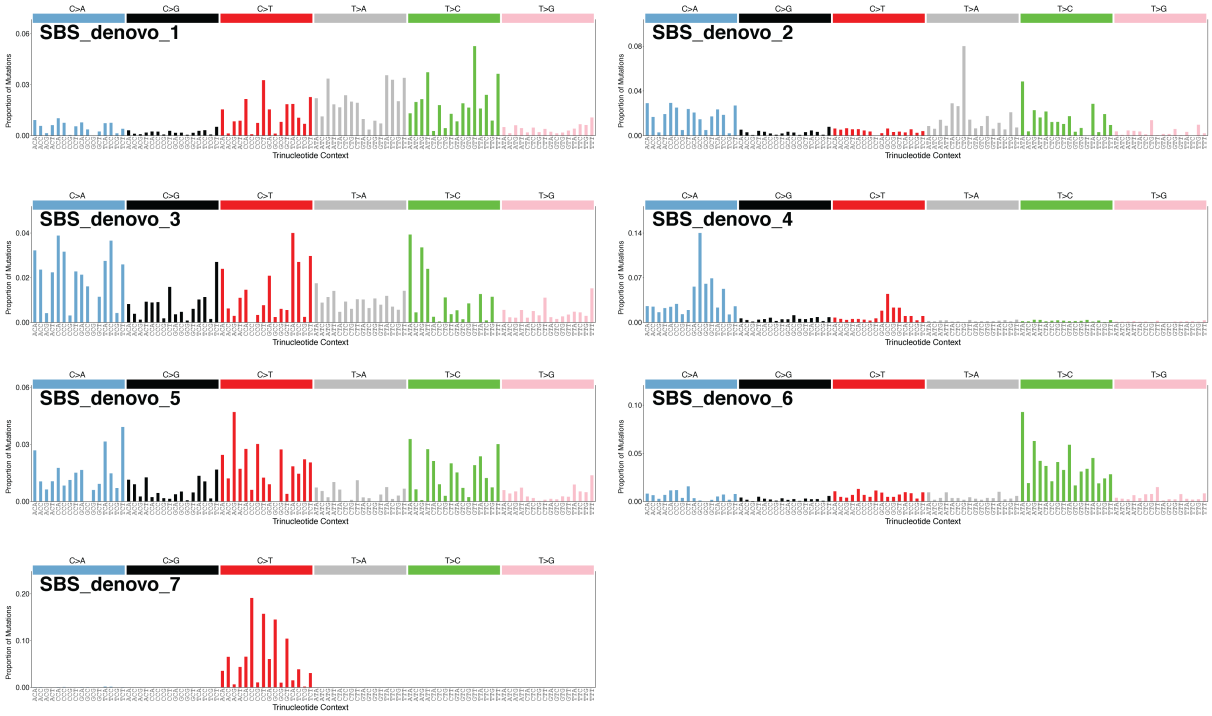

Figure 2: The 7 single base substitution mutational signatures extracted *de novo* from the 44 samples in the example dataset. Each mutational signature is represented as a barplot giving the frequency of mutations in each category, taking into account substitution types (top) and trinucleotide contexts (bottom).

5.2 Cosine similarity

Once *de novo* mutational signatures have been extracted, it is useful to determine whether they represent new mutational processes or if they correspond to previously described signatures. The `deconvolution_compare()` function estimates the cosine similarity score (0 = completely different, 1 = identical) between two sets of signatures (e.g. *de novo* signatures and the reference COSMIC signatures provided with the package) according to the method from Alexandrov et al. (2013). The `compare_results()` function prints a user friendly table that pairs each reference signature with its most similar *de novo* signature.

```
------------------------------------------------------------------
> # Compare the de novo signatures with published COSMIC signatures
> compare_results(reference_sigs = SBS_cosmic,extraction_1 = SBS_denovo_sigs)
> SBS_cosine_similarities <- deconvolution_compare(SBS_denovo_sigs,SBS_cosmic)
------------------------------------------------------------------
```

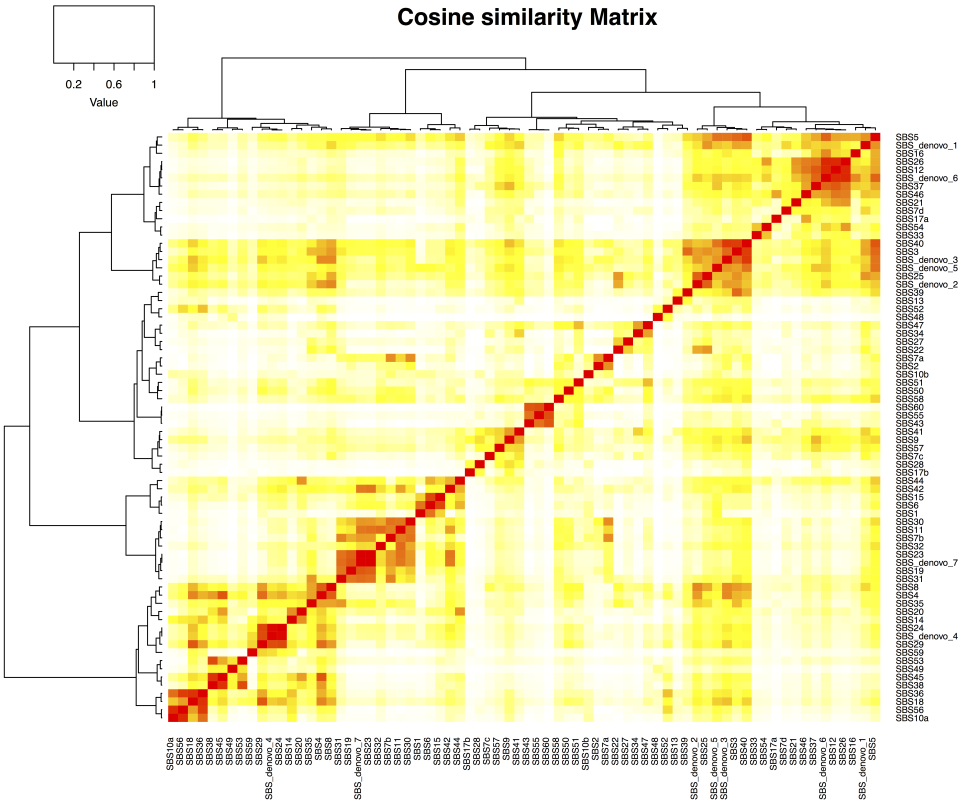

Figure 3: Cosine similarity heatmap comparing the 7 SBS *de novo* signatures to the reference COSMIC signatures. The colour code represents the similarity (white to red/0 to 1) between each pair of signatures. Signatures are grouped by their similarity using an unsupervised hierarchical clustering of their cosine similarity scores, with the corresponding dendrogram next to the plot.

5.3 Extracting known mutational signatures

In addition to *de novo* extraction, *Palimpsest* can extract known mutational signatures from the input tumours using NMF. You can skip the *de novo* extractions and start at this stage, although we highly recommend a *de novo* analysis as the start-point for any signature analysis, as this can identify completely new signatures/new variants of exisiting signatures in your data that may not be present in the COSMIC database.

You could provide a set of signatures from a previous *de novo* extraction, or use the COSMIC reference signatures from Alexandrov et al. (2018). In the following example we extract the 10 SBS signatures identified in liver tumours by Letouzé et al. (2018) from the example dataset.

```
-----------------------------------------------------------------
> # select desired COSMIC SBS reference signatures
> SBS_liver_names <- c("SBS1","SBS4","SBS5","SBS6","SBS12","SBS16",
                       "SBS17","SBS18","SBS22","SBS23", "SBS24")
> SBS_liver_sigs <- SBS_cosmic[rownames(SBS_cosmic) %in% SBS_liver_names,]
>
> # calculate and plot the exposure of the signatures across the series
> SBS_signatures_exp <- deconvolution_fit(input_matrices = SBS_input,
                                          input_signatures = SBS_liver_sigs,
                                          signature_colours = sig_cols,
                                          resdir = resdir)
>
> deconvolution_exposure(signature_contribution = SBS_signatures_exp,
                         signature_colours = sig_cols)
-----------------------------------------------------------------
```

5.4 DBS and Indel Signature Extraction

The analysis of DBS and indel signatures is a functionality new to *Palimpsest 2.0*, and serves as an alternative tool to those provided by Alexandrov et al. (2018). The *de novo* and COSMIC extractions are performed in exactly the same way as they are for SBS signatures. Excluding the `palimpsest_input()` function, all other *Palimpsest* functions will deduce which mutation type you are working with from the input you provide. See the "Palimpsest_test_script.R" demo script for examples usage of the package with these types. Example DBS and indel *de novo* extractions:

```
-----------------------------------------------------------------
> DBS_denovo_sigs <- NMF_Extraction(input_matrices = DBS_input,
                                    range_of_sigs = 1:10, nrun = 10,
                                    resdir = resdir)
> ID_denovo_sigs <- NMF_Extraction(input_matrices = ID_input,
                                    range_of_sigs = 1:10, nrun = 10,
                                    resdir = resdir)
-----------------------------------------------------------------
```

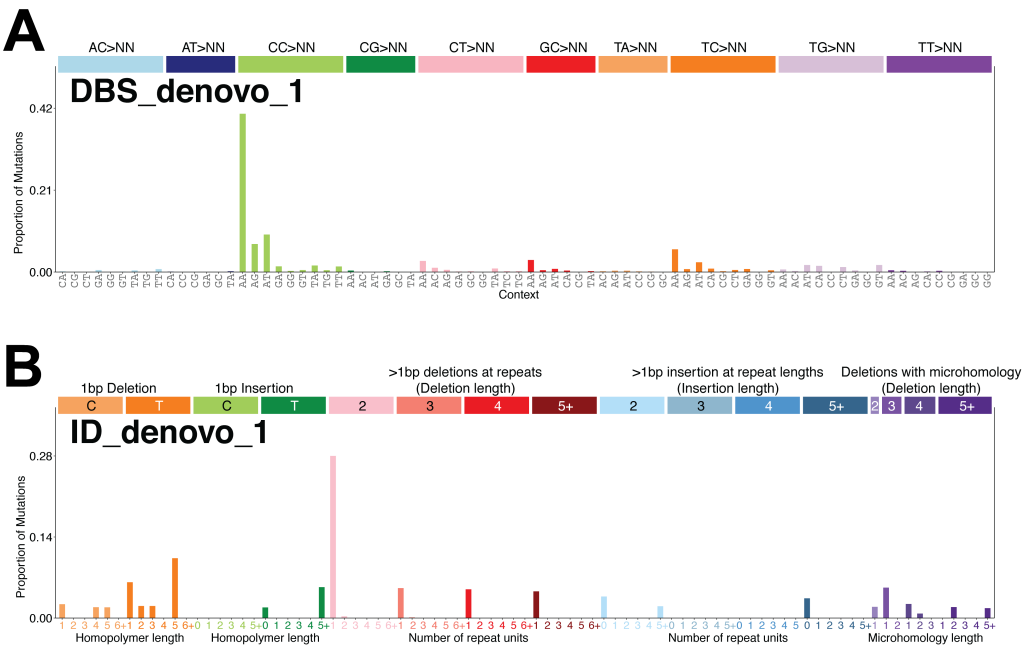

Figure 6: Examples of **(A)** DBS and **(B)** indel signatures extracted *de novo* by *Palimpsest*, displayed according to the 78 and 83 mutation categories respectively.

5.5 Estimating the exposures of mutational signatures

After extracting *de novo* or known mutational signatures, *Palimpsest* estimates the contribution of each signature to each individual tumour genome using the `deconvolution_fit()` function. Other graphical representations are also generated at this step (Fig. 4) and plotted in the *resdir*, such as the distribution of mutations across the mutation categories and a comparison of the transcription strand bias. The colours of each new signature in these graphical outputs can be generated automatically using the `signature_colour_generator()` function.

```
-----------------------------------------------------------------
> # Define signature colours for plotting
> SBS_col <- signature_colour_generator(rownames(SBS_denovo_sigs))
>
> # Calculate and plot the exposure of the signatures across the series
> SBS_signatures_exp = deconvolution_fit(input_matrices = SBS_input,
                                         input_signatures = SBS_denovo_sigs,
                                         signature_colours = SBS_col,
                                         resdir = resdir)
-----------------------------------------------------------------
```

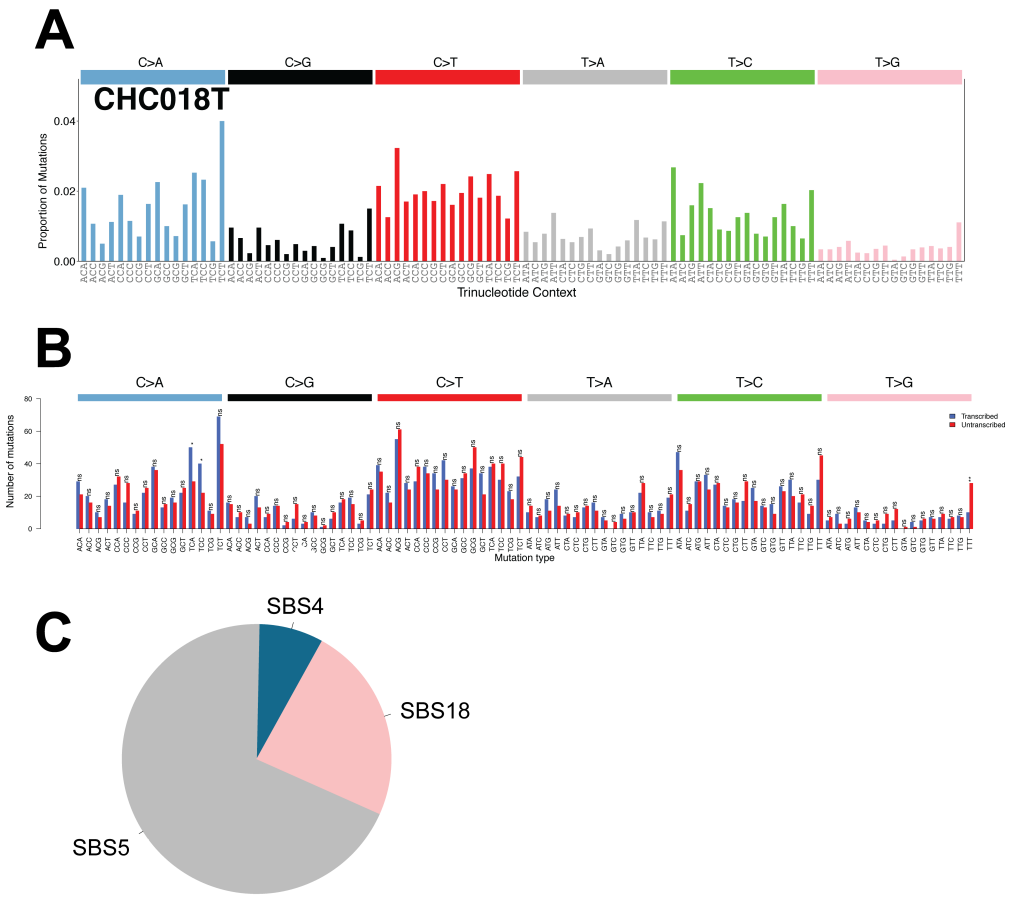

Figure 4: Fitting COSMIC mutational signatures into a tumour profile (CHC018T). The barplots indicate the distribution of mutations according to the 96 SBS mutation categories **(A)**, distinguishing whether or not mutations occurring on the transcribed and non-transcribed strands **(B)**. The pie chart indicates the contribution of the COSMIC signatures identified in the the mutation catalogue of the tumour by NMF **(C)**.

The `deconvolution_exposure()` function depicts the contribution of the signatures to each sample across the series (Fig. 5). Due to its high number of mutations, the hyper-mutated sample "CHC892T" has been removed from this example using the *rm_samples* parameter to aid the comparison of the numbers of mutations in the other samples.

```
-----------------------------------------------------------------
> deconvolution_exposure(signature_colours = SBS_col,
                         signature_contribution = SBS_signatures_exp,
                         rm_samples = c("CHC892T"))
-----------------------------------------------------------------
```

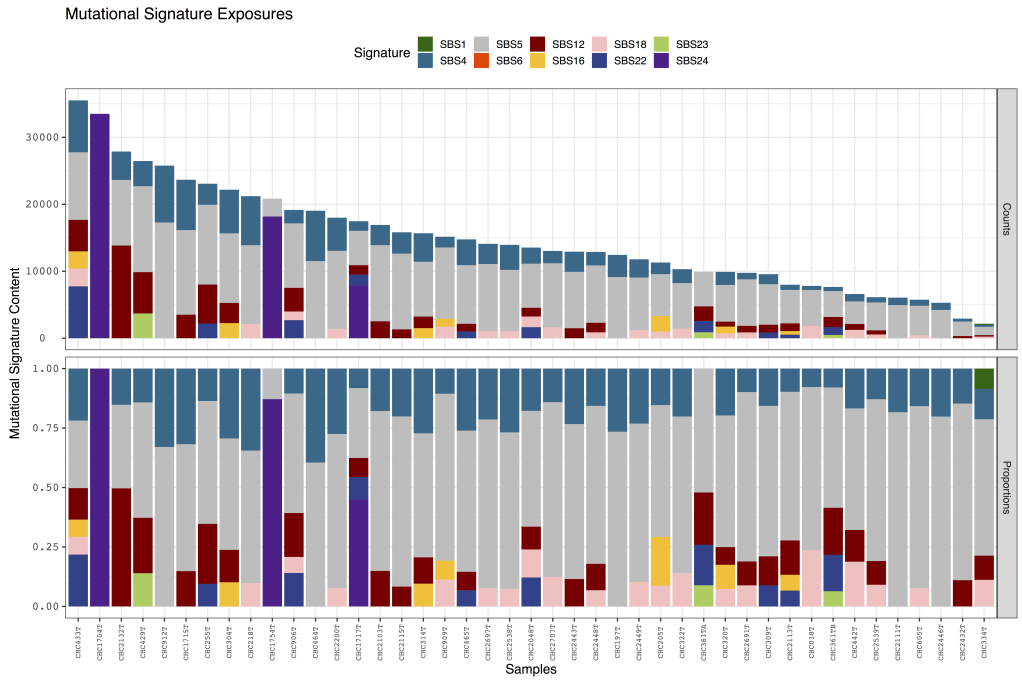

Figure 5: The contribution of the 10 COSMIC SBS signatures identified in liver cancer in the example dataset. The number (top) and proportion (bottom) of mutations attributed to each signature in each tumour is colour coded by signature.

5.6 Assigning the most likely signature at the origin of each mutation

Deciphering which signature gave rise to each mutation in a sample helps to elucidate the mechanism at the origin of the mutation, which is particularly useful for driver mutations. We developed a statistical framework to estimate the probability of each mutation being due to each process, considering the mutation category and the contribution of each signature to the corresponding tumour genome (Letouzé et al., 2017). The `signature_origins()` function implements this method to annotate each line of the VCF with these probabilities and the most probable signature.

Please see the "Palimspsest_test_script.R" for a step-by-step example of the usage of this functionality, where this function is employed to study the signatures behind driver gene events in the example dataset.

```
-----------------------------------------------------------------
> vcf <- signature_origins(input = vcf, Type = "SBS",
                           input_signatures = SBS_liver_sigs,
                           signature_contribution = SBS_signatures_exp)
-----------------------------------------------------------------
```

We can then estimate the cumulative contribution of signatures to each driver gene in the cohort, and identify processes preferentially associated with mutations in specific driver genes (Fig. 6). In this example, CTNNB1 mutations are preferentially associated with the age-related SBS5.

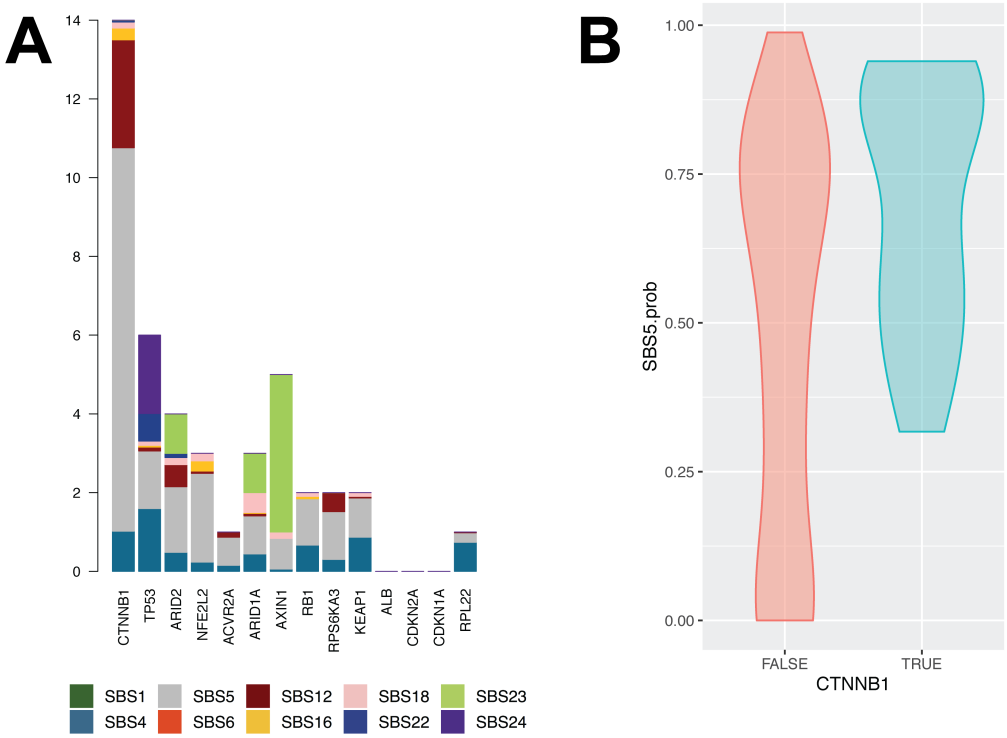

Figure 6: Association between mutational signatures and driver events in the 44 liver tumours. **(A)** The cumulative probabilities of driver gene mutations being due to each mutational process. **(B)** Comparison of the probability of each mutation being caused by SBS5 in CTNNB1 mutations and other coding mutations, demonstrating that CTNNB1 mutations are preferentially related to SBS5.

6 Clonality Analysis

Palimpsest provides functions to classify mutations as early clonal or late subclonal, and to monitor the evolution of mutational signatures along tumourigenesis in each tumour.

6.1 Copy number alterations and Cancer cell fraction (CCF)

First, the function `cnaCCF_annot()` allows you to calculate the cancer cell fraction (CCF) of each mutation, which is the proportion of tumour cells harbouring each mutation. This is done by adjusting the variant allele fraction (VAF) for the tumour purity (provided in the annotation file) and the absolute copy-number at each locus (provided in the CNA file). The 95% confidence interval of the CCF is also calculated, and mutations are classified as subclonal if the upper boundary of the 95% confidence interval is under a defined threshold (here, 0.95). The detailed formulas for CCF estimation are described by Letouzé et al. (2017).

```
-----------------------------------------------------------------
> vcf_cna <- cnaCCF_annot(vcf = vcf, annot_data = annot,
                          cna_data = cna_data, CCF_boundary = 0.95)
-----------------------------------------------------------------
```

This function adds several columns to the vcf file, including tumour purity, coverage log ratio (LogR), total number of copies at the locus (ntot), number of major (Nmaj) and minor (Nmin) alleles, cancer cell fraction (CCF) and confidence interval boundaries (CCF.min, CCF.max) and the assigned clonality status.

6.2 Clonality plots

The `cnaCCF_plots()` function visualises (Fig. 7) the clonality annotations deduced in the previous section:

```
-----------------------------------------------------------------
> cnaCCF_plots(vcf= vcf_cna, resdir = resdir)
-----------------------------------------------------------------
```

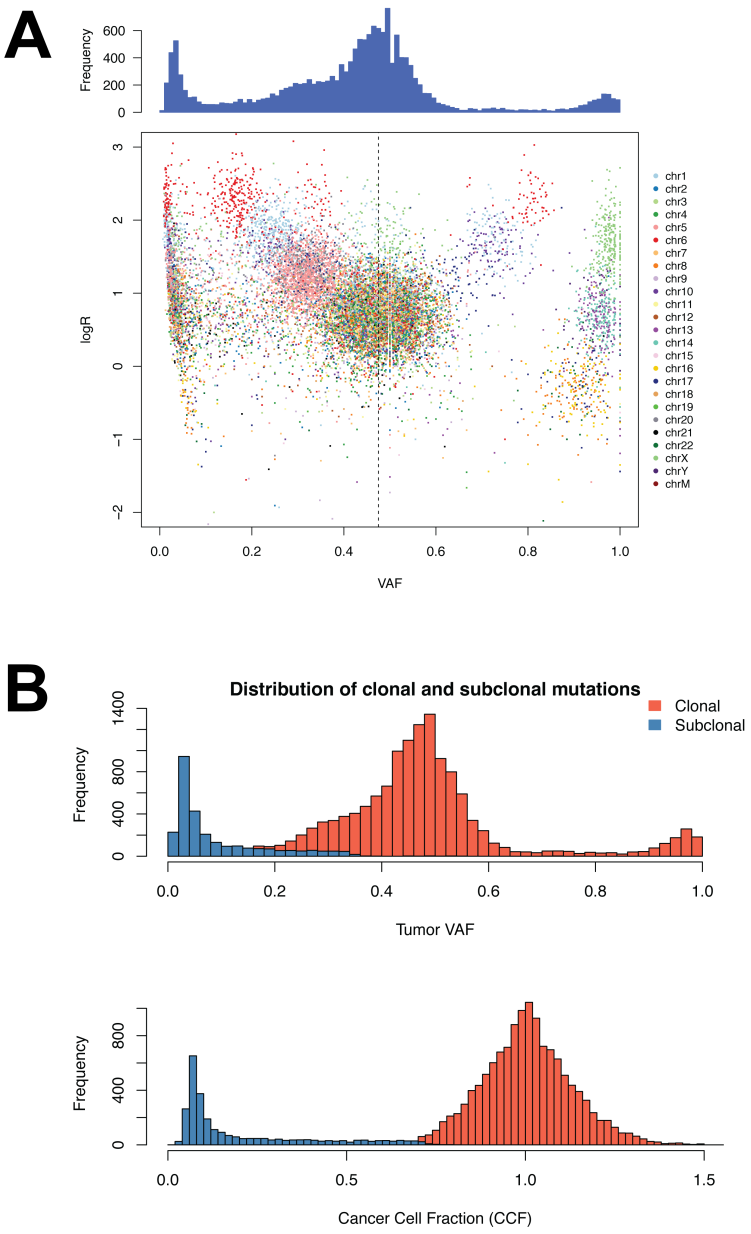

Figure 7: Genome-wide distribution of variant allele fractions (VAF) and cancer cell fractions (CCF) in a tumour (CHC909T). **(A)** The relationship between the VAF of mutations and local copy-number. Each point represents a somatic mutation, coloured according to chromosome, with the VAF on the x axis and coverage log-ratio between tumour and normal on the y axis. Deletions lead to a decreased log-ratio and increased VAF. Duplications lead to an increased log-ratio and either a decreased VAF (for mutations on the non-duplicated chromosome) or an increased VAF (for mutations on the duplicated chromosome). Subclonal mutations are visible as a cloud of mutations with very low VAF not explained by copy-number changes. **(B)** The histograms represent the distribution of VAF and CCF across all mutations in the tumour, with clonality status colour-coded.

6.3 Temporal evolution of mutational signatures

Once the clonality of each mutation has been established, Palimpsest allows you to analyse the evolution of mutational signatures between early clonal and late subclonal mutations. First, we reperform the deconvolution the contribution of each signature to each tumour, this time considering clonal and subclonal mutations separately.

```
-----------------------------------------------------------------
> # Estimate the contribution of each signature to clonal and
> # subclonal mutations in each tumour
> vcf.clonal <- vcf_cna[which(vcf_cna$Clonality=="clonal"),]
> SBS_input_clonal <- palimpsest_input(vcf = vcf.clonal,Type = "SBS")
> sig_exp_clonal <- deconvolution_fit(input_matrices = SBS_input_clonal,
                                      input_signatures = SBS_liver_sigs,
                                      resdir =  resdir,
                                      save_signatures_exp = F)
>
> vcf.subclonal <- vcf_cna[which(vcf_cna$Clonality=="subclonal"),]
> SBS_input_subclonal <- palimpsest_input(vcf = vcf.subclonal,Type = "SBS")
> sig_exp_subclonal <- deconvolution_fit(input_matrices = SBS_input_subclonal,
                                         input_signatures = SBS_liver_sigs,
                                         resdir = resdir,
                                         save_signatures_exp = F)
-----------------------------------------------------------------
```

Then, we use the `palimpsest_DissectSigs()` function to generate visual representations (Fig. 8) of the 96 mutation category spectrums in early and late mutations, and to compare the proportions of early and late mutations attributed to each signature in each tumour.

```
-----------------------------------------------------------------
> # Generate per tumour comparisons of clonal and subclonal mutations
> palimpsest_DissectSigs(vcf=vcf_cna,
                         signatures_exp_clonal = signatures_exp_clonal,
                         signatures_exp_subclonal = signatures_exp_subclonal,
                         sig_cols = sig_cols,resdir=resdir)
-----------------------------------------------------------------
```

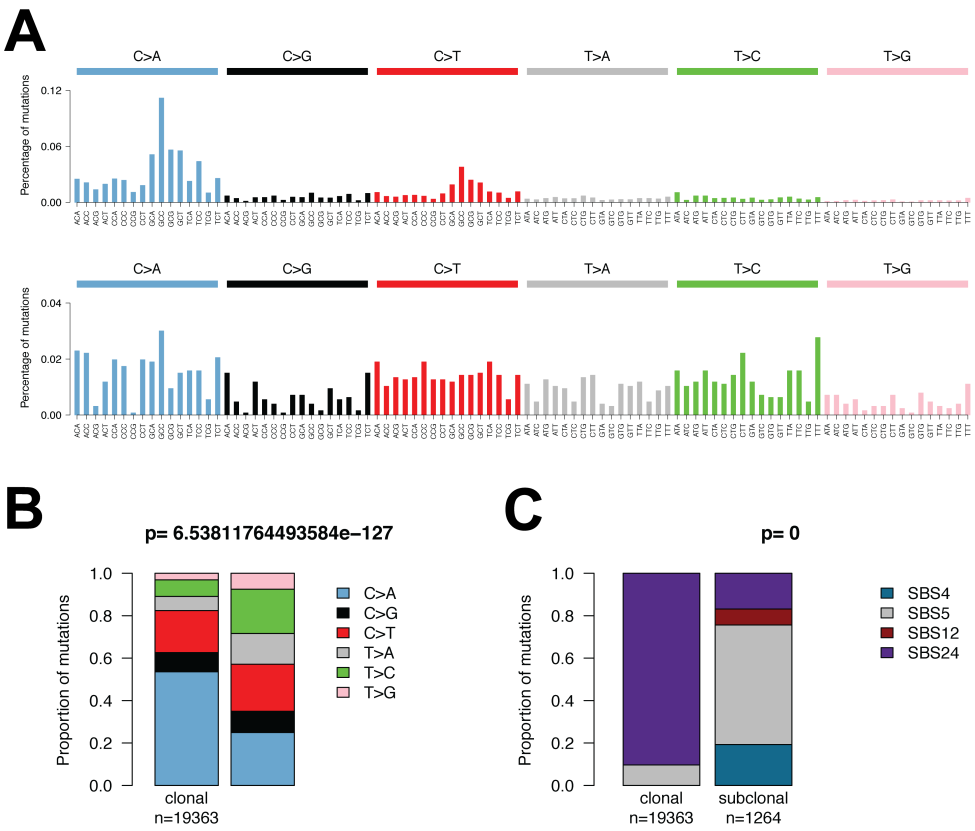

Figure 8: Temporal evolution of mutational signatures in a liver tumour (CHC1754T). **(A)** The 96 mutation category spectrums of clonal (top) and subclonal (bottom) mutations. **(B)** The proportions of the 6 substitution types in clonal and subclonal mutations, with the p-value above (chi-square test). **(C)** The proportions of clonal and subclonal mutations attributed to each signature, with the p-value above (chi-square test).

The evolution of mutational signatures between clonal and subclonal mutations across the series can also be conveniently visualised (Fig. 9) using the following function:

```
-------------------------------------------------------------------
> # Generate across the series comparisons of signature assigned
to clonal and subclonal mutations
> palimpsest_clonalitySigsCompare(clonsig = signatures_exp_clonal$sig_nums,
                                  subsig = signatures_exp_subclonal$sig_nums,
                                  msigcol = sig_cols, resdir = resdir)
-------------------------------------------------------------------
```

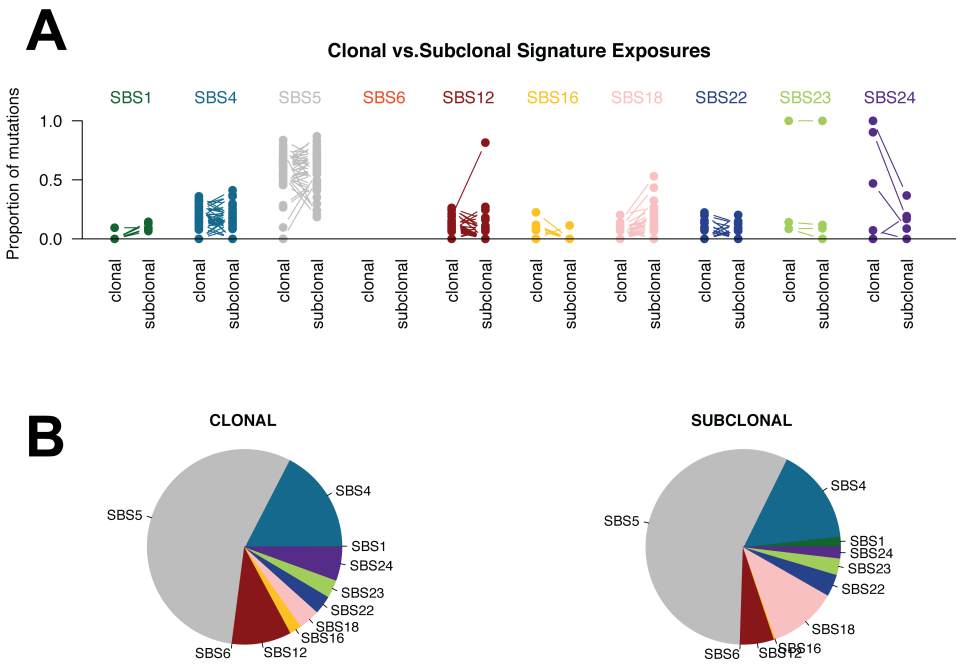

Figure 9: Evolution of mutational signatures between early clonal and late subclonal mutations in the example dataset. **(A)** The proportion of mutations attributed to each mutational signature in the clonal and subclonal mutations of each tumour (connected with a line). **(B)** The average contribution of each signature to clonal and subclonal mutations in all 44 tumours.

6.4 Timing chromosomal gains

Several tools exist that can identify early clonal and late subclonal copy-number alterations from Next-Gen sequencing data. *Palimpsest* allows to time the occurrence of chromosome duplications in molecular time. When a chromosome is duplicated (e.g. from 2 to 3 copies), mutations present in the duplicated chromosome copy are also duplicated and thus have an increased VAF. In contrast, mutations harboured by the other copy, or acquired after the duplication, are just present in one of the 3 copies and thus have a lower VAF. As a result, early duplications have fewer duplicated mutations in comparison to late duplications. *Palimpsest* uses the number of duplicated and non-duplicated mutations to estimate the timing of each chromosome duplication (see Letouzé et al. (2017) for a more detailed methodology). A cytoband table is required to generate graphical representations. Cytoband tables in hg19 and hg38 formats are provided with the package.

The `chrTime_annot()` function calculates this complex information from the *VCF* and CNA data, which is visualised by the `chrTime_plot()` function (Fig. 10).

```
-------------------------------------------------------------------
> # Annotate vcf with chromomal gain timings
> chrom_dup_time <- chrTime_annot(vcf = vcf_cna, cna_data = cna_data,
cyto = cytoband_hg19)
> vcf_cna <- chrom_dup_time$vcf
> point.mut.time <- chrom_dup_time$point.mut.time
> cna_data <- chrom_dup_time$cna_data
>
> # Visualising timing plots
> chrTime_plot(vcf = vcf_cna, point.mut.time = point.mut.time,
               resdir = resdir,cyto = cytoband_hg19)
-------------------------------------------------------------------
```

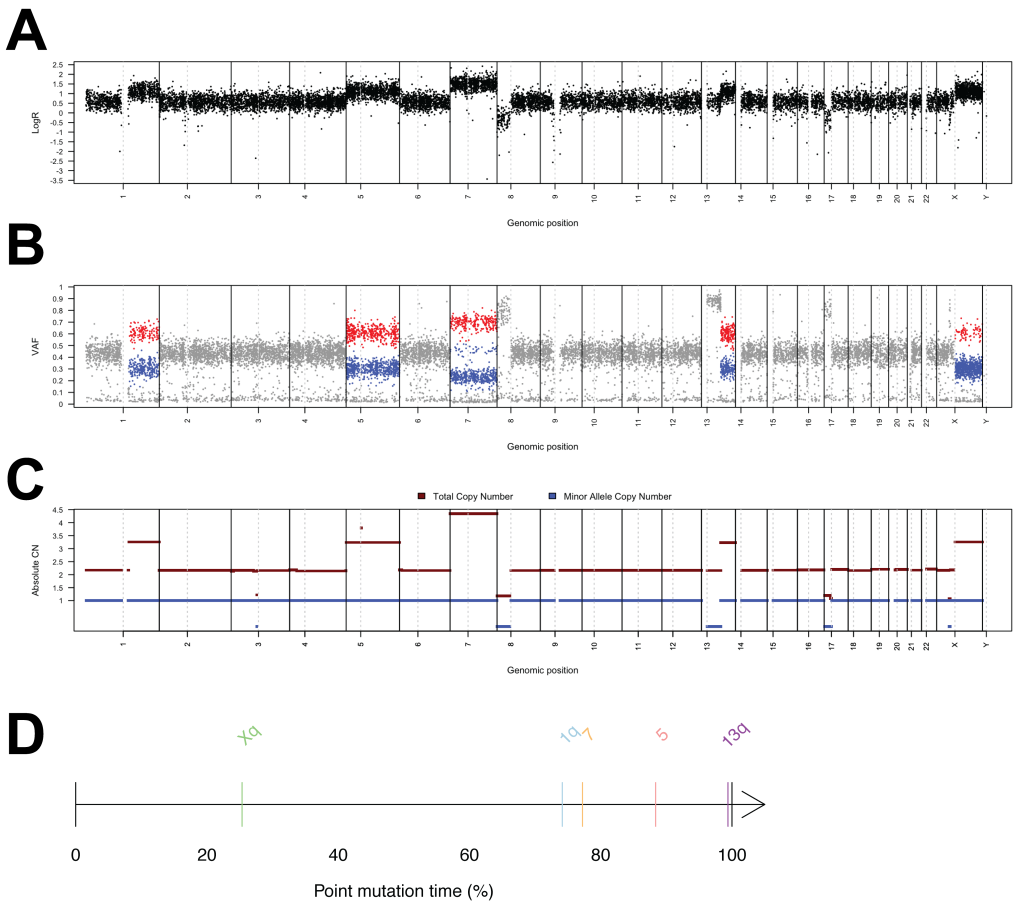

Figure 10: Chromosome duplication timing in a liver tumour (CHC2538T). The log-ratio **(A)** and variant allele fraction (VAF) **(B)** of each somatic mutation are represented throughout the genome, with the absolute copy-number below **(C)**. For each duplicated chromosome region, the colour of the points on the VAF graph distinguish duplicated (red) from non-duplicated (blue) somatic mutations. Here, most duplications have a similar amount of duplicated/non-duplicated mutations, indicating that they occurred late, when most mutations were already present. In contrast, the duplication of chromosome X has a lower number of duplicated mutations, so it must have occurred earlier. **(D)** The timing of chromosome duplications is represented in SNV mutation time (0 = time when no mutations had yet been acquired, 100 = time when they had all been acquired).

7 Structural Variants (SV) Signatures:

Palimpsest implements an adaptation of the mutational signature analysis framework to structural variants (SVs), as initially described by Nik-Zainal et al. (2016) and modified by Letouzé et al. (2017) and Bayard et al. (2018). SVs are first classified into 38 categories considering the type (deletion, tandem duplication, inversion, inter-chromosomal translocation), size (<1kb, 1-10kb, 10-100kb, 100kb-1Mb, 1-10Mb, >10Mb) and clustered nature of rearrangements. The same statistical tools as those used for other mutation types are then applied to extract SV signatures and their contribution to each tumour. *Palimpsest* also provides graphical representations of tumour SV profiles as CIRCOS plots and barplots showing the number of events per SV category. The workflow is very similar to that of the analysis of other mutation types:

```
-----------------------------------------------------------------------
> library(bedr); library(RCircos) # dependencies SV analysis
> SV.vcf <- preprocessInput_sv(input_data =  SV_data,resdir = resdir)
> SV_input <- palimpsest_input(vcf = SV.vcf,Type = "SV")
>
> # SV de novo extraction
> SV_denovo_sigs <- NMF_Extraction(input_matrices =  SV_input,
                                   range_of_sigs = 1:10, nrun = 10,
                                   num_of_sigs = 6, resdir = resdir)
-----------------------------------------------------------------------
```

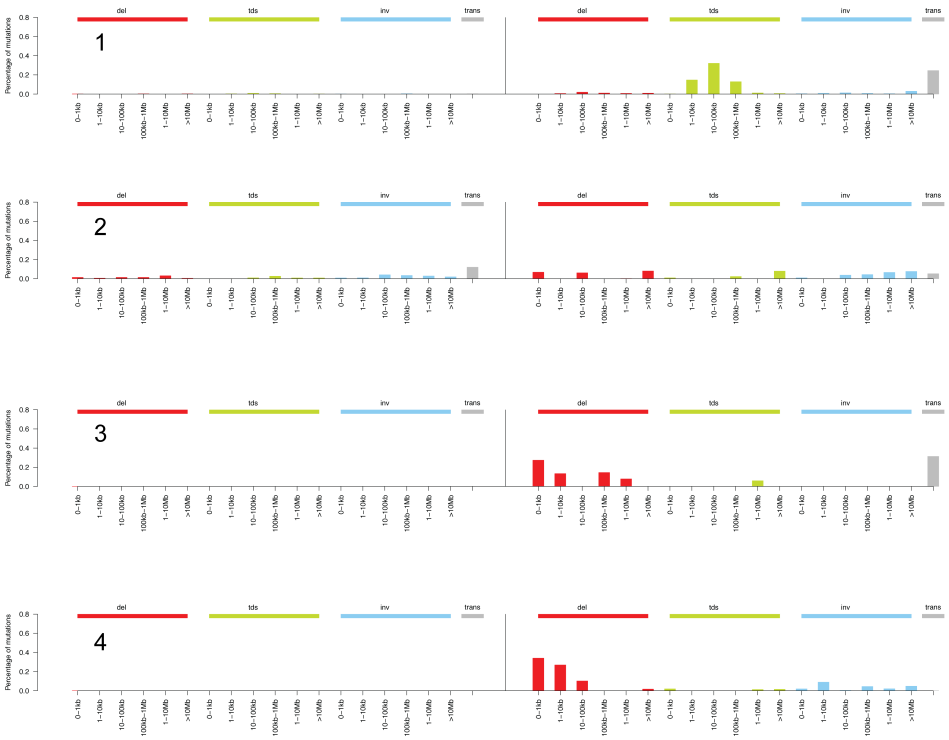

Figure 11: 4 rearrangement signatures identified in the example dataset. Structural rearrangements are classified in 38 categories considering their type (del: deletion, dup: tandem duplication, inv: inversion, trans: interchromosomal translocation) and size, and distinguishing clustered (left) from non-clustered events (right). The bargraphs represent the probability of each rearrangement category in each signature, with rearrangement types indicated above and rearrangement sizes below.

Further analyses of the exposure of the SV signatures across the tumour series (Fig. 12) are performed in the same way as they are for other mutation types:

```
-----------------------------------------------------------------
> # Calculate contribution of signatures in each sample:
> SVsignatures_exp <- deconvolution_fit_SV(vcf = SV.vcf,
                                           input_data = SV_input$mut_props,
                                           input_signatures = SV_denovo_sigs,
                                           sig_cols = SV_cols,
                                           resdir = resdir)
>
> # Plotting the exposures of signatures across the series:
> deconvolution_exposure(signature_contribution = SVsignatures_exp,
                         signature_colours = SV_cols)
>
> # Estimate the probability of each event being due to each process
> SV.vcf <- signature_origins(input = SV.vcf,Type = "SV",
                              signature_contribution = SVsignatures_exp,
                              input_signatures = SV_denovo_sigs)
-----------------------------------------------------------------
```

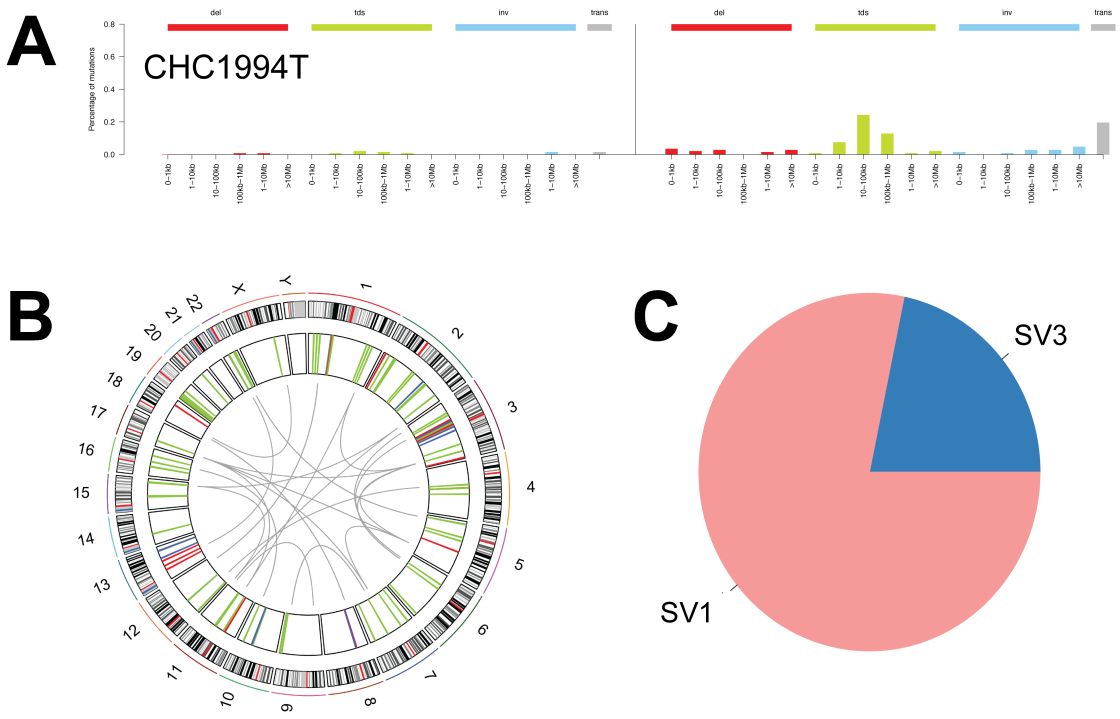

Figure 12: Fitting extracted structural variant signatures in a tumour profile. **(A)** The distribution of structural variations according to the 38 categories. **(B)** CIRCOS plot representing the structural rearrangement profile of the tumour genome. **(C)** The contribution of signatures in this sample.

8 Natural history of tumours

The final step in a typical *Palimpsest* analysis is to generate a schematic representation of the natural history of each tumour (Fig. 13). The `palimpsest_plotTumorHistories()` function combines the results of the previous sections to generate tumour history plots indicating the clonality of mutations, the contribution of mutational signatures to early clonal and late subclonal mutations, the timing of chromosome duplications, driver gene mutations and structural rearrangements (manually annotated by the user). Driver gene events are coloured according to the the mutational process that most likely generated them.

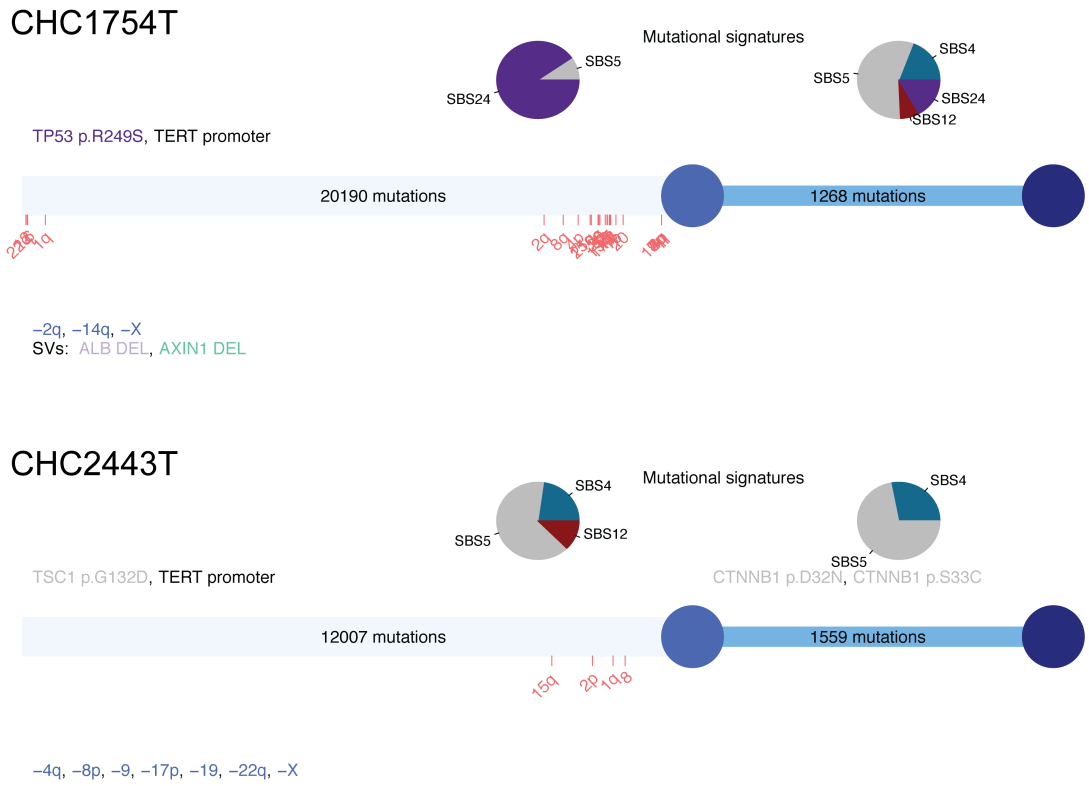

Figure 13: Plots representing the natural history of two liver tumours from the example dataset. The middle blue circle represents the last common ancestor of all tumour cells in the sample and the dark blue circle represents the final tumour sample. In the first tumour (aflatoxin B1-related, **top**), the highly active mutational signature 24 generated >20,000 clonal mutations including TP53 and TERT promoter mutations. This signature was less active in subclonal events, again replaced by signature 5. Deletions of ALB and AXIN1 genes also occurred early in tumour development, and synchronous acquisition of multiple gains occurred right before subclonal diversification of the tumour sample. The second tumour (unknown aetiology, **bottom**) shows less difference between its clonal and subclonal signatures.

9 References

1. Alexandrov, L. B. et al. (2012) Deciphering Signatures of Mutational Processes Operative in Human Cancer. *CellReports* 3, 246–259.
2. Alexandrov, L. B. et al. (2013) Signatures of mutational processes in human cancer. *Nature* 500, 415–21.
3. Alexandrov, L. B. et al. (2018) The Repertoire of Mutational Signatures in Human Cancer. *bioRxiv*.
4. Bayard, Q. et al. (2018) Cyclin A2/E1 activation defines a hepatocellular carcinoma subclass with a rearrangement signature of replication stress. *Nature Comms.* 9:5235
5. Gaujoux, R. & Seoighe, C. A flexible R package for nonnegative matrix factorisation. *BMC Bioinformatics* 11, 367.
6. Nik-Zainal, S. et al. (2016) Landscape of somatic mutations in 560 breast cancer whole-genome sequences. *Nature* 534, 47–54.
7. Nik-Zainal, S. et al. (2012) The life history of 21 breast cancers. *Cell* 149, 994–1007.
8. Letouzé, E., Shinde, J. et al. (2017) Mutational signatures reveal the dynamic interplay of risk factors and cellular processes during liver tumourigenesis. *Nature Comms.* 8(1):1315.

Session info

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] Palimpsest_2.0.0     GenomicRanges_1.36.0 GenomeInfoDb_1.20.0
##  [4] IRanges_2.18.1       S4Vectors_0.22.0     NMF_0.21.0
##  [7] synchronicity_1.3.5  bigmemory_4.5.33     Biobase_2.44.0
## [10] BiocGenerics_0.30.0  cluster_2.1.0        rngtools_1.3.1.1
## [13] pkgmaker_0.27        registry_0.5-1       dplyr_0.8.1
## [16] usethis_1.5.0        devtools_2.0.2       png_0.1-7
## [19] knitr_1.23           BiocStyle_2.12.0
##
## loaded via a namespace (and not attached):
##   [1] bigmemory.sri_0.1.3         colorspace_1.4-1
```

```
##   [3] ggsignif_0.5.0              rprojroot_1.3-2
##   [5] lsa_0.73.1                  XVector_0.24.0
##   [7] spgs_1.0-2                  fs_1.3.1
##   [9] rstudioapi_0.10             ggpubr_0.2.1
##  [11] SnowballC_0.6.0             remotes_2.0.4
##  [13] bit64_0.9-7                 AnnotationDbi_1.46.0
##  [15] codetools_0.2-16            doParallel_1.0.14
##  [17] pkgload_1.0.2               Rsamtools_2.0.0
##  [19] gridBase_0.4-7              graph_1.62.0
##  [21] BiocManager_1.30.4          compiler_3.6.0
##  [23] httr_1.4.0                  backports_1.1.4
##  [25] assertthat_0.2.1            Matrix_1.2-17
##  [27] lazyeval_0.2.2              cli_1.1.0
##  [29] htmltools_0.3.6             prettyunits_1.0.2
##  [31] tools_3.6.0                 gtable_0.3.0
##  [33] glue_1.3.1                  GenomeInfoDbData_1.2.1
##  [35] reshape2_1.4.3              Rcpp_1.0.1
##  [37] Biostrings_2.52.0           gdata_2.18.0
##  [39] rtracklayer_1.44.0          iterators_1.0.10
##  [41] xfun_0.7                    stringr_1.4.0
##  [43] ps_1.3.0                    testthat_2.1.1
##  [45] gtools_3.8.1                XML_3.98-1.20
##  [47] zlibbioc_1.30.0             RCircos_1.2.1
##  [49] scales_1.0.0                BSgenome_1.52.0
##  [51] VariantAnnotation_1.30.1    hms_0.4.2
##  [53] SummarizedExperiment_1.14.0 RColorBrewer_1.1-2
##  [55] yaml_2.2.0                  memoise_1.1.0
##  [57] ggplot2_3.2.0               biomaRt_2.40.0
##  [59] stringi_1.4.3               RSQLite_2.1.1
##  [61] desc_1.2.0                  plotrix_3.7-6
##  [63] foreach_1.4.4               caTools_1.17.1.2
##  [65] GenomicFeatures_1.36.1      pkgbuild_1.0.3
##  [67] BiocParallel_1.18.0         bibtex_0.4.2
##  [69] rlang_0.4.0                 pkgconfig_2.0.2
##  [71] bitops_1.0-6                matrixStats_0.54.0
##  [73] evaluate_0.14               lattice_0.20-38
##  [75] purrr_0.3.2                 GenomicAlignments_1.20.1
##  [77] bit_1.1-14                  processx_3.3.1
##  [79] tidyselect_0.2.5            plyr_1.8.4
##  [81] magrittr_1.5                bookdown_0.11
##  [83] R6_2.4.0                    gplots_3.0.1.1
##  [85] DelayedArray_0.10.0         DBI_1.0.0
##  [87] pillar_1.4.1                withr_2.1.2
##  [89] RCurl_1.95-4.12             tibble_2.1.3
##  [91] crayon_1.3.4                KernSmooth_2.23-15
##  [93] uuid_0.1-2                  rmarkdown_1.13
##  [95] progress_1.2.2              grid_3.6.0
##  [97] Rgraphviz_2.28.0            blob_1.1.1
##  [99] callr_3.2.0                 digest_0.6.19
## [101] xtable_1.8-4                munsell_0.5.0
## [103] sessioninfo_1.1.1
```