Using the TRONCO package

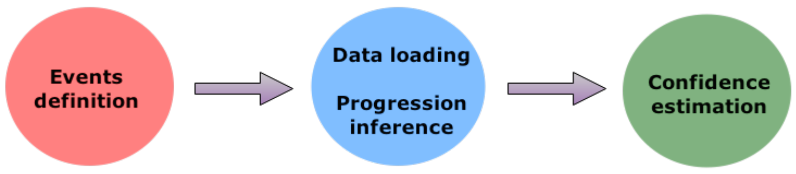
Marco Antoniotti[*] Giulio Caravagna[*] Alex Graudenzi[*] Ilya Korsunsky[†]
Mattia Longoni[*] Loes Olde Loohuis[‡] Giancarlo Mauri[*] Bud Mishra[†]
Daniele Ramazzotti[*]

October 21, 2014

Abstract. Genotype-level *cancer progression models* describe the ordering of accumulating mutations, e.g., somatic mutations / copy number variations, during cancer development. These graphical models help understand the "causal structure" involving events promoting cancer progression, possibly predicting complex patterns characterising genomic progression of a cancer. Reconstructed models can be used to better characterise genotype-phenotype relation, and suggest novel targets for therapy design.
TRONCO (TR*anslational* ONCO*logy*) is a R package aimed at collecting state-of-the-art algorithms to infer *progression models* from *cross-sectional* data, i.e., data collected from independent patients which does not necessarily incorporate any evident temporal information. These algorithms require a binary input matrix where: (*i*) each row represents a patient genome, (*ii*) each column an event relevant to the progression (a priori selected) and a 0/1 value models the absence/presence of a certain mutation in a certain patient.
The current first version of TRONCO implements the CAPRESE algorithm (CA*ncer* PR*ogression* E*xtraction with* S*ingle* E*dges*) to infer possible progression models arranged as *trees*; cfr.

- *Inferring tree causal models of cancer progression with probability raising*, L. Olde Loohuis, G. Caravagna, A. Graudenzi, D. Ramazzotti, G. Mauri, M. Antoniotti and B. Mishra. PLoS One, *to appear.*

This vignette shows how to use TRONCO to infer a tree model of ovarian cancer progression from CGH data of copy number alterations (classified as gains or losses over chromosome's arms). The dataset used is available in the SKY/M-FISH database. The reference manual for TRONCO is available in the package.



The TRONCO *workflow.*

Requirements:

You must have `rgraphviz` installed to use the package, see `Bioconductor.org`.

[*]Dipartimento di Informatica Sistemistica e Comunicazione, Universitá degli Studi Milano-Bicocca Milano, Italy.
[†]Courant Institute of Mathematical Sciences, New York University, New York, USA.
[‡]Center for Neurobehavioral Genetics, University of California, Los Angeles, USA.

1. Types/Events definition

First, load TRONCO in your R console.

```
> library(TRONCO)
```

Every node in the plotted topology can be colored according to the color table defined in R. You can use the command `colors` to see the available colors, e.g., `"red"`, `"blue"` or RGB `"#FF9900FF"`.

You can start defining the *event types* that you are considering, and assign them a color.

As an example, for CGH data we define two types of events, *gain* and *loss*, which we color *red* and *green* to represent amplifications or deletion of a chromosome arm. For instance, we can do this as follows:

```
> types.add("gain", "cornflowerblue")

Set color value "cornflowerblue" for events of type "gain"

> types.add("loss", "brown1")

Set color value "brown1" for events of type "loss"
```

If many types have to be defined it might be convenient to load all of them at once. This is possible by using a tabular input file (in `csv` format):

`type_name, type_color`   e.*g*., `red, gain`

and issuing the command `types.load("types.txt")` – if types are defined in file `types.txt`. The output produced by TRONCO might show warnings due to, e.g., different types assigned the same color.

Once types are defined, you can define the set of *events* in the dataset (which will constitute the progression), give them a *label*, a type and bind them to a dataset column. Since in general there are much more events than types, it might be convenient to prepare an external file to load via command `events.load("events.txt")`. The format expected for events is similar to the one expected for types, namely as a tabular input file in `csv` format:

`event_name, event_type, column_number`   e.*g*., `8p+, gain, 1`.

For the ovarian CGH dataset, such a file contains the following rows (we show the first 3 lines)

```
8p+, gain, 1
3p+, gain, 2
5q-, loss, 3
......
```

which define, as events, gains in arm p of chromosomes 8 and 3, losses on arm q of chromosomes 5, etc. Given the file *events.txt* where are defined the events with the above notation, the events can be loaded from a file as follows.

```
>  events.load("events.txt")

Added event "8q+" of type "gain" (color: "cornflowerblue"), dataset column "1"
Added event "3q+" of type "gain" (color: "cornflowerblue"), dataset column "2"
Added event "5q-" of type "loss" (color: "brown1"), dataset column "3"
Added event "4q-" of type "loss" (color: "brown1"), dataset column "4"
Added event "8p-" of type "loss" (color: "brown1"), dataset column "5"
Added event "1q+" of type "gain" (color: "cornflowerblue"), dataset column "6"
Added event "Xp-" of type "loss" (color: "brown1"), dataset column "7"
```

Events will constitute the nodes in the progression model. If one is willing to add events in a iterative fashion the command `events.add(event_name, event_type, column_number)` can be used. For instance `events.add("8q+", "gain", 1)`.

At this point, TRONCO executes some consistency checks to ensure that all the added events are of a declared type, and report the user potential inconsistencies.

2. Data loading & Progression inference

Once events are set, you can load the input dataset, which must be stored in a text file as a binary matrix (once loaded, you can use `tronco.data.view(your_data)` to visualise loaded data as a heatmap).

```
> data(ov.cgh)
> data.load(ov.cgh)
```

```
Data frame validated and data.values variable is loaded in global environment
```

```
> str(data.values)
```

```
'data.frame':        87 obs. of  7 variables:
 $ 8q+:gain: int  0 0 1 1 0 1 1 0 0 1 ...
 $ 3q+:gain: int  0 0 1 0 0 1 1 1 1 0 ...
 $ 5q-:loss: int  0 0 1 0 0 1 1 1 0 1 ...
 $ 4q-:loss: int  0 1 1 0 0 1 1 0 0 1 ...
 $ 8p-:loss: int  0 0 0 0 0 1 1 0 0 1 ...
 $ 1q+:gain: int  1 1 0 0 0 0 0 0 0 1 ...
 $ Xp-:loss: int  0 0 0 0 0 0 1 0 1 1 ...
```

In this case 87 samples are available and 7 events are considered (in general, the inference problem is well posed if there are more samples than events, which is the case here for ovarian).

Further consistency checks are performed by TRONCO at data-loading time; these include checking that:

- All the columns of the dataset are assigned a unique event;
- There are no identical columns in the dataset. If this is the case, the columns get merged and the events associated get merged too (a default type is assigned in this case);
- There are no columns in the dataset solely constituted by 0s or 1s. If this is the case, the columns and the events associated are deleted.

TRONCO signals the user that the data presents some inconsistency, if that is the case. Once the input is loaded, CAPRESE can be executed.

```
> topology <- tronco.caprese(data.values, lambda=0.5)
```

```
Executed CAPRESE algorithm with shrinkage coefficient: 0.5
 Estimated false positives error rate:  0.1949365
 Estimated false negative error rate:  0.1168414
```

In the above example, CAPRESE is executed with a *shrinkage coefficient* set to 0.5 (the default value, if not specified), which is the optimal value for data containing *false positives* and *false negatives.* If these were absent, the optimal coefficient should be set to an arbitrary small value, e.g. 10^{-3}; in any case the coefficient must be in $[0, 1]$. Notice that TRONCO provides an *empirical estimation* of the the rate of false positives and negatives in the data, given the reconstructed model; this is done via ℓ_2 distance.

The returned topology can be printed to screen by using the `topology` object print method, or can be visualized by using the `tronco.plot` function.

```
> topology
```

```
 Tree progression model with  7 events
 8q+:gain  ->  3q+:gain
 8q+:gain  ->  8p-:loss
 5q-:loss  ->  4q-:loss
 8p-:loss  ->  Xp-:loss

 Estimated false positives error rate: 0.1949365
 Estimated false negative error rate: 0.1168414
```

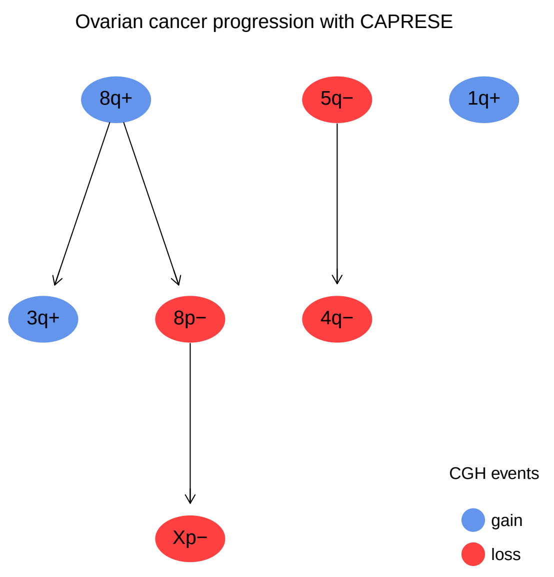


Figure 1: **Ovarian cancer CGH tree reconstructed with CAPRESE.** We show the result of reconstruction with CAPRESE. These trees are plot as explained in §2 and 3. The tree is the reconstructed model without confidence information.

```
> tronco.plot(topology, title="Ovarian cancer progression with CAPRESE", legend.title="CGH events",
+         legend.coeff = 1.0, label.coeff = 1.2, legend = TRUE)
```

```
Plot created successfully
```

In this case we are assigning a title to the plot, we are requiring to display a legend (`legend = TRUE`), and we are setting custom size for the text in the legend (`legend.coeff = 0.7`, 70% of the default size) and in the model (`label.coeff = 1.2`); see Figure 1.

3. Confidence estimation

Data and model probabilities. Before estimating the confidence of a reconstruction, one might print and visualise the *frequency of occurrence* for each event, the *joint distribution* and the *conditional distribution* according to the input data (i.e., the *observed* probabilities). Notice that for the conditional distribution we condition only on the parent of a node, as reconstructed in the returned model. Plots of these distributions are shown in Figure 2, and are evaluated as follows.

```
>  confidence.data.single(topology)
```

```
 8q+:gain  3q+:gain  5q-:loss  4q-:loss  8p-:loss  1q+:gain  Xp-:loss 
0.7011494 0.5517241 0.5287356 0.5057471 0.4712644 0.4367816 0.4252874 
```

```
>  confidence.data.joint(topology)
```

```
          8q+:gain  3q+:gain  5q-:loss  4q-:loss  8p-:loss  1q+:gain  Xp-:loss
8q+:gain 0.7011494 0.4827586 0.4022989 0.4022989 0.4252874 0.2988506 0.3218391
3q+:gain 0.4827586 0.5517241 0.3333333 0.2988506 0.2988506 0.2643678 0.2758621
5q-:loss 0.4022989 0.3333333 0.5287356 0.3908046 0.3678161 0.2298851 0.2988506
4q-:loss 0.4022989 0.2988506 0.3908046 0.5057471 0.3448276 0.2413793 0.2873563
8p-:loss 0.4252874 0.2988506 0.3678161 0.3448276 0.4712644 0.1954023 0.3103448
1q+:gain 0.2988506 0.2643678 0.2298851 0.2413793 0.1954023 0.4367816 0.1954023
Xp-:loss 0.3218391 0.2758621 0.2988506 0.2873563 0.3103448 0.1954023 0.4252874
```

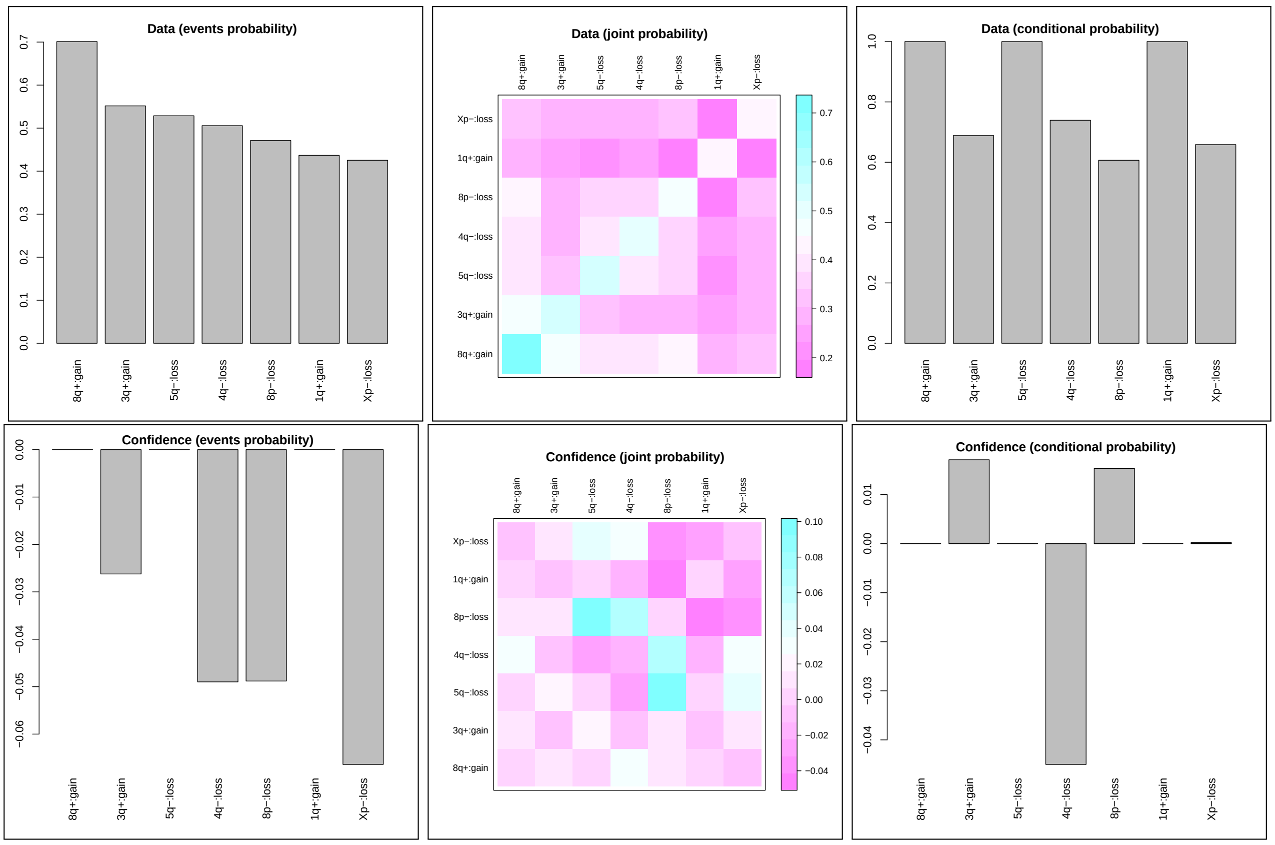


Figure 2: **Probabilities (input data): visualisation and comparison with model's predictions.** Top: observed *frequencies* of *observed*, *joint* and *conditional* distributions of events (conditionals are restricted according to the reconstructed progression model) as emerge from the data. Bottom: difference between observed and fitted probabilities, according to the reconstructed progression.

```
>  confidence.data.conditional(topology)

 8q+:gain  3q+:gain  5q-:loss  4q-:loss  8p-:loss  1q+:gain  Xp-:loss
1.0000000 0.6885246 1.0000000 0.7391304 0.6065574 1.0000000 0.6585366
```

In a similar way, by using `confidence.fit.single(topology)`, `confidence.fit.joint(topology)` or `confidence.fit.conditional(topology)`, the analogous probabilities can be assessed according to the model. This are not shown in this vignette.

The difference between observed and fit probabilities can be visualised as follows.

```
> confidence.single(topology)

   8q+:gain    3q+:gain    5q-:loss    4q-:loss    8p-:loss    1q+:gain
 0.00000000 -0.02622809  0.00000000 -0.04898781 -0.04880958  0.00000000
   Xp-:loss
-0.06640509

> confidence.joint(topology)

             8q+:gain     3q+:gain    5q-:loss    4q-:loss     8p-:loss
8q+:gain  0.000000000  0.011991503  0.00000000  0.02808834  0.010747173
3q+:gain  0.011991503 -0.004805063  0.02040118 -0.01135714  0.011736173
5q-:loss  0.000000000  0.020401180  0.00000000 -0.02379790  0.092338948
4q-:loss  0.028088341 -0.011357135 -0.02379790 -0.01704690  0.071838213
8p-:loss  0.010747173  0.011736173  0.09233895  0.07183821 -0.001166571
```

```
1q+:gain  0.000000000 -0.006400071  0.00000000 -0.01583318 -0.041573393
Xp-:loss -0.004093925  0.007657191  0.04174007  0.03261753 -0.032038023
             1q+:gain     Xp-:loss
8q+:gain  0.000000000 -0.004093925
3q+:gain -0.006400071  0.007657191
5q-:loss  0.000000000  0.041740070
4q-:loss -0.015833176  0.032617531
8p-:loss -0.041573393 -0.032038023
1q+:gain  0.000000000 -0.025002872
Xp-:loss -0.025002872 -0.005904709
```

```
> confidence.conditional(topology)
```

```
     8q+:gain      3q+:gain      5q-:loss      4q-:loss      8p-:loss
 0.0000000000  0.0171026351  0.0000000000 -0.0450090696  0.0153279358
     1q+:gain      Xp-:loss
 0.0000000000  0.0002016449
```

Bootstrap confidence.

Confidence in a model can be estimated via *parametric* and *non-parametric bootstrap*. In the former case, the model is assumed to be correct and data is sampled by the model, in the latter case resamples are taken from the input data, with repetitions. In any case, the reconstruction confidence is the number of times that the estimated tree or edge is inferred out of a number of resamples. The parameters of the bootstrap procedure can be custom set.

```
> set.seed(12345)
> topology <- tronco.bootstrap(topology, type="non-parametric", nboot=1000)
```

```
Executing bootstrap algorithm this may take several time...
Executed non-parametric bootsrap with 1000 as sampling number and 0.5 as lambda value

         8q+:gain 3q+:gain 5q-:loss 4q-:loss 8p-:loss 1q+:gain Xp-:loss
8q+:gain        0    0.931        0    0.000    0.489        0    0.000
3q+:gain        0    0.000        0    0.000    0.000        0    0.000
5q-:loss        0    0.000        0    0.529    0.000        0    0.000
4q-:loss        0    0.000        0    0.000    0.000        0    0.000
8p-:loss        0    0.000        0    0.000    0.000        0    0.625
1q+:gain        0    0.000        0    0.000    0.000        0    0.000
Xp-:loss        0    0.000        0    0.000    0.000        0    0.000

Confidence overall value: 72
Confidence overall frequency: 0.072
```

```
> tronco.bootstrap.show(topology)
```

```
         8q+:gain 3q+:gain 5q-:loss 4q-:loss 8p-:loss 1q+:gain Xp-:loss
8q+:gain        0    0.931        0    0.000    0.489        0    0.000
3q+:gain        0    0.000        0    0.000    0.000        0    0.000
5q-:loss        0    0.000        0    0.529    0.000        0    0.000
4q-:loss        0    0.000        0    0.000    0.000        0    0.000
8p-:loss        0    0.000        0    0.000    0.000        0    0.625
1q+:gain        0    0.000        0    0.000    0.000        0    0.000
Xp-:loss        0    0.000        0    0.000    0.000        0    0.000
```

In this case, for instance, we are performing non-parametric bootstrap (the default one) with 1000 repetitions and, since no shrinkage coefficient is specified, we are still using 0.5. Here the estimated error rates are used to include noise levels estimated from the data/model. To perform parametric bootstrap is enough to use the flag `type="parametric"`.

```
> set.seed(12345)
> topology <- tronco.bootstrap(topology, type="parametric", nboot=1000)

Executing bootstrap algorithm this may take several time...
Executed parametric bootsrap with 1000 as sampling number and 0.5 as lambda value

         8q+:gain 3q+:gain 5q-:loss 4q-:loss 8p-:loss 1q+:gain Xp-:loss
8q+:gain        0    0.632        0     0.00    0.544        0    0.000
3q+:gain        0    0.000        0     0.00    0.000        0    0.000
5q-:loss        0    0.000        0     0.76    0.000        0    0.000
4q-:loss        0    0.000        0     0.00    0.000        0    0.000
8p-:loss        0    0.000        0     0.00    0.000        0    0.661
1q+:gain        0    0.000        0     0.00    0.000        0    0.000
Xp-:loss        0    0.000        0     0.00    0.000        0    0.000

Confidence overall value: 186
Confidence overall frequency: 0.186

> tronco.bootstrap.show(topology)

         8q+:gain 3q+:gain 5q-:loss 4q-:loss 8p-:loss 1q+:gain Xp-:loss
8q+:gain        0    0.632        0     0.00    0.544        0    0.000
3q+:gain        0    0.000        0     0.00    0.000        0    0.000
5q-:loss        0    0.000        0     0.76    0.000        0    0.000
4q-:loss        0    0.000        0     0.00    0.000        0    0.000
8p-:loss        0    0.000        0     0.00    0.000        0    0.661
1q+:gain        0    0.000        0     0.00    0.000        0    0.000
Xp-:loss        0    0.000        0     0.00    0.000        0    0.000
```

Results of bootstrapping are visualized as a table (useful for edge confidence), and as a heatmap by using command `tronco.bootstrap.show`. The overall model confidence is reported, too. In Figure 3 results of bootstrap are shown. If one is willing to visualize this confidence in the plot of the inferred tree an input flag `confidence` can be used with function `tronco.plot`. For instance:

```
> tronco.plot(topology, title="Ovarian cancer progression with CAPRESE", legend.title="CGH events",
+          legend.coeff = 1.0, label.coeff = 1.2, legend = TRUE, confidence = TRUE)

Plot created successfully
```

In this case, the thicker lines reflect the most confident edges; confidence is also reported as labels of edges, as shown in Figure 4

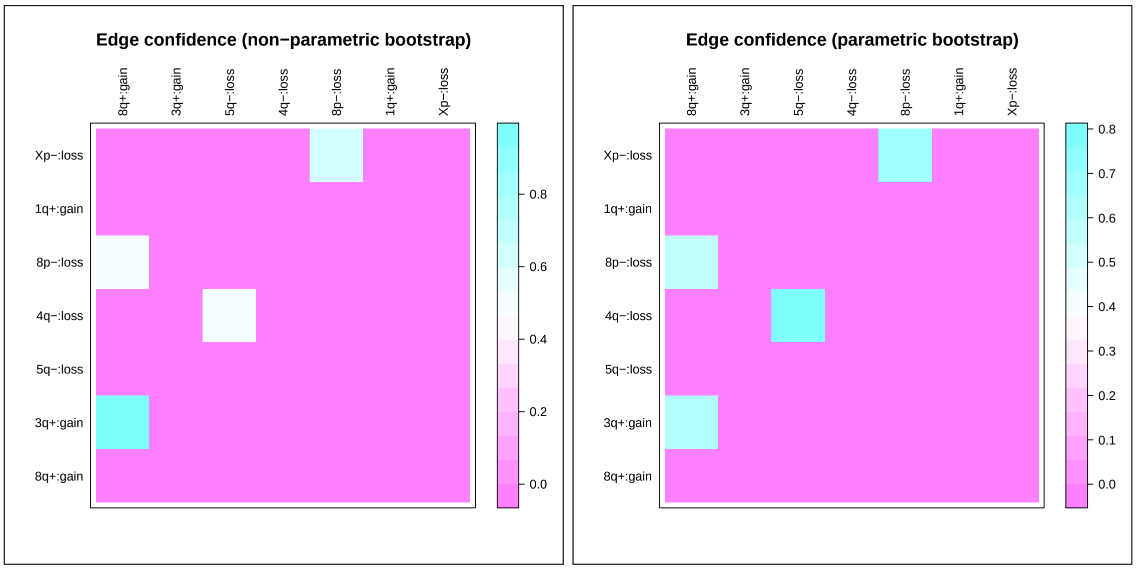


Figure 3: **Bootstrap for edge confidence.** Non-parametric and parametric confidence in each reconstructed edge as assessed via bootstrapping.

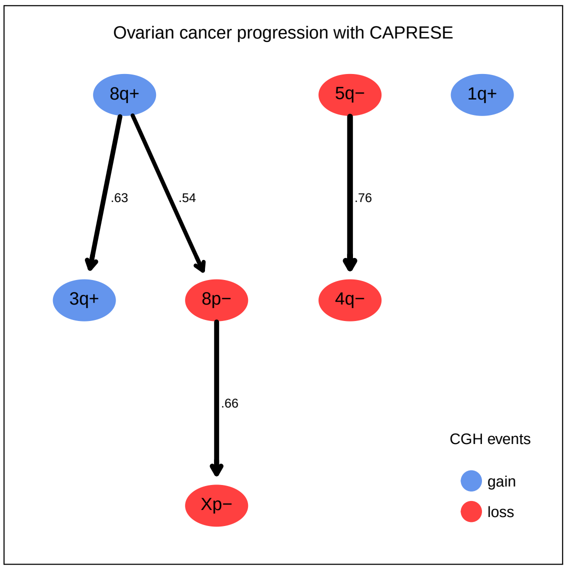


Figure 4: **Bootstrap information included in the model.** You can include the result of edge confidence estimation via bootstrap by using flag `confidence`. In this case the thickness of each edge is proportional to its estimated confidence.