CellPhy - The hard way (tree search, bootstrapping, mutation mapping)

Alexey Kozlov, João M Alves*, Alexandros Stamatakis & David Posada

March 2021

1 Introductory note

In the following supporting document, we will go through all steps required to perform a full single-cell phylogenetic analysis using **CellPhy**, including phylogenetic tree search, bootstrap support estimation and mutation mapping. For reproducibility purposes, we provide a toy single-cell dataset (from a colorectal cancer patient - CRC24) together with a step-by-step tutorial containing all information needed to generate similar results to the ones presented in **Figure 6** of the main manuscript (Kozlov *et al.* 2020).

Importantly, in order to speed up the analysis, the toy example used here is composed of 500 randomly chosen SNVs, thus corresponding to a "thinned" version of the original dataset. In any case, we followed exactly the same steps shown in this document to generate the results presented in our study.

Before starting, make sure to download and install CellPhy from:

* CellPhy's github

CellPhy distibution contains following files and folders:

```
cellphy
  |- bin/                    <- raxml-ng binaries
  |- doc/                    <- documentation
  |- example/                <- toy dataset for this tutorial
  |- script/                 <- scripts for specific subtasks (e.g. mutation mapping)
  |- cellphy.sh              <- top-level script
  |- install.sh              <- dependency installation script
```

The simplest way to use CellPhy is to run

```
$ cellphy.sh input.vcf
```

This will perform the standard analysis with default parameters (run `cellphy.sh -h` for details).

In this tutorial, we will use the "expert mode" of CellPhy (`cellphy.sh RAXML` command). It allows for the highest level of customization by specifying the full RAxML-NG command line.

We will run all commands from the `example` directory where sample input files reside, and thus we will use the respective relative paths for all scripts (e.g., `../cellphy.sh` for the main script).

---

2 Input data

CellPhy accepts two types of input:

- 1) matrix of genotypes in FASTA or **PHYLIP** format - *CRC24.ToySet.phy*
- 2) **standard VCF file** - *CRC24.ToySet.vcf*

When the input is a genotype matrix, genotypes are encoded as shown in Table S2 of Kozlov *et al.* 2020. When the input is a VCF, CellPhy can be run in two distinct modes. The first mode ("**CellPhy-ML**") requires a VCF with at least the GT field (that stores the genotype calls), in which case CellPhy simply extracts the genotype matrix. The second mode ("**CellPhy-GL**") requires a VCF with the PL field (which stores the phred-scaled genotype likelihoods) and uses instead the likelihoods of each genotype.

Here's how they look like:

```
# PHYLIP file
$ head -n 5 CRC24.ToySet.phy
25 500
TI-TNS10   SNNCGTNCTNGAATGCNTANTGANNTYTNTNTGCG [...]
TI-TNS11   NNGCNYANTNGAGNGTGTATTGAANCNTGTNTANT [...]
TI-TNS12   NNACGTATNAGAATGNGTATNGANNCYTGNGCACG [...]
TI-TNS16   CNNNNTACTAGNATGNATATTGANTNNTGNGTANT [...]

# VCF file
$ grep -v "##" CRC24.ToySet.vcf | head -n 5
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  TI-TNS10    TI-TNS11 [...]
1   5107124 .   G   C   21  .   .   GT:SO:AD:BI:GQ:PL   0/1:True:6,6:0.594:21:58,37,148 ./.:.:.:.:.:. [...]
1   23800582    .   T   A   6   .   .   GT:SO:AD:BI:GQ:PL   ./.:.:.:.:.:.   ./.:.:.:.:.:. [...]
1   25776505    .   A   G   13  .   .   GT:SO:AD:BI:GQ:PL   ./.:.:.:.:.:.   1/1:True:0,1:0.6:4:9,2,0 [...]
1   26426850    .   C   T   92  .   .   GT:SO:AD:BI:GQ:PL   0/0:True:9,0:0.6:15:5,20,284    0/0:True:9,0:0.6:15:5,20,286 [...]
```

---

3 Tree inference

Let us now infer a tree with default parameters using both models of CellPhy. Here, we will use a single thread and set a fixed random seed to ensure reproducibility.

3.1 Tree inference with "CellPhy-ML"

This model will take as input a genotype matrix to infer a tree under the genotype error model "*GT16+FO+E*". If the input file is a PHYLIP file, CellPhy will automatically recognize the file type and proceed with the analysis. In contrast, if the input is a VCF file we need to set `--prob-msa off` which will instruct CellPhy to use the genotype field (GT) instead of the genotype likelihoods (PL).

These two runs should therefore return exactly the same result:

```
$ ../cellphy.sh RAXML --msa CRC24.ToySet.phy --model GT16+FO+E --seed 2 --threads 1 --prefix CRC24.PHY.ML16.Tree
$ ../cellphy.sh RAXML --msa CRC24.ToySet.vcf --model GT16+FO+E --seed 2 --threads 1 --prefix CRC24.VCF.ML16.Tree --prob-msa off
```

```
Analysis options:
  run mode: ML tree search
  start tree(s): random (10) + parsimony (10)

[00:00:00] Reading alignment from file: CRC24.ToySet.phy
[00:00:00] Loaded alignment with 25 taxa and 500 sites

[...]
```

In default settings, CellPhy will perform 20 tree searches using 10 random and 10 parsimony-based starting trees, and pick the best-scoring topology. While this is a reasonable choice for many practical cases, users are free to increase the number of starting trees to explore the tree space more thoroughly (e.g. for 50 tree searches, using 25 random and 25 parsimony starting trees, one would add `--tree pars{25},rand{25}` to the command line above).

Once it's finished (~ 5 minutes), CellPhy will output the ML estimates of the genotyping error and the ADO rates as well as the final tree score.

```
Optimized model parameters:

   Partition 0: noname
   Error model (ML): P20,  SEQ_ERROR: 0.000000,  ADO_RATE: 0.698807
   Rate heterogeneity: NONE
   Base frequencies (ML): 0.046975 0.076605 0.093862 0.059652 0.000047 0.000047 0.062979 0.000047 0.000047 0.000047 [...]
   Substitution rates (ML): 0.001000 0.001000 0.001000 27.354070 989.178276 93.810669 0.001000 0.001000 0.001000 27.354070 [...]

Final LogLikelihood: -9536.825816
```

If we now compare the results obtained with the two runs we can confirm that they are indeed the same:

```
$ grep "Final LogLikelihood:" CRC24.{PHY,VCF}.ML16.Tree.raxml.log
CRC24.PHY.ML16.Tree.raxml.log:Final LogLikelihood: -9536.825816
CRC24.VCF.ML16.Tree.raxml.log:Final LogLikelihood: -9536.825816
```

3.2 Tree inference with “CellPhy-GL”

Let’s now use instead the genotype likelihood model of CellPhy. This time, our input file needs to be a VCF with an appropriate PL field - if your VCF file was generated using the **SC-Caller algorithm**, make sure to use our VCF-conversion script (*sc-caller-convert.sh*) to add a standard PL Field to it.

```
$ ../cellphy.sh RAXML --msa CRC24.ToySet.vcf --model GT16+FO --seed 2 --threads 1 --prefix CRC24.VCF.GL16.Tree
```

Again, once it finishes, the program will output the final tree score and write all results into different files:

```
Optimized model parameters:

   Partition 0: noname
   Rate heterogeneity: NONE
   Base frequencies (ML): 0.123626 0.175592 0.186970 0.142850 0.000124 0.000124 0.027265 0.013460 0.000124 0.000124 [...]
   Substitution rates (ML): 0.001000 0.001000 0.001000 304.877289 263.121312 306.737332 0.001000 0.001000 0.001000 [...]

Final LogLikelihood: -30087.656427

[...]

Best ML tree saved to: CRC24.VCF.GL16.Tree.raxml.bestTree
All ML trees saved to: CRC24.VCF.GL16.Tree.raxml.mlTrees
Optimized model saved to: CRC24.VCF.GL16.Tree.raxml.bestModel
```

3.3 Tree inference with the GT10 model

In addition to the default genotype model with 16 states (*GT16*), CellPhy offers a faster approximate model with 10 states – *GT10*:

```
$ ../cellphy.sh RAXML --msa CRC24.ToySet.phy --model GT10+FO+E --seed 2 --threads 1 --prefix CRC24.PHY.ML10.Tree
$ ../cellphy.sh RAXML --msa CRC24.ToySet.vcf --model GT10+FO --seed 2 --threads 1 --prefix CRC24.VCF.GL10.Tree
```

On unphased genotype input data, which in fact only contain 10 states, the *GT10* model appears to be as accurate as *GT16*, but requires only half of the time:

```
$ grep Elapsed CRC24.VCF.ML1?.Tree.raxml.log
CRC24.VCF.ML10.Tree.raxml.log:Elapsed time: 122.266 seconds
CRC24.VCF.ML16.Tree.raxml.log:Elapsed time: 240.350 seconds
```

---

4 Bootstrapping trees and tree branch support

Next we will perform a standard non-parametric bootstrap by re-sampling alignment columns and re-inferring trees for each bootstrap replicate. By default, CellPhy will automatically determine the optimal number of replicates (up to 1000), but here we will manually set the number of replicates to 100 in order to keep runtime reasonable.

```
$ ../cellphy.sh RAXML --bootstrap --msa CRC24.ToySet.vcf --model GT16+FO --seed 2 --threads 1 --bs-trees 100 \
    --prefix CRC24.VCF.GL16.Boot

[...]
[00:00:00] Data distribution: max. partitions/sites/weight per thread: 1 / 500 / 5000
[00:00:00] Starting bootstrapping analysis with 100 replicates.
[00:00:11] Bootstrap tree #1, logLikelihood: -31748.764525
[00:00:20] Bootstrap tree #2, logLikelihood: -30019.613055
[...]
[00:15:07] Bootstrap tree #99, logLikelihood: -29931.390293
[00:15:18] Bootstrap tree #100, logLikelihood: -29263.239019

Bootstrap trees saved to: CRC24.VCF.GL16.Boot.raxml.bootstraps
```

Now that we have the bootstrap trees, we can map the BS support values onto the (previously inferred) best-scoring ML tree:

```
$ ../cellphy.sh RAXML --support -tree CRC24.VCF.GL16.Tree.raxml.bestTree \
    --bs-trees CRC24.VCF.GL16.Boot.raxml.bootstraps \
    --prefix CRC24.VCF.GL16.Support --threads 1

Reading reference tree from file: CRC24.VCF.GL.Tree.raxml.bestTree
Reference tree size: 25

Reading bootstrap trees from file: CRC24.VCF.GL.Boot.raxml.bootstraps
Loaded 100 trees with 25 taxa.

Best ML tree with Felsenstein bootstrap (FBP) support values saved to: CRC24.VCF.GL16.Support.raxml.support
```

As stated in the log above, CellPhy will output a tree with support values (in NEWICK format) that can be visualized using any tree viewer software. For you convenience, CellPhy also provides a script called *support-map.R* for plotting the support tree. This script is using the **ape** and **ggtree** r-packages. Usage:

```
$ ../script/support-map.R CRC24.VCF.GL16.Support.raxml.support Healthy

Generating support tree plot...
Done!
```

This call will generate a tree plot in PDF (*CRC24.VCF.GL16.Support.raxml.support.pdf*) and SVG (*CRC24.VCF.GL16.Support.raxml.support.svg*) formats. The result is shown in Figure 1.

---

5 Mapping mutations onto a phylogenetic tree

Cancer genomics studies are, for the most part, interested in understanding when "**driver**" mutations appeared in the malignant cell population. On this basis, we will next show how to map mutations onto a phylogenetic tree using CellPhy. Although the full VCF can be used, users are free to choose which mutations they wish to map onto the inferred phylogeny. In this tutorial, we will focus solely on a set of 15 exonic mutations from the original VCF.

Our input files will therefore consist of a "trimmed" VCF only carrying this subset of exonic mutations, together with the best tree and model estimates from our original CellPhy run (tree search).

```
$ head -n 5 CRC24.MutationsMap
#Chr    Position    GeneID
2   71042907    CLEC4F
2   142274377   LRP1B
3   33048242    GLB1
4   16764214    LDB2

$ bcftools view -T CRC24.MutationsMap CRC24.ToySet.vcf -O v -o CRC24.ToySet.Exonic.vcf

$ ../cellphy.sh RAXML --mutmap \
    --msa CRC24.ToySet.Exonic.vcf \
    --model CRC24.VCF.GL16.Tree.raxml.bestModel \
    --tree CRC24.VCF.GL16.Tree.raxml.bestTree \
    --opt-branches off --prefix CRC24.ToySet.Exonic.Mapped --threads 1
```

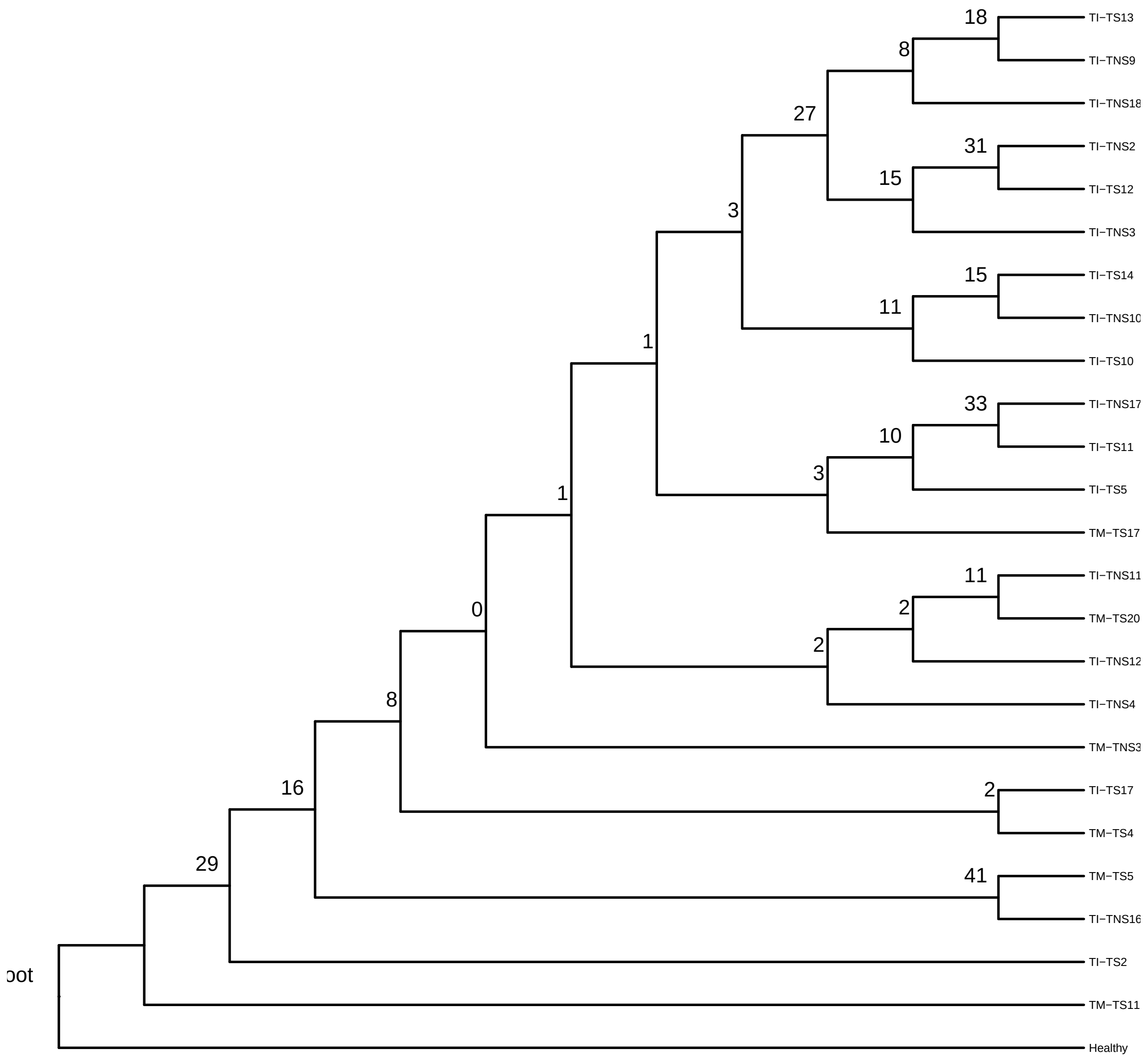

Figure 1: Phylogenetic tree of CRC24 dataset with bootstrap support.

```
Branch-labeled tree saved to: CRC24.ToySet.Exonic.Mapped.raxml.mutationMapTree
Per-branch mutation list saved to: CRC24.ToySet.Exonic.Mapped.raxml.mutationMapList
```

---

6 Visualizing the results

Once it's done, CellPhy will output 2 distinct files:.

- **(A)** *CRC24.ToySet.Exonic.Mapped.raxml.mutationMapTree*
  → Newick tree file with indexed branches
- **(B)** *CRC24.ToySet.Exonic.Mapped.raxml.mutationMapList*
  → Text file with the number and the list of mutations per branch

We can now use the *mutation-map.R* accompanying script to plot the mutations onto the phylogenetic tree. If you run this script wihtout any parameters, it will show a help message:

```
$ ../script/mutation-map.R

CellPhy - Mutation mapping plot - 22.07.2020
Created by: Alexey Kozlov, Joao M Alves, Alexandros Stamatakis & David Posada

Usage: ./mutation-map.R raxml.mutationMapTree raxml.mutationMapList Outgroup Output_prefix [geneIDs]

*Required files:
    -Tree
    -Mutation List
    -Outgroup name (comma-delimited list of taxa or NONE)
    -Output Prefix

*Optional:
    -Gene IDs (tab-delimited)
```

Now let's run it again but this time with the required parameters:

```
$ ../script/mutation-map.R \
    CRC24.ToySet.Exonic.Mapped.raxml.mutationMapTree \
    CRC24.ToySet.Exonic.Mapped.raxml.mutationMapList \
    Healthy \
    CRC24.ToySet.ExonicMutMap

Generating mutation-mapped tree plot...
Done!
```

If everything went as expected, you should have generated the following figure, in PDF format (*CRC24.ToySet.ExonicMutMap.pdf*), where the mutations are mapped onto the tree branches (**Figure 2**).

If you are interested in plotting the gene names instead, you can provide a tab-delimited file (as the one we used to subset our original VCF) linking the genomic position to its gene ID:

```
$ head -n 5 CRC24.MutationsMap
#Chr    Position    GeneID
2   71042907    CLEC4F
2   142274377   LRP1B
3   33048242    GLB1
4   16764214    LDB2
```

Afterwards, we can run *mutation-map.R* again, but changing the output prefix so that you don't overwrite the previous results:

```
$ ../script/mutation-map.R \
    CRC24.ToySet.Exonic.Mapped.raxml.mutationMapTree \
    CRC24.ToySet.Exonic.Mapped.raxml.mutationMapList \
    Healthy \
    CRC24.ToySet.ExonicMutMap-GeneID \
    CRC24.MutationsMap

Converting positions to GeneID...
Generating mutation-mapped tree plot...
Done!
```

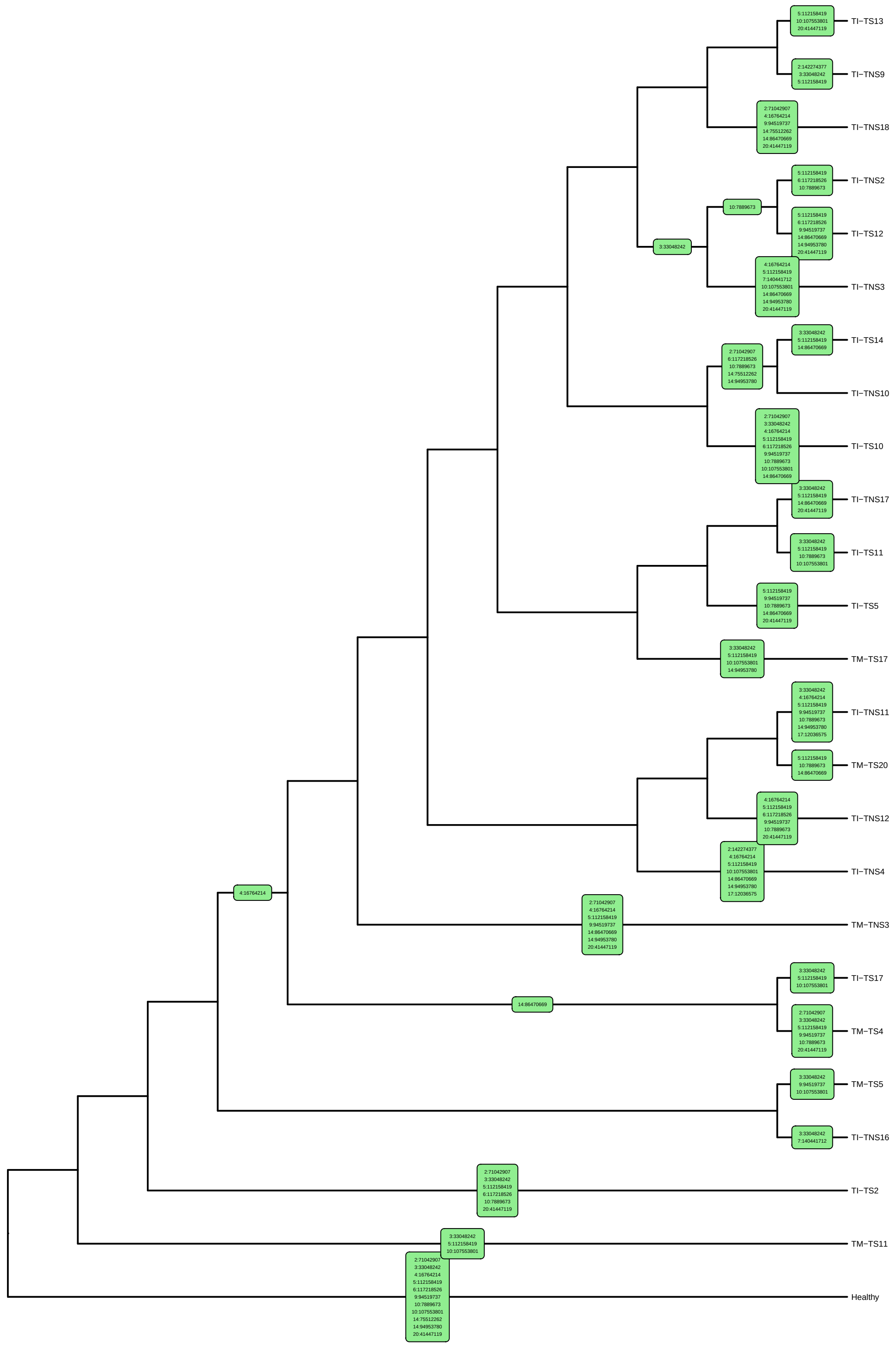

Figure 2: CRC24 phylogenetic tree with 15 exonic mutations mapped (genomic position)

You will notice that our tree now has the gene names displayed, instead of the genomic positions (Figure 3).

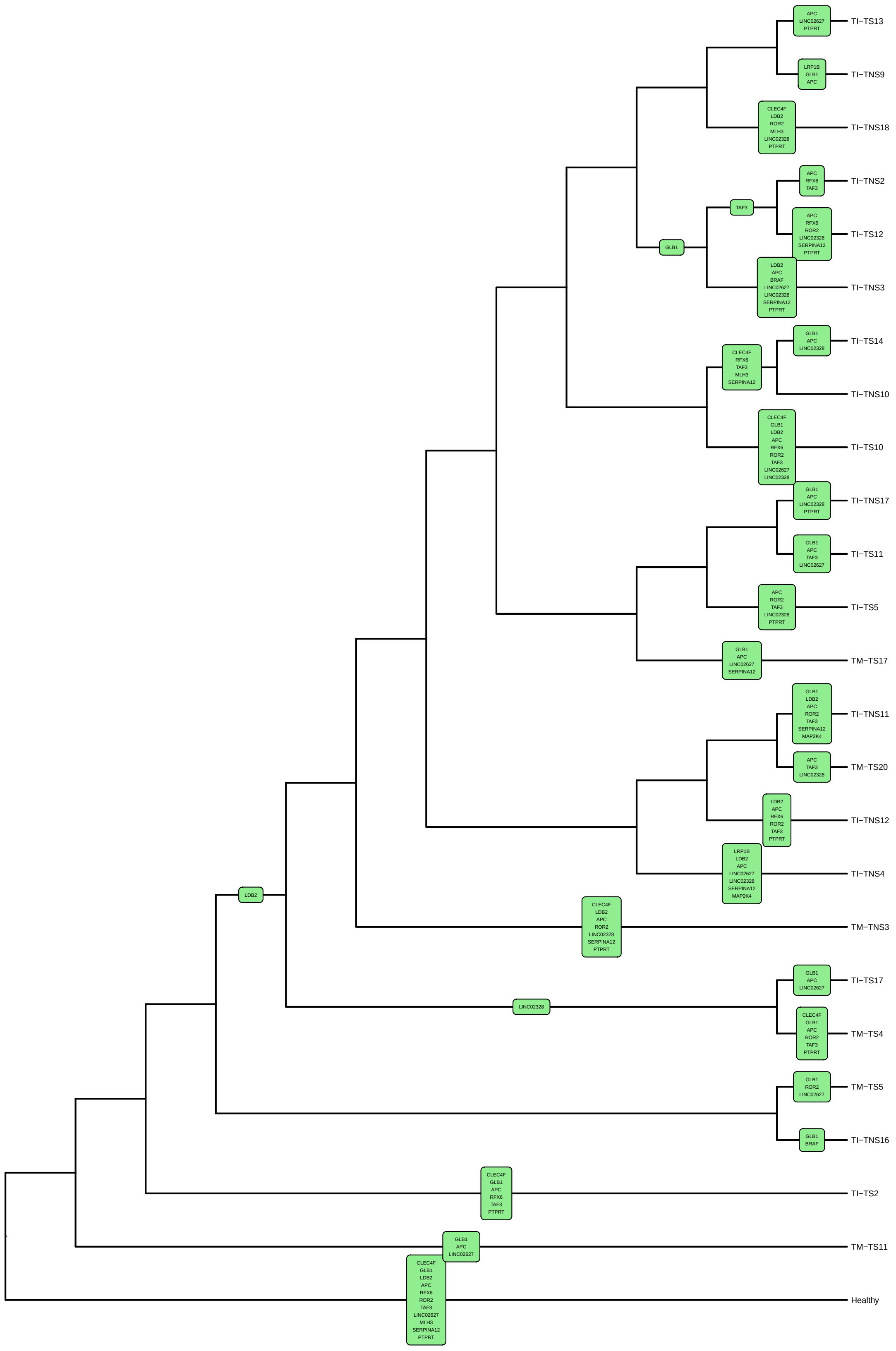
APC LINC02627 PTPRT
TI−TS13
LRP1B GLB1 APC
TI−TNS9
CLEC4F LDB2 ROR2 MLH3 LINC02328 PTPRT
TI−TNS18
APC RFX6 TAF3
TI−TNS2
TAF3
APC RFX6 ROR2 LINC02328 SERPINA12 PTPRT
TI−TS12
GLB1
LDB2 APC BRAF LINC02627 LINC02328 SERPINA12 PTPRT
TI−TNS3
GLB1 APC LINC02328
TI−TS14
CLEC4F RFX6 TAF3 MLH3 SERPINA12
TI−TNS10
CLEC4F GLB1 LDB2 APC RFX6 ROR2 TAF3 LINC02627 LINC02328
TI−TS10
GLB1 APC LINC02328 PTPRT
TI−TNS17
GLB1 APC TAF3 LINC02627
TI−TS11
APC ROR2 TAF3 LINC02328 PTPRT
TI−TS5
GLB1 APC LINC02627 SERPINA12
TM−TS17
GLB1 LDB2 APC ROR2 TAF3 SERPINA12 MAP2K4
TI−TNS11
APC TAF3 LINC02328
TM−TS20
LDB2 APC RFX6 ROR2 TAF3 PTPRT
TI−TNS12
LRP1B LDB2 APC LINC02627 LINC02328 SERPINA12 MAP2K4
TI−TNS4
LDB2
CLEC4F LDB2 APC ROR2 LINC02328 SERPINA12 PTPRT
TM−TNS3
GLB1 APC LINC02627
TI−TS17
LINC02328
CLEC4F GLB1 APC ROR2 TAF3 PTPRT
TM−TS4
GLB1 ROR2 LINC02627
TM−TS5
GLB1 BRAF
TI−TNS16
CLEC4F GLB1 APC RFX6 TAF3 PTPRT
TI−TS2
GLB1 APC LINC02627
TM−TS11
CLEC4F GLB1 LDB2 APC RFX6 ROR2 TAF3 LINC02627 MLH3 SERPINA12 PTPRT
Healthy

Figure 3: CRC24 phylogenetic tree with 15 exonic mutations mapped (gene names)