Implementation of the TRONCO package for TRanslational ONCOlogy

Marco Antoniotti[1,2], Giulio Caravagna[1], Luca De Sano[1], Alex Graudenzi[1], Giancarlo Mauri[1], Bud Mishra[3], and Daniele Ramazzotti[1]

[1]Dipartimento di Informatica Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano, Italy.
[2]Milan Center for Neuroscience, University of Milan-Bicocca, Milan, Italy.
[3]Courant Institute of Mathematical Sciences, New York University, New York, USA.

Abstract

Models of *cancer progression* provide insights on the order of accumulation of genetic alterations during cancer development. Algorithms to infer such models from the currently available mutational profiles collected from different cancer patiens (*cross-sectional data*) have been defined in the literature since late 90s. These algorithms differ in the way they extract a *graphical model* of the events modelling the progression, e.g., somatic mutations or copy-number alterations.

TRONCO is an R package for TRanslational ONcology which provides a serie of functions to assist the user in the analysis of cross-sectional genomic data and, in particular, it implements algorithms that aim to model cancer progression by means of the notion of selective advantage. These algorithms are proved to outperform the current state-of-the-art in the inference of cancer progression models. TRONCO also provides functionalities to load input cross-sectional data, set up the execution of the algorithms, assess the statistical confidence in the results and visualize the models.

Availability. Freely available at http://www.bioconductor.org/ under GPL license; project hosted at http://bimib.disco.unimib.it/ and https://github.com/BIMIB-DISCo/TRONCO.

Contact. tronco@disco.unimib.it

1 Introduction

In the last two decades many specific genes and genetic mechanisms involved in different types of cancer have been identified and yet our understanding of cancer progression is still largely elusive as it still faces fundamental challenges.

Concomitantly, a growing number of cancer-related genomic data have become available lately (see [5]). Thus, there now exists an urgent to leverage a number of sophisticated computational methods in biomedical research to elucidate the fast-growing biological datasets. Motivated by this state of affairs, we focus on the problem of *reconstructing progression models*

of cancer. In particular, we aim at inferring the plausible sequences of *genomic alterations* that, by a process of *accumulation*, selectively make a tumor fitter to survive, expand and diffuse (i.e., metastasize).

We developed a sequel of algorithms (see [4, 6]) which are implemented in the *TRanslational ONCOlogy* (TRONCO) package. Starting from cross-sectional genomic data, such algorithms aim at reconstructing a probabilistic progression model by inferring "selectivity relations", where a mutation in a gene A "selects" for a later mutation in a gene B. These relations are depicted in a combinatorial graph and resemble the way a mutation exploits its "*selective advantage*" to allow its host cells to expand clonally. Among other things, a selectivity relation implies a putatively invariant temporal structure among the genomic alterations (i.e., *events*) in a specific cancer type. In addition, a selectivity relation between a pair of events here signifies that the presence of the earlier genomic alteration (i.e., the *upstream event*) is advantageous in a Darwinian competition scenario raising the probability with which a subsequent advantageous genomic alteration (i.e., the *downstream event*) "survives" in the clonal evolution of the tumor (see [6]).

Notice that, in general, the inference of cancer progression models requires a complex data processing pipeline, as summarized in Figure 1. Initially, one collects *experimental data* (which could be accessible through publicly available repositories such as TCGA) and performs *genomic analyses* to derive profiles of, e.g., somatic mutations or Copy-Number Variations for each patient. Then, statistical analysis and biological priors are used to select events relevant to the progression - e.g., *driver mutations*. This complex pipeline can also include further statistics and priors to determine cancer subtypes and to generate *patterns of selective advantage* - , e.g, hypotheses of mutual exclusivity. Given these inputs, our algorithms (such as CAPRESE and CAPRI) can extract a progression model and assess *confidence* measures using various metrics based on non-parametric bootstrap and hypergeometric testing. *Experimental validation* concludes the pipeline. TRONCO provides support to all the steps of the pipeline.

2 Inference algorithms

TRONCO, provides a series of functions to support the user in each step of the pipeline, i.e., from data import, through data visualization and, finally to the inference of cancer progression models. Specifically, in the currenct version, TRONCO implements CAPRESE and CAPRI algorithms for cancer progression inference, which we briefly describe in the following.

Central to these algorithms, is Suppes' notion of *probabilistic causation*, which can be stated in the following terms: a selectivity relation between two observables i and j is said to hold if (1) i occurs earlier than j – *temporal priority* (TP) – and (2) if the probability of observing i raises the probability of observing j, i.e., $\mathcal{P}(j \mid i) > \mathcal{P}(j \mid \bar{i})$ – *probability raising* (PR). For the detailed description of the methods, we refer the reader to [4, 6].

CAncer PRogression Extraction with Single Edge. The *CAncer PRogression Extraction with Single Edges* algorithm, i.e., CAPRESE, extracts tree-based models of cancer progression with (*i*) multiple independent starting points and (*ii*) branches. The former models the emergence of different progressions as a result of the natural heterogeneity of cancer (cfr., [4]). The latter models the possibility of a clone to undergo positive selection by acquiring different mutations.

The inference of CAPRESE's models is driven by a *shrinkage* estimator of the confidence in the relation between pair of genes, which augments robustness to noise in the input data.

As shown in [4], CAPRESE is currently the state-of-the-art algorithm to infer tree cancer progression models, although its expressivity is limited to this kind of selective advantage

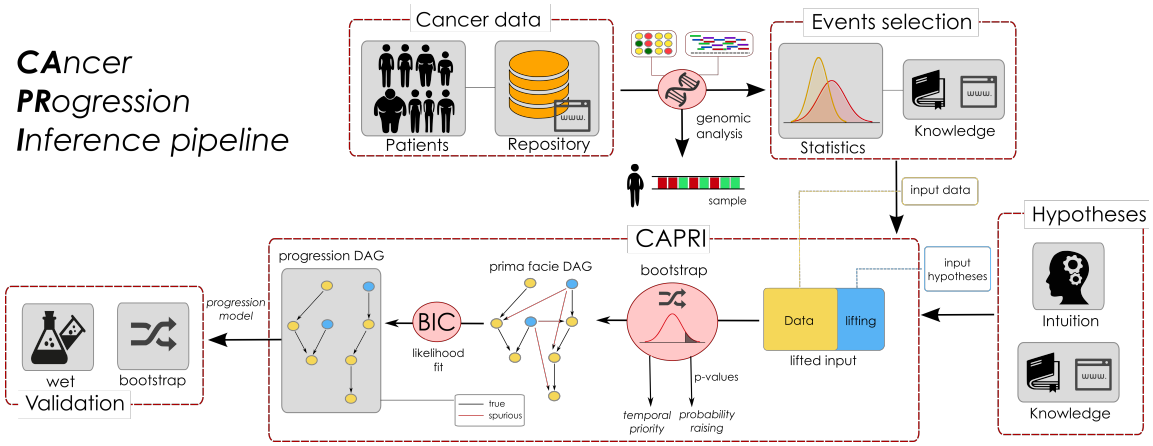

Figure 1: **Data processing pipeline for the cancer progression inference.** TRONCO implements a series of functions and algorithms to extract cancer progression models from cross-sectional data. Initially, one collects *experimental data* (which could be accessible through publicly available repositories such as TCGA) and performs *genomic analyses* to derive profiles of, e.g., somatic mutations or Copy-Number Variations for each patient. Then, statistical analysis and biological priors are used to select events relevant to the progression - e.g., *driver mutations*. This complex pipeline can also include further statistics and priors to determine cancer subtypes and to generate *patterns of selective advantage* - , e.g, hypotheses of mutual exclusivity. Given these inputs, our algorithms (such as CAPRESE and CAPRI) can extract a progression model and assess various *confidence* measures on its constituting relations such as non-parametric bootstrap and hypergeometric testing. *Experimental validation* concludes the pipeline, see [6].

models (cfr., [6]). Since this limitation is rather unappealing in analyzing cancer data, an improved algorithm to overcome it was sought in [6].

CAncer PRogression Inference. The *CAncer PRogression Inference* algorithm, i.e., CAPRI, extends tree models by allowing multiple predecessors of any common downstream event, thus allowing construction of directed acyclic graph (DAGs) progression models.

CAPRI performs maximum likelihood estimation for the progression model with constraints grounded in Supped' prima facile causality (cfr., [6]). In particular, the search space of the possible valid solutions is limited to the selective advantage relations where both TP and PR are verified and then, on this reduced search space, the likelihood fit is performed.

In [6], CAPRI was shown to be effective and polynomial in the size of the inputs.

3 Package implementation

In this section we will review the structure and implementation of the TRONCO package. For the sake of clarity, we will structure the description through the following functionalities that are implemented in the package.

- **Data import.** Functions for the importation of data both from flat files (e.g., MAF,

GISTIC) and from Web querying (e.g., cbioportal).

- **Data export and correctness.** Functions for the export and visualization of the imported data.
- **Data editing.** Functions for the preprocessing of the data in order to tidy them.
- **External utilities.** Functions for the interaction with external tools for the analysis of cancer subtypes or groups of mutually exclusive genes.
- **Inference algorithms.** In the current version of TRONCO, the CAPRESE and CAPRI algorithms are provided in a polinomial implementation.
- **Confidence estimation.** Functions for the statistical estimation of the confidence of the reconstructed models.
- **Visualization.** Functions for the visualization of both the input data and the results of the inference and of the confidence estimation.

Data import

The starting point of TRONCO analysis pipeline, is a dataset of genomics alterations (i.e., somatic mutations and copy number variations) which need to be imported as a TRONCO compliant data structure, i.e., a list R structure containing the required data both for the inference and the visualization. The data import functions take as input such genomic data and from them create a TRONCO compliant data structure.

The core of data import from flat files, is the function `import.genotypes(geno, event.type = ''variant'', color = ''Darkgreen'')`. This function imports a matrix of 0/1 alterations as a TRONCO compliant dataset. The input *"geno"* can be either a dataframe or a file name. In any case the dataframe or the table stored in the file must have a column for each altered gene and a rows for each sample. Colnames will be used to determine gene names, if data are loaded from file the first column will be assigned as rownames.

Besides this function, TRONCO implements data import from other file format such as MAF and GISTIC files as wrappers of the function `import.genotypes`. Specifically, the function `import.MAF(file, sep = ''\t'', is.TCGA = TRUE)` imports mutation profiles from a Manual Annotation Format (MAF) file. All mutations are aggregated as a unique event type labeled "Mutation" and are assigned a color accordingly to the default of function `import.genotypes`. If the input is in the TCGA MAF file format, the function also checks for multiple samples per patient and a warning is raised if any are found. Furthermore, the function `import.GISTIC(x)` transforms GISTIC scores for CNAs in a TRONCO compliant object. The input can be a matrix, with columns for each altered gene and rows for each sample; in this case colnames/rownames mut be provided. If the input is a character an attempt to load a table from file is performed. In this case the input table format should be constitent with TCGA data for focal CNA; there should hence be: one column for each sample, one row for each gene, a column Hugo_Symbol with every gene name and a column Entrez_Gene_Id with every genes Entrez ID. A valid GISTIC score should be any value of: "Homozygous Loss" (-2), "Heterozygous Loss" (-1), "Low-level Gain" (+1), "High-level Gain" (+2).

Finally, TRONCO also provides utilities for the query of genomic data from cbioportal. This is implemented in the function `cbio.query(cbio.study = NA, cbio.dataset = NA, cbio.profile = NA, genes)` which is a wrapper for the *CGDS* package. This can work either automatically, if one sets `cbio.study`, `cbio.dataset` and `cbio.profile`, or interactively. A list of genes to query with less than 900 entries should be provided. This function returns a list with two dataframes: the required genetic profile along with clinical data for the `cbio.study`. The output is also saved to disk as Rdata file. See also the cbioportal page at `http://www.cbioportal.org`.

The function `show(x, view = 10)` prints to console a short report of a dataset "x", which should be a TRONCO compliant dataset.

All the functions described in the following sections will assume as input a TRONCO compliant data structure.

Data export and correctness

TRONCO provides a series of function to explore the imported data and the inferred models. All these functions are named with the "as." prefix.

Concerning the imported data, the function `as.genotypes(x)` returns the 0/1 genotypes matrix. This function can be used in combination with the function `keysToNames(x, matrix)` to translate colnames to event names given the input matrix with colnames/rownames which represent genotypes keys. Also, functions to get the list of genes, events (i.e., each columns in the genotypes matrix, it differs from genes as the same genes of different types are considered different events), alterations (i.e., genes of different types are merged as 1 unique event), samples (i.e., patients or also single cells) and alteration types (see functions `as.genes(x, types = NA)`, `as.events(x, genes = NA, types = NA)`, `as.alterations(x, new.type = "Alteration", new.color = "khaki")`, `as.samples(x)` and `as.types(x, genes = NA)`).

Functions of this kind are also implemented to explore the results like the models that have been inferred (see `as.models(x, models = names(x$model))`), the reconstructions (see `as.adj.matrix(x, events = as.events(x), models = names(x$model),type = "fit")`), the considered patters (see `as.patterns(x)`) and the confidence (see `as.confidence(x, conf)`).

Similarly, a set of function to extract the cardinality of the compliant TRONCO data structure are defined (see `nevents(x, genes = NA, types = NA)`, `ngenes(x, types = NA)`, `npatterns(x)`, `nsamples(x)` and `ntypes(x)`).

Furthermore, functions to asses the correctness of the inputes are also provided. The function `is.compliant(x, err.fun = "[ERR]", stage = !(all(is.null(x$stages)) || all(is.na(x$stages))))` verifies the TRONCO data structure to be compliant with the standards. The function `consolidate.data(x, print = FALSE)` verifies if the input data are consolidated, i.e., if there are events with 0 or 1 probability or indistinguishable in terms of observations. Any indistinguishable event is returned by the function `duplicates(x)`.

Finally, TCGA specific functions are provided. `TCGA.multiple.samples(x)` checks if there are multiple sample in the input, while `TCGA.remove.multiple.samples(x)` remove them accordingly to TCGA barcodes naming.

Data editing

TRONCO provides a wide range of editing functions. We will describe some of them in the following, for a technical description we refer to the manual.

Removing and merging. A set of functions to remove items from the data is provided; such functions are named with the "delete." prefix. Specifically, it is possible to remove genes (`delete.gene(x, gene)`, events (`delete.event(x, gene, type)`), samples (`delete.samples(x, samples)`), types (`delete.type(x, type)`) as well as patterns (`delete.pattern(x, pattern)`) and inferred models (`delete.model(x)`). At the same time, it is possible to merge events (`merge.events(x, ..., new.event, new.type, event.color)`) and types (`merge.types(x, ..., new.type = "new.type", new.color = "khaki")`).

Binding. The purpose of the binding functions is to combine different datasets. The function `ebind(...)` combines events from one or more datasets, whose events need be defined over the

same set of samples while the function `sbind(...)` combines samples from one or more datasets, whose samples need to be defined over the same set of events. Samples and events of two dataset can also be intersected through the function `intersect.datasets(x, y, intersect.genomes = TRUE)`.

Changing and renaming. The functions `rename.gene(x, old.name, new.name)` and `rename.type(x, old.name, new.name)` can be used respectively to rename genes or alterations types. The function `change.color(x, type, new.color)` can be used to change the color associated to the specified alteration type.

Selecting and splitting. Genomics data usually involves a vast number of genes, the most of which is not relevant for cancer development (i.e., such as passenger mutations). For this reason, TRONCO implements the function `events.selection(x, filter.freq = NA, filter.in.names = NA,filter.out.names = NA)` which allows the selection of a set of genes to be analyzed. The selection can be performed by frequency and gene symbols. The 0 probability events can are removed by the function `trim(x)`. Moreover, the functions `samples.selection(x, samples)` and `ssplit(x, clusters, idx = NA)` respectively filters a dataset based on selected samples id and splits the dataset into groups (i.e., groups). This latter function can be used to analyze specific subtypes within a tumor.

External utilities

TRONCO permits the interaction with external tools to (*i*) reduce inter-tumor heterogeneity by cohort subtyping and (*ii*) detect fitness equivalent exclusive alterations. The first issue can be attacked by adopting clustering techniques to split the dataset in order to analyze each cluster subtype separately. Currently, TRONCO can export and inport data from [3] via the function `export.nbs.input(x, map_hugo_entrez, file = "tronco_to_nbs.mat")` and the previously described splitting functions.

To handle alterations with equivalent fitness, TRONCO implements the interaction with the method proposed by [1] through the functions `export.mutex(x, filename = "to_mutex", filepath = "./", label.mutation = "SNV", label.amplification = list("High-level Gain"), label.deletion = list("Homozygous Loss"))` and `import.mutex.groups(file, fdr = 0.2, display = TRUE)`. Such exclusivity groups can then be further added as patters (see the next section).

Inference algorithms

In the current version of TRONCO are implemented the algorithms CAPRESE [4] and CAPRI [6].

CAPRESE The CAPRESE algorithm [4] can be executed by the function named `tronco.caprese(data, lambda = 0.5, do.estimation = FALSE, silent = FALSE)` with "data" being a compliant TRONCO data structure. The parameter "lambda" can be used to tune the shrinkage-alike estimator adopted by CAPRESE, with the default being 0.5 as suggested in [4].

CAPRI The CAPRI algorithm [6] is executed by the function `tronco.capri(data, command = "hc", regularization = c("bic", "aic"), do.boot = TRUE, nboot = 100, pvalue = 0.05, min.boot = 3, min.stat = TRUE, boot.seed = NULL, do.estimation = FALSE, silent = FALSE)` with "data" being a TRONCO compliant data structure. The parameters "command" and "regularization" allows respectively to choose the heuristic search to be performed to fit the

network and the regularizer to be used in the likelihood fit (see [6]). CAPRI can be also executed without the bootstrap preprocessing step by the parameter "do.boot"; this is discouraged, but can speed up the execution with big input datasets.

As discussed in [6], CAPRI constrains the seach space using Suppes' prima facie conditions which lead to a subset of possible valid selective advantage relations which are then evaluated by the likelihood fit. Although uncommon, it may so happen (especially when pattern are given as input) that such a resulting prima facie graphical structure may still contain cycles. When this happens, the cycles are removed through the heuristic algorithm implemented in `remove.cycles(adj.matrix, weights.temporal.priority, weights.matrix, not.ordered, hypotheses = NA, silent)`. The function takes as input a set of weights in term of confidence for any selective advantage valid edge, ranks all the valid edges in increasing confidence levels and, starting from the less confident, goes through each edge removing the ones that can break the cycles.

Patterns CAPRI allows for the input of patterns, i.e., group of events which express possible selective advantage relations. Such patters are given as input using the function `hypothesis.add(data, pattern.label, lifted.pattern, pattern.effect = "*", pattern.cause = "*")`. This function is wrapped within the functions `hypothesis.add.homologous(x, pattern.cause = "*", pattern.effect = "*", genes = as.genes(x), FUN = OR)` and `hypothesis.add.group(x, FUN, group, pattern.cause = "*", pattern.effect = "*", dim.min = 2, dim.max = length(group), min.prob = 0)` which, respectively, allow the addition of analogous patterns (i.e., patterns involving the same gene of different types) and patterns involving a specified group of genes. In the current version of TRONCO, the implemented possible patterns are the ones expressed using the boolean operators AND, OR and XOR (functions `AND(...)`, `OR(...)` and `XOR(...)`).

Confidence estimation

To asses the confidence of any selectivity relations TRONCO implements non-parametric and statistical bootstrap. For the non-parametric bootstrap, each event row is uniformly sampled with repetitions from the input genotype and then, on such an input, the inference algorithms are performed. The assesment concludes after K repetitions (e.g., $K = 100$). Similarly, for CAPRI, a statistical bootstrap is provided: in this case the input dataset is kept fixed, but different seeds for the statistical procedures are sampled (see, e.g., [2] for an overview of these methods). The bootstrap is implemented in the function `tronco.bootstrap(reconstruction, type = "non-parametric", nboot = 100, verbose = FALSE)` where "reconstruction" is a compliant object obtained by the inference by one of the implemented algorithms.

Visualization

During the development of the TRONCO package, a lot of attention was paid to the visualization features which are crucial for the understanding of biological results. Shown below is an overview of the main features; for a detailed description of each function, please refer to the manual.

OncoPrint. OncoPrints are compact means of visualizing distinct genomic alterations, including somatic mutations, copy number alterations, and mRNA expression changes across a set of cases. They are extremely useful for visualizing gene set and pathway alterations across a set of cases, and for visually identifying trends, such as trends in mutual exclusivity or co-occurence between gene pairs within a gene set. Individual genes are represented as rows, and individual cases or patients are represented as columns. See `http://www.cbioportal.org/`.

The function `oncoprint` provides such visualizations with a TRONCO compliant data structure as input. The function `oncoprint.cbio` exports the input for the cbioportl visualization, see `http://www.cbioportal.org/public-portal/oncoprinter.jsp`.

It is also possible to annotate a description (`annotate.description(x, label)`) and tumor stages (`annotate.stages(x, stages, match.TCGA.patients = FALSE)`) to any oncoprint.

Reconstruction. The inferred models can be displayed by the function `tronco.plot`. The features included in the plots are multiple, such as the choice of the regularizer(s), editing font of nodes and edges, scaling nodes' size in terms of estimated marginal probabilities, annotating the pathway of each gene and displaying the estimated confidence of each edge. We refer to the manual for a detailed description.

Reports. Finally, report utilities are provided. The function `genes.table.report(x, name, dir = getwd(), maxrow = 33, font = 10, height = 11, width = 8.5, fill = "lightblue")` can be used to generate LaTeX code to be used as report, while `genes.table.plot(x, name, dir = getwd())` generates reports histograms.

4 Use case of TRONCO

In this Section, we will present a case study for the usage of the TRONCO package based on the work presented in [6].

Events selection

We will start by loading the TRONCO package in R along with an example *"dataset"* that comes within the package.

```
> library(TRONCO)
> data(aCML)
> hide.progress.bar <<- TRUE
```

We then use the function `show` to get a short summary of the aCML dataset that has just been loaded.

```
> show(aCML)
Description: CAPRI - Bionformatics aCML data.
Dataset: n=64, m=31, |G|=23.
Events (types): Ins/Del, Missense point, Nonsense Ins/Del, Nonsense point.
Colors (plot): darkgoldenrod1, forestgreen, cornflowerblue, coral.
Events (10 shown):
 gene 4 : Ins/Del TET2
 gene 5 : Ins/Del EZH2
 gene 6 : Ins/Del CBL
 gene 7 : Ins/Del ASXL1
 gene 29 : Missense point SETBP1
 gene 30 : Missense point NRAS
 gene 31 : Missense point KRAS
 gene 32 : Missense point TET2
 gene 33 : Missense point EZH2
 gene 34 : Missense point CBL
Genotypes (10 shown):
```

```
          gene 4 gene 5 gene 6 gene 7 gene 29 gene 30 gene 31 gene 32 gene 33 gene 34
patient 1      0      0      0      0       1       0       0       0       0       0
patient 2      0      0      0      0       1       0       0       0       0       1
patient 3      0      0      0      0       1       1       0       0       0       0
patient 4      0      0      0      0       1       0       0       0       0       1
patient 5      0      0      0      0       1       0       0       0       0       0
patient 6      0      0      0      0       1       0       0       0       0       0
```

Using the function `as.events`, we can have a look at the events in the dataset.

```
> as.events(aCML)
         type               event
gene 4   "Ins/Del"          "TET2"
gene 5   "Ins/Del"          "EZH2"
gene 6   "Ins/Del"          "CBL"
gene 7   "Ins/Del"          "ASXL1"
gene 29  "Missense point"   "SETBP1"
gene 30  "Missense point"   "NRAS"
gene 31  "Missense point"   "KRAS"
gene 32  "Missense point"   "TET2"
gene 33  "Missense point"   "EZH2"
gene 34  "Missense point"   "CBL"
gene 36  "Missense point"   "IDH2"
gene 39  "Missense point"   "SUZ12"
gene 40  "Missense point"   "SF3B1"
gene 44  "Missense point"   "JARID2"
gene 47  "Missense point"   "EED"
gene 48  "Missense point"   "DNMT3A"
gene 49  "Missense point"   "CEBPA"
gene 50  "Missense point"   "EPHB3"
gene 51  "Missense point"   "ETNK1"
gene 52  "Missense point"   "GATA2"
gene 53  "Missense point"   "IRAK4"
gene 54  "Missense point"   "MTA2"
gene 55  "Missense point"   "CSF3R"
gene 56  "Missense point"   "KIT"
gene 66  "Nonsense Ins/Del" "WT1"
gene 69  "Nonsense Ins/Del" "RUNX1"
gene 77  "Nonsense Ins/Del" "CEBPA"
gene 88  "Nonsense point"   "TET2"
gene 89  "Nonsense point"   "EZH2"
gene 91  "Nonsense point"   "ASXL1"
gene 111 "Nonsense point"   "CSF3R"
```

These events account for alterations in the following genes.

```
> as.genes(aCML)
 [1] "TET2"   "EZH2"   "CBL"    "ASXL1"  "SETBP1" "NRAS"   "KRAS"   "IDH2"   "SUZ12"
   "SF3B1"  "JARID2" "EED"    "DNMT3A" "CEBPA"  "EPHB3"  "ETNK1"  "GATA2"  "IRAK4"
     "MTA2"   "CSF3R"
[21] "KIT"    "WT1"    "RUNX1"
```

Now we take a look at the alterations of only the gene `SETBP1` across the samples.

```
> as.gene(aCML, genes='SETBP1')
           Missense point SETBP1
patient 1                      1
patient 2                      1
patient 3                      1
patient 4                      1
patient 5                      1
patient 6                      1
patient 7                      1
patient 8                      1
patient 9                      1
patient 10                     1
patient 11                     1
patient 12                     1
patient 13                     1
patient 14                     1
patient 15                     0
patient 16                     0
patient 17                     0
patient 18                     0
patient 19                     0
patient 20                     0
patient 21                     0
patient 22                     0
patient 23                     0
patient 24                     0
patient 25                     0
patient 26                     0
patient 27                     0
patient 28                     0
patient 29                     0
patient 30                     0
patient 31                     0
patient 32                     0
patient 33                     0
patient 34                     0
patient 35                     0
patient 36                     0
patient 37                     0
patient 38                     0
patient 39                     0
patient 40                     0
patient 41                     0
patient 42                     0
patient 43                     0
patient 44                     0
patient 45                     0
patient 46                     0
patient 47                     0
patient 48                     0
```

```
patient 49                    0
patient 50                    0
patient 51                    0
patient 52                    0
patient 53                    0
patient 54                    0
patient 55                    0
patient 56                    0
patient 57                    0
patient 58                    0
patient 59                    0
patient 60                    0
patient 61                    0
patient 62                    0
patient 63                    0
patient 64                    0
```

We consider a subset of all the genes in the dataset to be involved in patters based on the support we found in the literature. See [6] as a reference.

```
> gene.hypotheses = c('KRAS', 'NRAS', 'IDH1', 'IDH2', 'TET2', 'SF3B1', 'ASXL1')
```

Regardless from which types of mutations we include, we select only the genes which appear alterated in at least 5% of the patients. Thus, we first transform the dataset into *"Alteration"* (i.e., by collapsing all the event types for the same gene), and then we consider only these events from the original dataset.

```
> alterations = events.selection(as.alterations(aCML), filter.freq = .05)
*** Aggregating events of type(s) Ins/Del, Missense point, Nonsense Ins/Del, Nonsense point
in a unique event with label "Alteration".
Dropping event types Ins/Del, Missense point, Nonsense Ins/Del, Nonsense point for 23 genes.
*** Binding events for 2 datasets.
*** Events selection: #events=23, #types=1 Filters freq|in|out = {TRUE, FALSE, FALSE}
Minimum event frequency: 0.05 (3 alterations out of 64 samples).
Selected 7 events.

Selected 7 events, returning.
```

We now show a plot of the selected genes. Note that this plot has no title as by default the function `events.selection` does not add any. The resulting figure is shonw in 2.

```
> dummy = oncoprint(alterations,font.row=12,cellheight=20,cellwidth=4)
*** Oncoprint for ""
with attributes: stage=FALSE, hits=TRUE
Sorting samples ordering to enhance exclusivity patterns.
```

Adding Hypotheses

We now create the `dataset` to be used for the inference of the progression model. We consider the original dataset and from it we select all the genes whose mutations are occurring at least 5% of the times together with any gene involved in any hypothesis. To do so, we use the parameter `filter.in.names` as shown below.

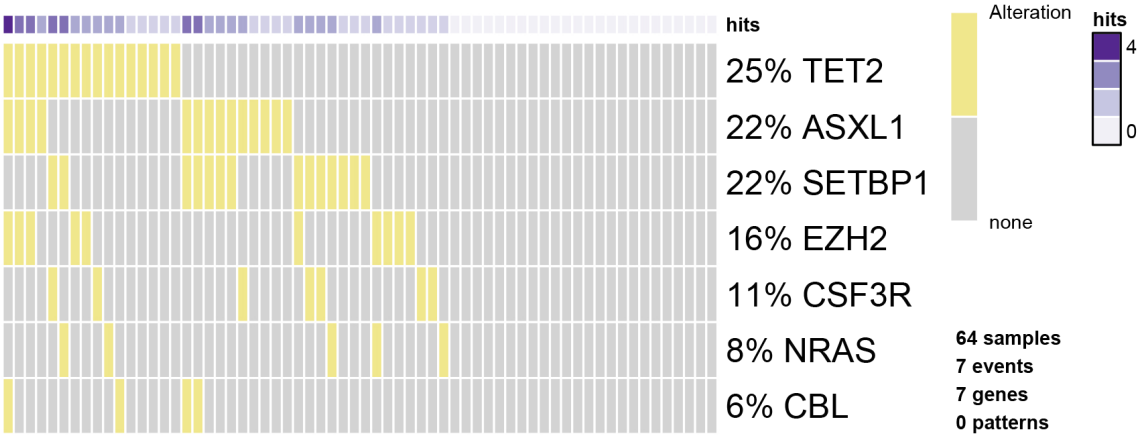

Figure 2: **Oncoprint function in** TRONCO**.** Result of the oncoprint function in TRONCO on the aCML dataset.

```
> hypo = events.selection(aCML, filter.in.names=c(as.genes(alterations), gene.hypotheses))
*** Events selection: #events=31, #types=4 Filters freq|in|out = {FALSE, TRUE, FALSE}
[filter.in] Genes hold: TET2, EZH2, CBL, ASXL1, SETBP1 ...  [10/14 found].
Selected 17 events, returning.
> hypo = annotate.description(hypo, 'CAPRI - Bionformatics aCML data (selected events)')
```

We show a new oncoprint of this latest dataset where we annotate the genes in `gene.hypotheses` in order to identify them 3. The sample names are also shown.

```
> dummy = oncoprint(hypo, gene.annot = list(priors= gene.hypotheses), sample.id = T, font.row=12,
font.column=5, cellheight=20, cellwidth=4)
*** Oncoprint for "CAPRI - Bionformatics aCML data (selected events)"
with attributes: stage=FALSE, hits=TRUE
Sorting samples ordering to enhance exclusivity patterns.
Annotating genes with RColorBrewer color palette Set1 .
```

We now also add the hypotheses that are described in CAPRI's manuscript. Hypothesis of hard exclusivity (XOR) for NRAS/KRAS events (Mutation). This hypothesis is tested against all the events in the dataset.

```
> hypo = hypothesis.add(hypo, 'NRAS xor KRAS', XOR('NRAS', 'KRAS'))
```

We then try to include also a soft exclusivity (OR) pattern but, since its *"signature"* is the same of the hard one just included, it will not be included. The code below is expected to result in an error.

```
> hypo = hypothesis.add(hypo, 'NRAS or KRAS',  OR('NRAS', 'KRAS'))
Error in hypothesis.add(hypo, "NRAS or KRAS", OR("NRAS", "KRAS")) :
  [ERR] Pattern duplicates Pattern NRAS xor KRAS.
```

To better highlight the perfect (hard) exclusivity among NRAS/KRAS mutations, one can examine further their alterations. See Figure 4.

Figure 3: **Annotated oncoprint.** Result of the oncoprint function on the selected dataset in TRONCO with annotations.

```
> dummy = oncoprint(events.selection(hypo, filter.in.names = c('KRAS', 'NRAS')),font.row=12,
cellheight=20, cellwidth=4)
*** Events selection: #events=18, #types=4 Filters freq|in|out = {FALSE, TRUE, FALSE}
[filter.in] Genes hold: KRAS, NRAS ...  [2/2 found].
Selected 2 events, returning.
*** Oncoprint for ""
with attributes: stage=FALSE, hits=TRUE
Sorting samples ordering to enhance exclusivity patterns.
```

We repeated the same analysis as before for other hypotheses and for the same reasons, we will include only the hard exclusivity pattern.

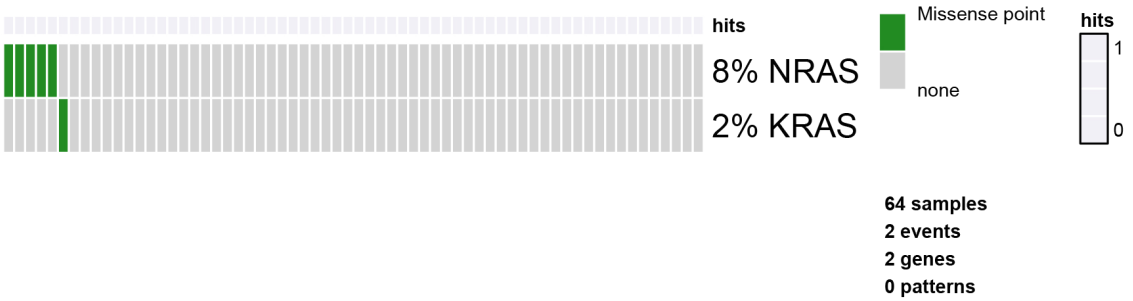

Figure 4: **RAS oncoprint.** Result of the oncoprint function in TRONCO for only the RAS genes to better show their hard exclusivity pattern.

```
> hypo = hypothesis.add(hypo, 'SF3B1 xor ASXL1', XOR('SF3B1', OR('ASXL1')), '*')
> hypo = hypothesis.add(hypo, 'SF3B1 or ASXL1', OR('SF3B1', OR('ASXL1')), '*')
Error in hypothesis.add(hypo, "SF3B1 or ASXL1", OR("SF3B1", OR("ASXL1")),  :
  [ERR] Pattern duplicates Pattern SF3B1 xor ASXL1.
```

Finally, we now repeat the same for genes TET2 and IDH2. In this case 3 events for the gene TET2 are present, that is *"Ins/Del"*, *"Missense point"* and *"Nonsense point"*. For this reason, since we are not specifying any subset of such events to be considered, all TET2 alterations are used. Since the events present a perfect hard exclusivity, their patters will be included as an XOR. See Figure 5.

```
> as.events(hypo, genes = 'TET2')
        type             event
gene 4  "Ins/Del"        "TET2"
gene 32 "Missense point" "TET2"
gene 88 "Nonsense point" "TET2"
> hypo = hypothesis.add(hypo, 'TET2 xor IDH2', XOR('TET2', 'IDH2'), '*')
> hypo = hypothesis.add(hypo,  'TET2 or IDH2', OR('TET2', 'IDH2'), '*')
> dummy = oncoprint(events.selection(hypo, filter.in.names = c('TET2', 'IDH2')),font.row=12,
cellheight=20,cellwidth=4)
*** Events selection: #events=21, #types=4 Filters freq|in|out = {FALSE, TRUE, FALSE}
[filter.in] Genes hold: TET2, IDH2 ...  [2/2 found].
Selected 4 events, returning.
*** Oncoprint for ""
with attributes: stage=FALSE, hits=TRUE
Sorting samples ordering to enhance exclusivity patterns.
```

We now finally add any possible group of homologous events. For any gene having more than one event associated we also add a soft exclusivity pattern among them.

```
> hypo = hypothesis.add.homologous(hypo)
*** Adding hypotheses for Homologous Patterns
 Genes: TET2, EZH2, CBL, ASXL1, CSF3R
 Function: OR
 Cause: *
 Effect: *
```

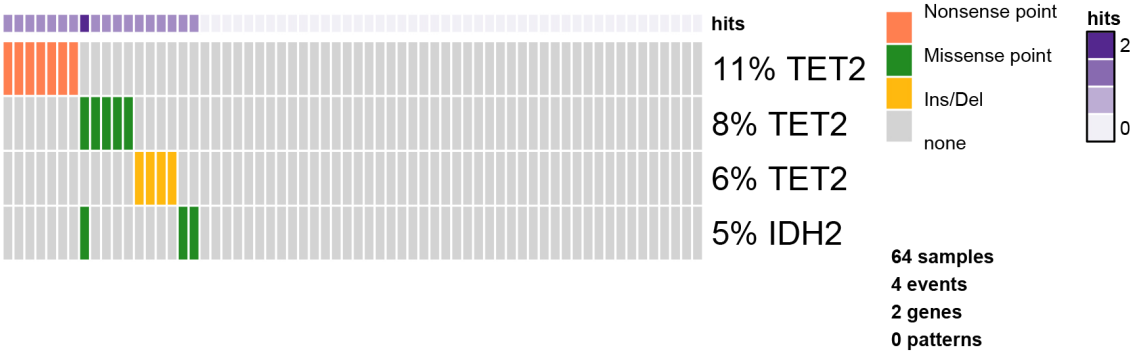

Figure 5: **TET/IDH2 oncoprint.** Result of the oncoprint function in TRONCO for only the TET/IDH2 genes.

```
Hypothesis created for all possible gene patterns.
```

The final dataset that will be given as input to CAPRI is now finally shown. See Figure 6.

```
> dummy = oncoprint(hypo, gene.annot = list(priors= gene.hypotheses), sample.id = T, font.row=10,
font.column=5, cellheight=15, cellwidth=4)
*** Oncoprint for "CAPRI - Bionformatics aCML data (selected events)"
with attributes: stage=FALSE, hits=TRUE
Sorting samples ordering to enhance exclusivity patterns.
Annotating genes with RColorBrewer color palette Set1 .
```

Model reconstruction

We next infer the model by running CAPRI algorithm with its default parameters: we use both AIC and BIC as regularizators, Hill-climbing as heuristic search of the solutions and exhaustive bootstrap (nboot replicates or more for Wilcoxon testing, i.e., more iterations can be performed if samples are rejected), p-value set at 0.05. We set the seed for the sake of reproducibility.

```
> model = tronco.capri(hypo, boot.seed = 12345, nboot=10)
*** Checking input events.
*** Inferring a progression model with the following settings.
Dataset size: n = 64, m = 26.
Algorithm: CAPRI with "bic, aic" regularization and "hc" likelihood-fit strategy.
Random seed: 12345.
Bootstrap iterations (Wilcoxon): 10.
exhaustive bootstrap: TRUE.
p-value: 0.05.
minimum bootstrapped scores: 3.
*** Bootstraping selective advantage scores (prima facie).
Evaluating "temporal priority" (Wilcoxon, p-value 0.05)
Evaluating "probability raising" (Wilcoxon, p-value 0.05)
*** Loop detection found loops to break.
```

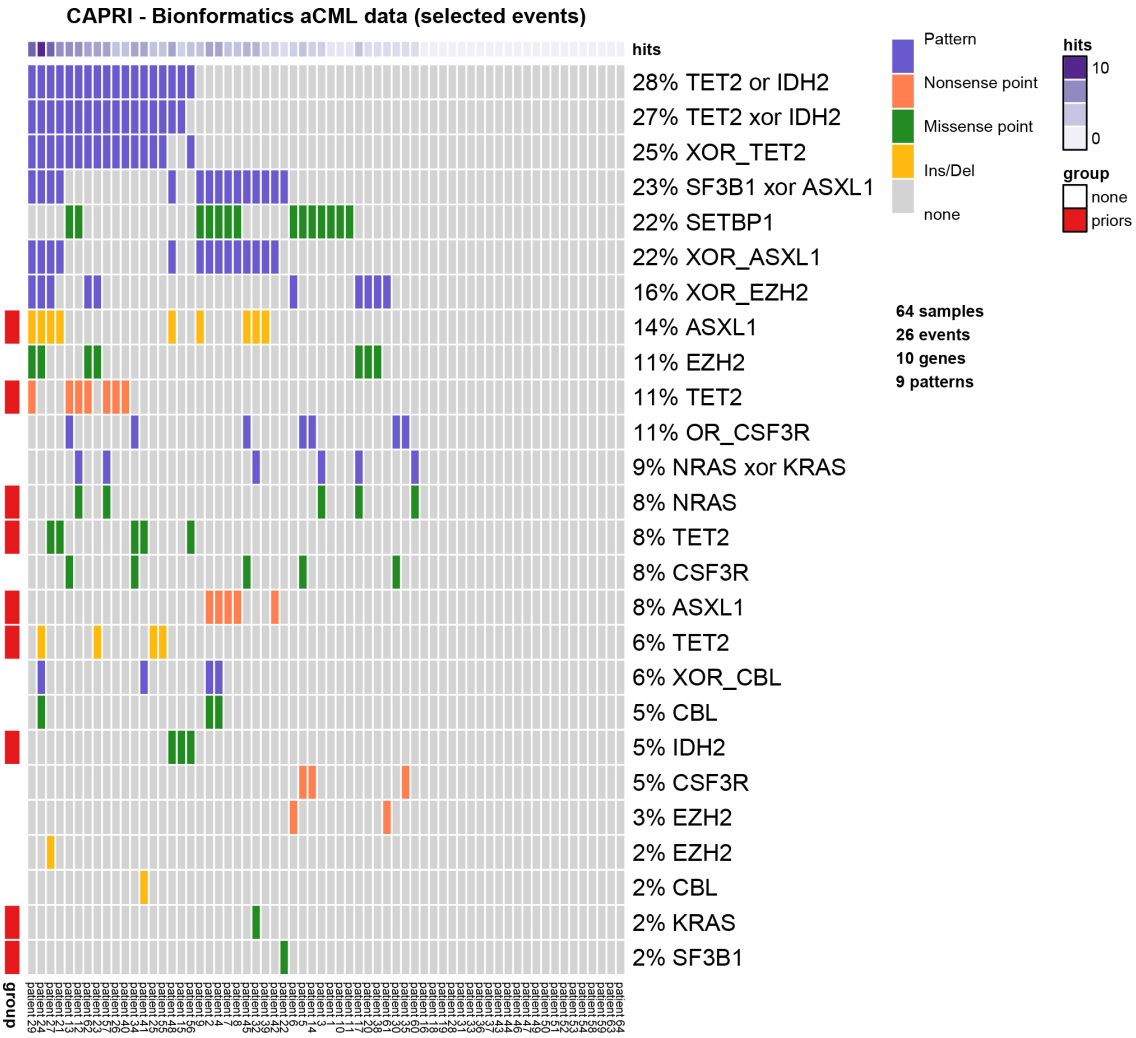

Figure 6: **Final dataset for CAPRI.** Result of the oncoprint function in TRONCO on the dataset used in [6].

```
Removed 26 edges out of 68 (38%)
*** Performing likelihood-fit with regularization bic.
*** Performing likelihood-fit with regularization aic.
The reconstruction has been successfully completed in 00h:00m:02s
```

We then plot the model inferred by CAPRI with BIC as a regularizer and we set some parameters to get a good plot; the confidence of each edge is shown both in terms of temporal priority and probability raising (selective advantage scores) and hypergeometric testing (statistical relevance of the dataset of input). See Figure 7.

```
> tronco.plot(model, fontsize = 13, scale.nodes = .6, regularization="bic",
confidence = c('tp', 'pr', 'hg'), height.logic = 0.25, legend.cex = .5,
pathways =  list(priors= gene.hypotheses), label.edge.size=5)
*** Expanding hypotheses syntax as graph nodes:
*** Rendering graphics
Nodes with no incoming/outgoing edges will not be displayed.
Annotating nodes with pathway information.
Annotating pathways with RColorBrewer color palette Set1 .
Adding confidence information: tp, pr, hg
RGraphviz object prepared.
Plotting graph and adding legends.
```

Bootstrapping data

Finally, we perform non-parametric bootstrap as a further estimation of the confidence in the inferred results. See Figure 8.

```
> model.boot = tronco.bootstrap(model, nboot=10)
Executing now the bootstrap procedure, this may take a long time...
Expected completion in approx. 00h:00m:03s
*** Using 7 cores via "parallel"

*** Reducing results

Performed non-parametric bootstrap with 10 resampling and 0.05 as pvalue
for the statistical tests.

> tronco.plot(model.boot, fontsize = 13, scale.nodes = .6, regularization="bic", confidence=c('npb'),
height.logic = 0.25, legend.cex = .5, pathways = list(priors= gene.hypotheses), label.edge.size=10)
*** Expanding hypotheses syntax as graph nodes:
*** Rendering graphics
Nodes with no incoming/outgoing edges will not be displayed.
Annotating nodes with pathway information.
Annotating pathways with RColorBrewer color palette Set1 .
Adding confidence information: npb
RGraphviz object prepared.
Plotting graph and adding legends.
```

We now conclude this analysis with an example of inference with the CAPRESE algorithm. As CAPRESE does not consider any patter as input, we use the dataset shown in Figure 3. These results are shown in Figure 9.

```
> model.boot.caprese = tronco.bootstrap(tronco.caprese(hypo))
*** Checking input events.
*** Inferring a progression model with the following settings.
Dataset size: n = 64, m = 17.
Algorithm: CAPRESE with shrinkage coefficient: 0.5.
The reconstruction has been successfully completed in 00h:00m:00s
Executing now the bootstrap procedure, this may take a long time...
Expected completion in approx. 00h:00m:00s
```

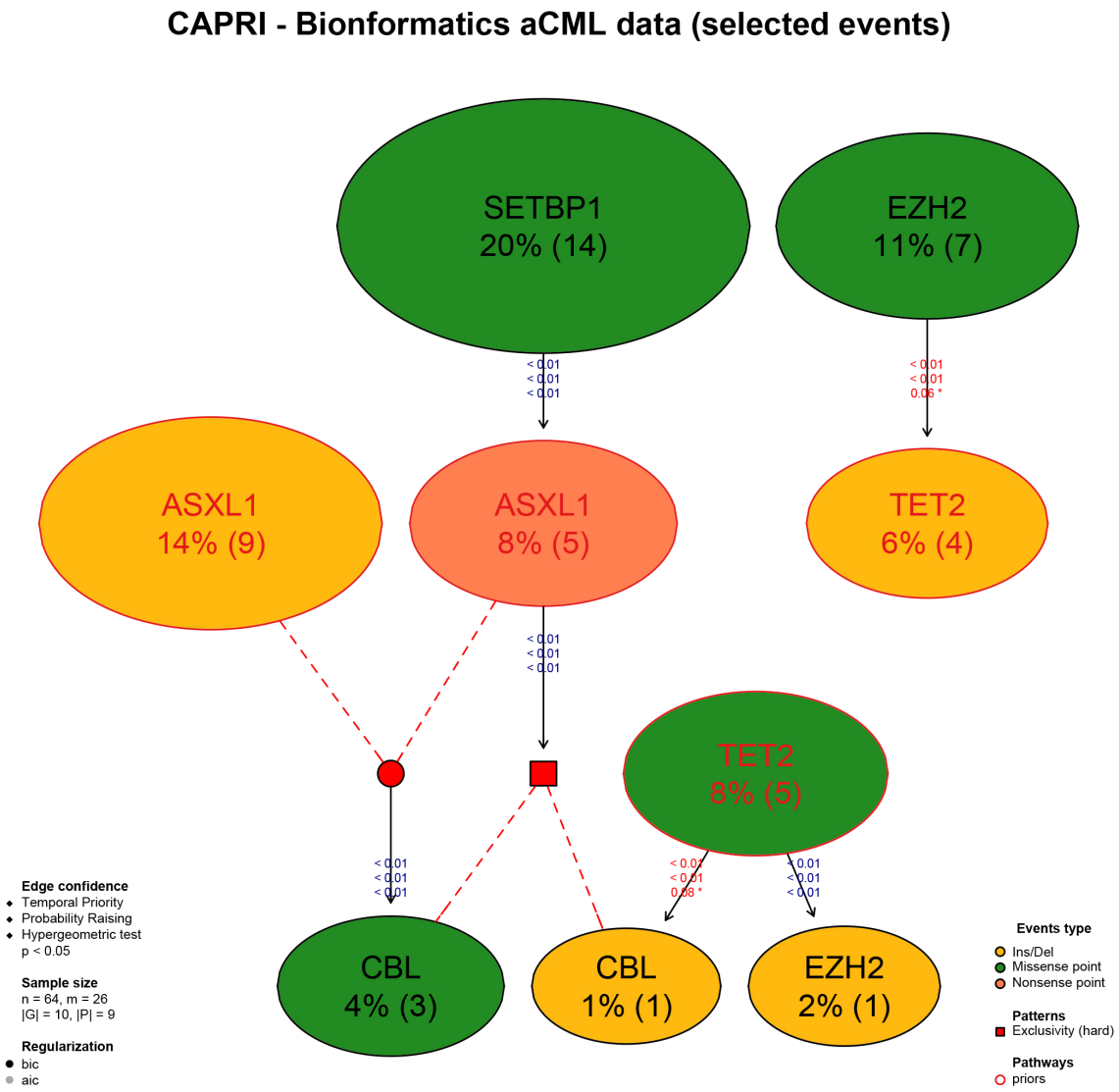

Figure 7: **Reconstruction by CAPRI.** Result of the reconstruction by CAPRI on the input dataset.

```
Performed non-parametric bootstrap with 100 resampling and 0.5
as shrinkage parameter.

> tronco.plot(model.boot.caprese, fontsize = 13, scale.nodes = .6, confidence=c('npb'),
height.logic = 0.25, legend.cex = .5, pathways = list(priors= gene.hypotheses), label.edge.size=10,
legend.pos="top")
```

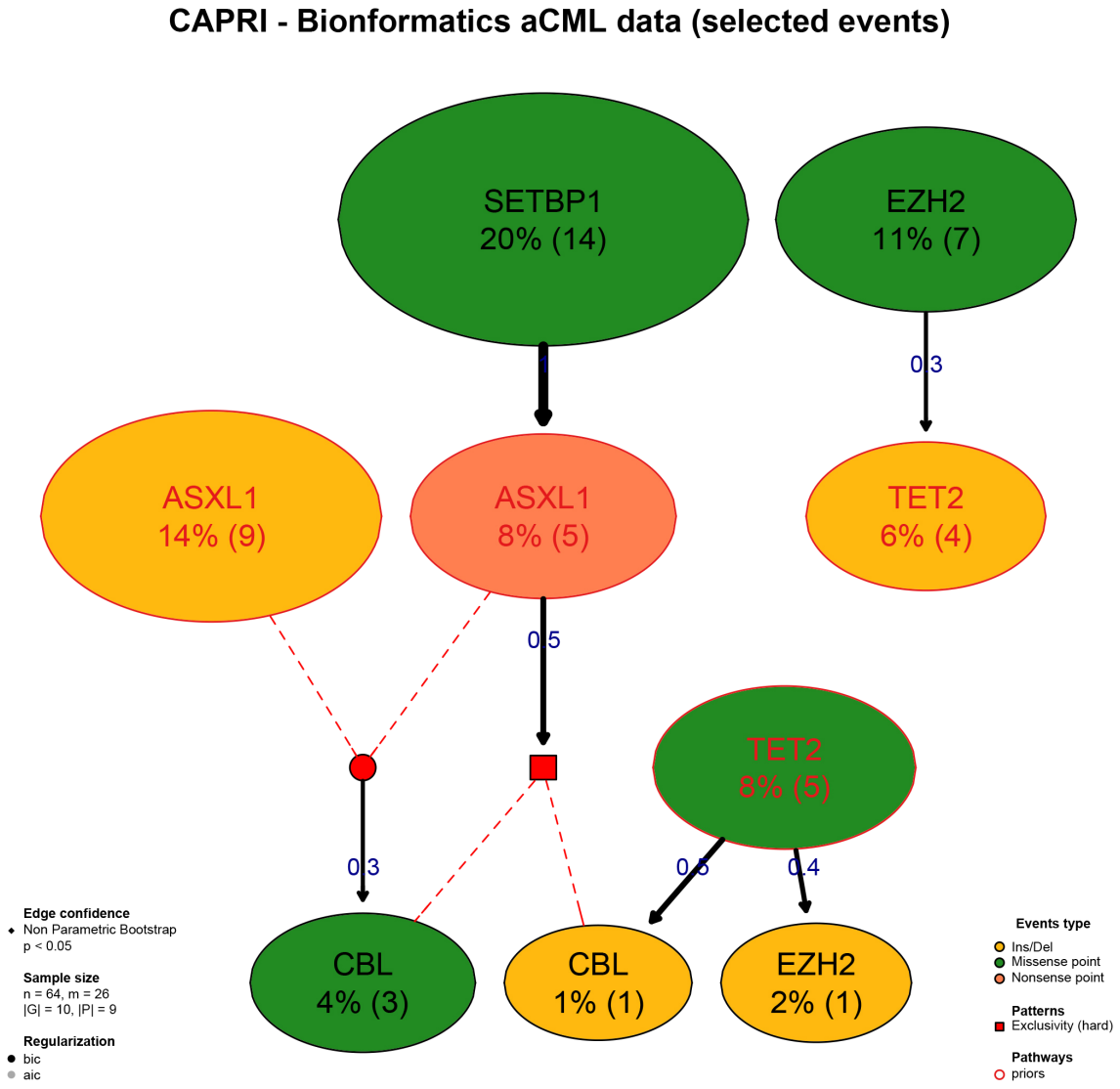

Figure 8: **Reconstruction by CAPRI and Bootstrap.** Result of the reconstruction by CAPRI on the input dataset with the assesment by non-parametric bootstrap.

```
*** Expanding hypotheses syntax as graph nodes:
*** Rendering graphics
Nodes with no incoming/outgoing edges will not be displayed.
Annotating nodes with pathway information.
Annotating pathways with RColorBrewer color palette Set1 .
Adding confidence information: npb
```

```
RGraphviz object prepared.
Plotting graph and adding legends.
```

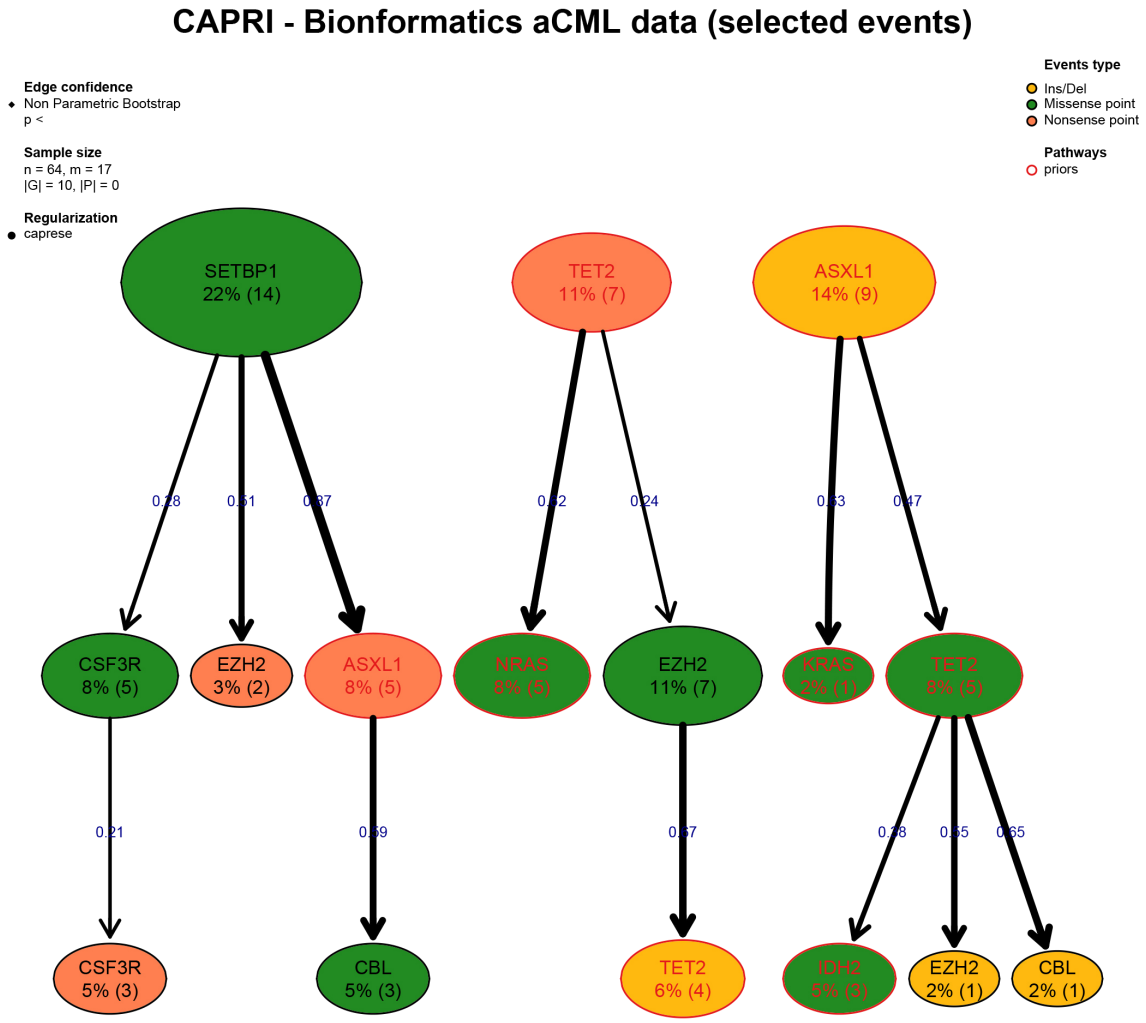

Figure 9: **Reconstruction by CAPRESE and Bootstrap.** Result of the reconstruction by CAPRESE on the input dataset with the assesment by non-parametric bootstrap.

5 Conclusions

We have described TRONCO, an R package that provides sequels of state-of-the-art techniques to support the user during the analysis of cross-sectional genomic data with the aim of un-

derstanding cancer evolution. In the currenct version, TRONCO implements CAPRESE and CAPRI algorithms for cancer progression inference together with functionalities to load input cross-sectional data, set up the execution of the algorithms, assess the statistical confidence in the results and visualize the inferred models.

Financial support. MA, GM, GC, AG, DR acknowledge Regione Lombardia (Italy) for the research projects RetroNet through the ASTIL Program [12-4-5148000-40]; U.A 053 and Network Enabled Drug Design project [ID14546A Rif SAL-7], Fondo Accordi Istituzionali 2009. BM acknowledges founding by the NSF grants CCF-0836649, CCF-0926166 and a NCI-PSOC grant.

References

[1] Özgün Babur, Mithat Gönen, Bülent Arman Aksoy, Nikolaus Schultz, Giovanni Ciriello, Chris Sander, and Emek Demir. Systematic identification of cancer driving signaling pathways based on mutual exclusivity of genomic alterations. *bioRxiv*, page 009878, 2014.

[2] Bradley Efron and B Efron. *The jackknife, the bootstrap and other resampling plans*, volume 38. SIAM, 1982.

[3] Matan Hofree, John P Shen, Hannah Carter, Andrew Gross, and Trey Ideker. Network-based stratification of tumor mutations. *Nature methods*, 10(11):1108–1115, 2013.

[4] Loes Olde Loohuis, Giulio Caravagna, Alex Graudenzi, Daniele Ramazzotti, Giancarlo Mauri, Marco Antoniotti, and Bud Mishra. Inferring tree causal models of cancer progression with probability raising. *PLoS One*, 9(10), 2014.

[5] NCI and the NHGRI. The cancer genome atlas. `http://cancergenome.nih.gov/`, 2005.

[6] Daniele Ramazzotti, Giulio Caravagna, Loes Olde-Loohuis, Alex Graudenzi, Ilya Korsunsky, Giancarlo Mauri, Marco Antoniotti, and Bud Mishra. Capri: Efficient inference of cancer progression models from cross-sectional data. *Bioinformatics*, page btv296, 2015.