Demonstration of HyperTraPS in R

HyperTraPS (hypercubic transition path sampling) is a family of algorithms for inferring the dynamics of “accumulation” processes. These are processes where a set of binary features are acquired over time.

HyperTraPS will take a set of observed states, described by binary strings recording the presence/absence of each feature. It may optionally take initial states from which these states are reached, and information on the timings associated with each observation. It returns various summaries of which feature are acquired and when, and how these features influence each other.

Loading the software

If we just want the HyperTraPS code without other dependencies, we only need Rcpp, and can use the following

```
library(Rcpp)
sourceCpp("hypertraps-r.cpp")
```

If we want various helper functions and ggplot functions in addition, use this

```
source("hypertraps.R")
```

Simple demo

Here we’ll construct a simple synthetic dataset. The `m.1` matrix will store a set of initial states, and the `m.2` matrix will store a set of later observed states. The first row of `m.1` , for example, stores the state 00000, where no features have been acquired. The first row of `m.2` stores 10000, where the first of five features has been acquired.

The times correspond to each observation, so each transition has an associated time of 0.1 (in whatever units we are working with).

```
m.1 = matrix(rep(c(0,0,0,0,0,
                1,0,0,0,0,
                1,1,0,0,0,
                1,1,1,0,0,
                1,1,1,1,0,
                0,0,0,0,0,
                0,0,0,0,1,
                0,0,0,1,1,
                0,0,1,1,1,
                0,1,1,1,1),5), byrow=TRUE, ncol=5)
m.2 = matrix(rep(c(1,0,0,0,0,
                1,1,0,0,0,
                1,1,1,0,0,
                1,1,1,1,0,
                1,1,1,1,1,
                0,0,0,0,1,
                0,0,0,1,1,
                0,0,1,1,1,
                0,1,1,1,1,
                1,1,1,1,1),5), byrow=TRUE, ncol=5)
times = rep(0.1, 50)
```

Let’s run HyperTraPS with these “before” and “after” observations. By using `times` as both the start and end time arguments, we say that each transition takes precisely that associated time – we could allow broader time windows to capture uncertainty in timing. Finally, we provide labels for the five individual features involved.

```
my.post = HyperTraPS(m.2, initialstates = m.1,
                     starttimes = times, endtimes = times,
                     featurenames = c("A", "B", "C", "D", "E"))
```

```
##
## HyperTraPS(-CT)
## Sep 2023
##
## Unpublished code -- please do not circulate!
## Published version available at:
##     https://github.com/StochasticBiology/HyperTraPS
## with stripped-down version at:
##     https://github.com/StochasticBiology/hypertraps-simple
##
## Running HyperTraPS-CT with:
## [observations-file]: rcpp
## [start-timings-file]:
## [end-timings-file]:
## [random number seed]: 1
## [length index]: 3
## [kernel index]: 5
## [walkers]: 200
## [losses (1) or gains (0)]: 0
## [APM]: 0
## [model]: 2
## [penalty]: 0.000e+00
## [lasso]: 0
##
## Using MH MCMC
## Number of features is 5, I found 50 observation pairs and 50 timing pairs
##
## Starting with simple initial param guess
## One likelihood estimation took 1.093000e-03 seconds.
## Initial likelihood is -9.895393e+01
## Second guess is -9.895393e+01
## This code (1000 steps) will probably take around 1.093 seconds (0.000 hours) to complete.
##
## 0 - Iteration 0 likelihood -98.953934 total-acceptance 0.000000 recent-acceptance 0.000000 trial-likelihood -1
03.296770 penalty 0.000000,0.000000
## 100 - Iteration 100 likelihood -73.336344 total-acceptance 0.227723 recent-acceptance 0.230000 trial-likelihoo
d -80.977429 penalty 0.000000,0.000000
## 200 - Iteration 200 likelihood -65.896941 total-acceptance 0.243781 recent-acceptance 0.260000 trial-likelihoo
d -70.983106 penalty 0.000000,0.000000
## 300 - Iteration 300 likelihood -64.723588 total-acceptance 0.229236 recent-acceptance 0.200000 trial-likelihoo
d -70.464060 penalty 0.000000,0.000000
## 400 - Iteration 400 likelihood -64.696648 total-acceptance 0.236908 recent-acceptance 0.260000 trial-likelihoo
d -68.705614 penalty 0.000000,0.000000
## 500 - Iteration 500 likelihood -62.483817 total-acceptance 0.235529 recent-acceptance 0.230000 trial-likelihoo
d -63.075140 penalty 0.000000,0.000000
## 600 - Iteration 600 likelihood -59.241903 total-acceptance 0.236273 recent-acceptance 0.240000 trial-likelihoo
d -61.894393 penalty 0.000000,0.000000
## 700 - Iteration 700 likelihood -56.371564 total-acceptance 0.232525 recent-acceptance 0.210000 trial-likelihoo
d -60.974947 penalty 0.000000,0.000000
## 800 - Iteration 800 likelihood -55.392606 total-acceptance 0.220974 recent-acceptance 0.140000 trial-likelihoo
d -58.086008 penalty 0.000000,0.000000
## 900 - Iteration 900 likelihood -57.435868 total-acceptance 0.229745 recent-acceptance 0.300000 trial-likelihoo
d -57.435868 penalty 0.000000,0.000000
##
## HyperTraPS(-CT) posterior analysis
##
## Verbose flag is 0
## Bin scale is 10.000000
## Taking 10 samples per parameterisation
## Using posterior samples with 7 x 25 entries
## Based on rcpp with 25 params per model and model 2, there are 5 features
## Output label is rcpp
## allruns is 70
## 0 1.708571
## 1 2.113000
## 2 2.041286
## 3 2.179000
## 4 1.958143
```

That output takes us through the HyperTraPS process. First, the arguments provided to the function call are described, then the algorithm chosen (MH MCMC) and a summary of the input data is given. To estimate runtime, HyperTraPS reports the time taken for a single likelihood calculation, then scales this by the estimated number of calculations required to give a runtime estimate. Here this is only 1.17 seconds, but for chains long enough for satisfactory convergence (and more complicated systems) this may be dramatically longer.

Because we didn't turn off output, HyperTraPS periodically outputs information about the run as it goes. Every 100th step, it outputs: the

likelihood of the current parameterisation; the step acceptance rate through the whole run; the step acceptance rate since the last output; the likelihood of the most recent proposal. These can help design efficient MCMC approaches: if the acceptance rate is too low and/or the recent proposal likelihood is much lower than the current, consider a smaller perturbation kernel. It also helps us see when the chain is "burned in" – when the likelihood fluctuates, rather than consistently increasing, we're probably more stable.

After the MCMC run, the posterior analysis begins, and outputs a few details. These include the size of the posterior sample set being explored, the model being used, and a quick summary of the mean acquisition orderings of each feature. These are really just checks for debugging; not much useful information can be seen from them.

The default chain length (1000 steps) is good for a short demo, but we'll get more robust results if we run for longer. Let's do 10000 steps, suppressing output to avoid pages of updates. "length" specifies log10(number of steps).

```
my.post = HyperTraPS(m.2, initialstates = m.1,
                     starttimes = times, endtimes = times,
                     featurenames = c("A", "B", "C", "D", "E"),
                     length = 4, limited_output = 1)
```

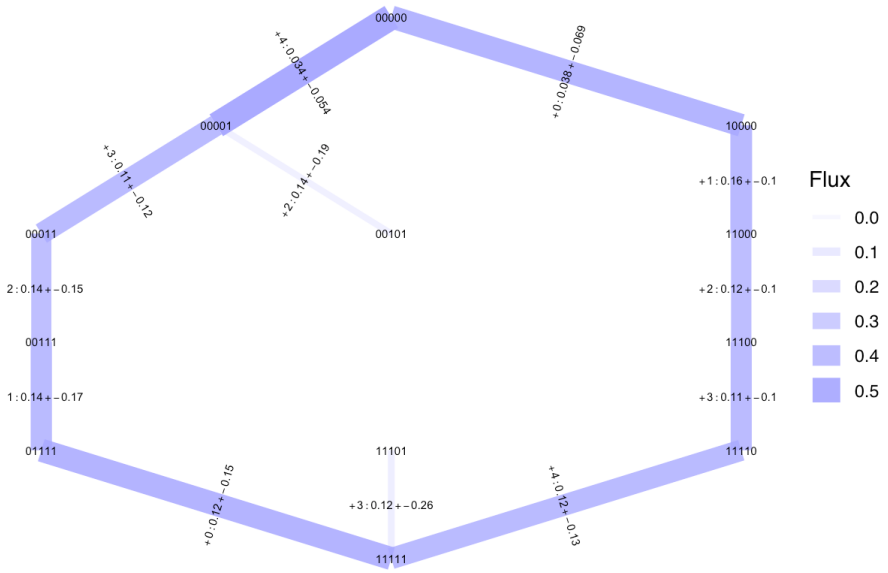
```
## One likelihood estimation took 1.087000e-03 seconds.
## Initial likelihood is -9.895393e+01
## Second guess is -9.895393e+01
## This code (10000 steps) will probably take around 10.870 seconds (0.003 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 - 1000 - 1100 - 1200 - 1300 - 1400 - 1500 - 1600 - 170
0 - 1800 - 1900 - 2000 - 2100 - 2200 - 2300 - 2400 - 2500 - 2600 - 2700 - 2800 - 2900 - 3000 - 3100 - 3200 - 3300
- 3400 - 3500 - 3600 - 3700 - 3800 - 3900 - 4000 - 4100 - 4200 - 4300 - 4400 - 4500 - 4600 - 4700 - 4800 - 4900 -
5000 - 5100 - 5200 - 5300 - 5400 - 5500 - 5600 - 5700 - 5800 - 5900 - 6000 - 6100 - 6200 - 6300 - 6400 - 6500 - 6
600 - 6700 - 6800 - 6900 - 7000 - 7100 - 7200 - 7300 - 7400 - 7500 - 7600 - 7700 - 7800 - 7900 - 8000 - 8100 - 82
00 - 8300 - 8400 - 8500 - 8600 - 8700 - 8800 - 8900 - 9000 - 9100 - 9200 - 9300 - 9400 - 9500 - 9600 - 9700 - 980
0 - 9900 -
```

Now – visualising the results.

Transition graph plot. This plot shows a set of sampled routes on the hypercubic space. Each edge is a transition, corresponding to the acquisition of a new feature. The edges are labelled by the feature acquired and the statistics of the time taken (mean +- s.d.). The edge width gives the probability flux through that transition.

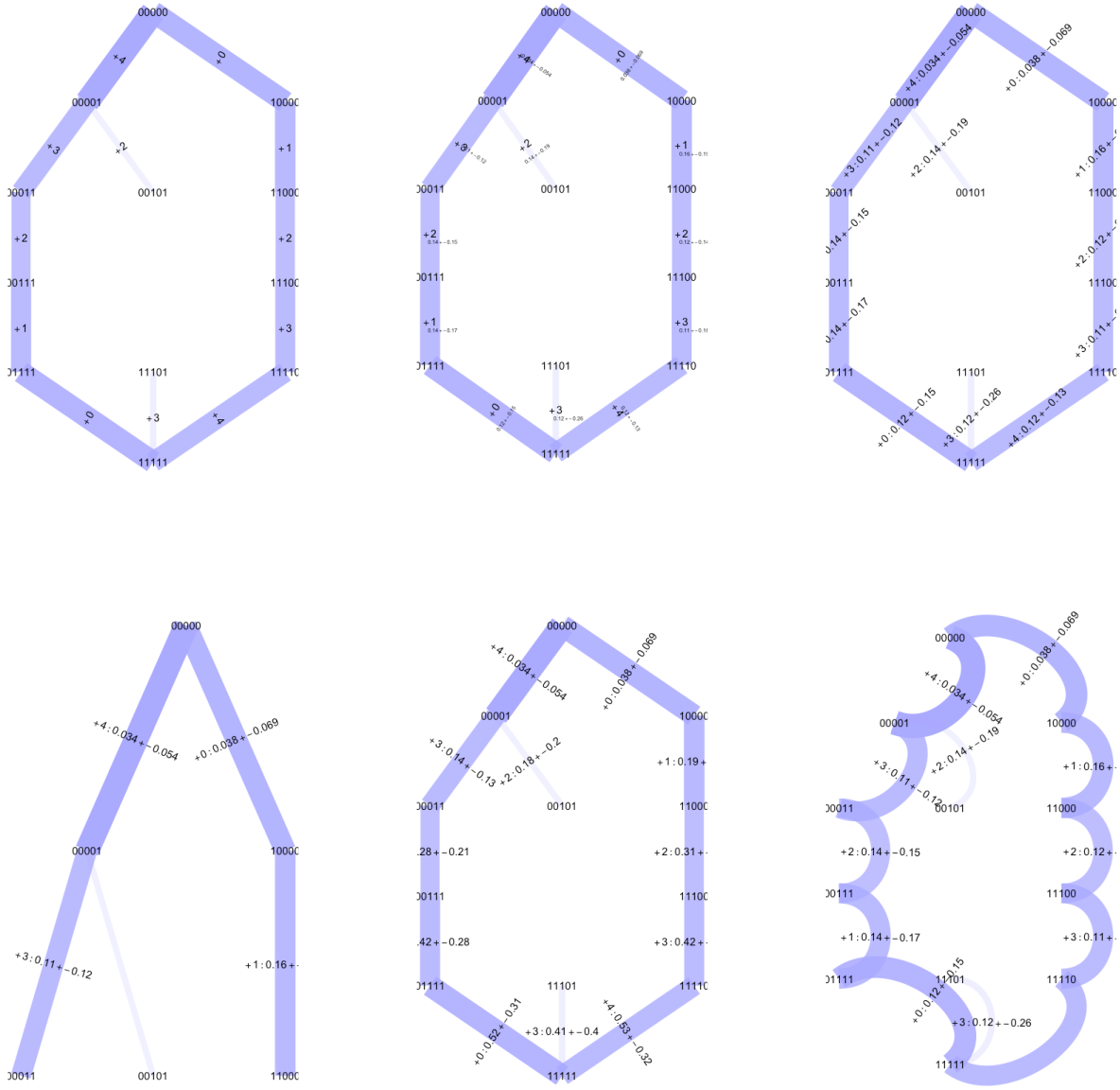
```
plotHypercube.sampledgraph2(my.post)
```

```
## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` in the `default_aes` field and elsewhere instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



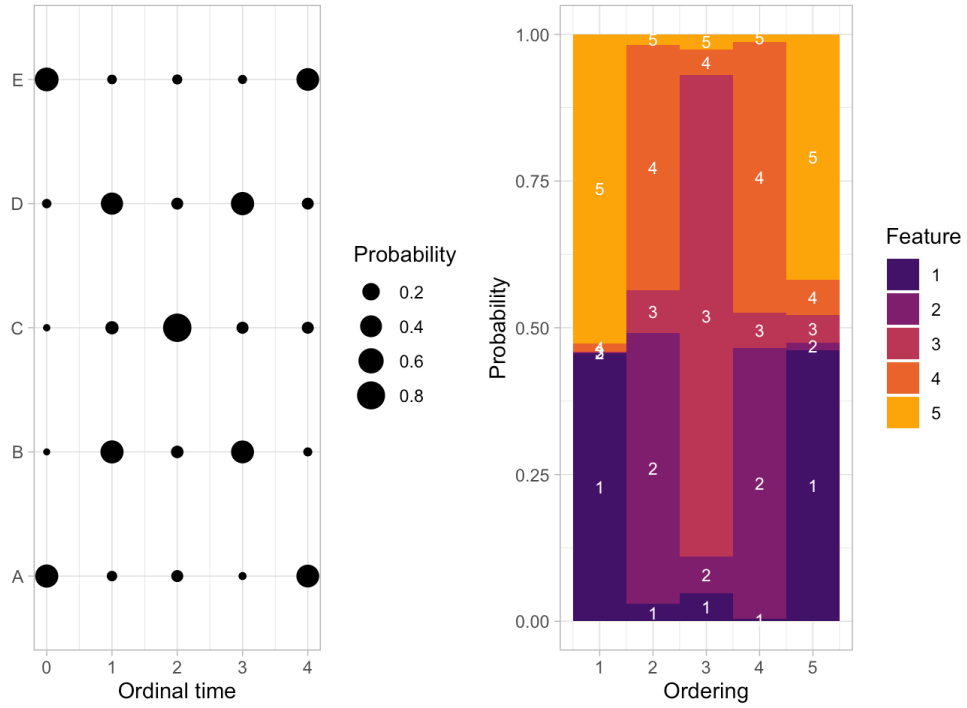
There are a lot of options for this plot type. Here are some of them. Respectively: (i) no time labels; (ii) small time labels; (iii) different label angles; (iv) cube truncated two steps after the root; (v) total timings from start, not timing of each transition; (vi) different edge style.

```
ggarrange(plotHypercube.sampledgraph2(my.post, no.times=TRUE) + theme(legend.position = "none"),
          plotHypercube.sampledgraph2(my.post, no.times=TRUE, small.times=TRUE) + theme(legend.position = "non
e"),
          plotHypercube.sampledgraph2(my.post, edge.label.angle="none") + theme(legend.position = "none"),
          plotHypercube.sampledgraph2(my.post, truncate=2) + theme(legend.position = "none"),
          plotHypercube.sampledgraph2(my.post, use.timediffs=FALSE) + theme(legend.position = "none"),
          plotHypercube.sampledgraph2(my.post, use.arc=TRUE) + theme(legend.position = "none")
          )
```



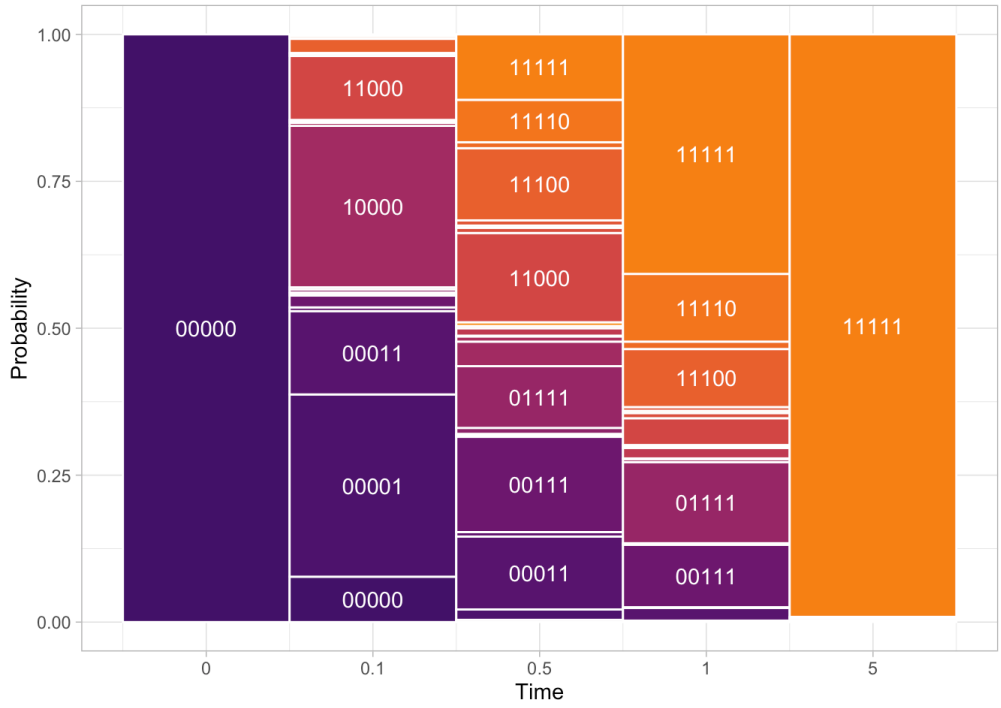
Bubble/motif plot. These plots summarise the inferred dynamics of features, forgetting specific states. The size of a circle at point x,y, or the height of bar y at point x, gives the probability that feature y is acquired at ordinal time x. You can choose which is more beautiful/informative.

```
ggarrange(plotHypercube.bubbles(my.post),
          plotHypercube.motifs(my.post))
```



Motif time series plot. Given a set of sampling times, this plot gives the set of likely states of the system at those times. As with the motifs above, the height of a bar gives its probability at that time.

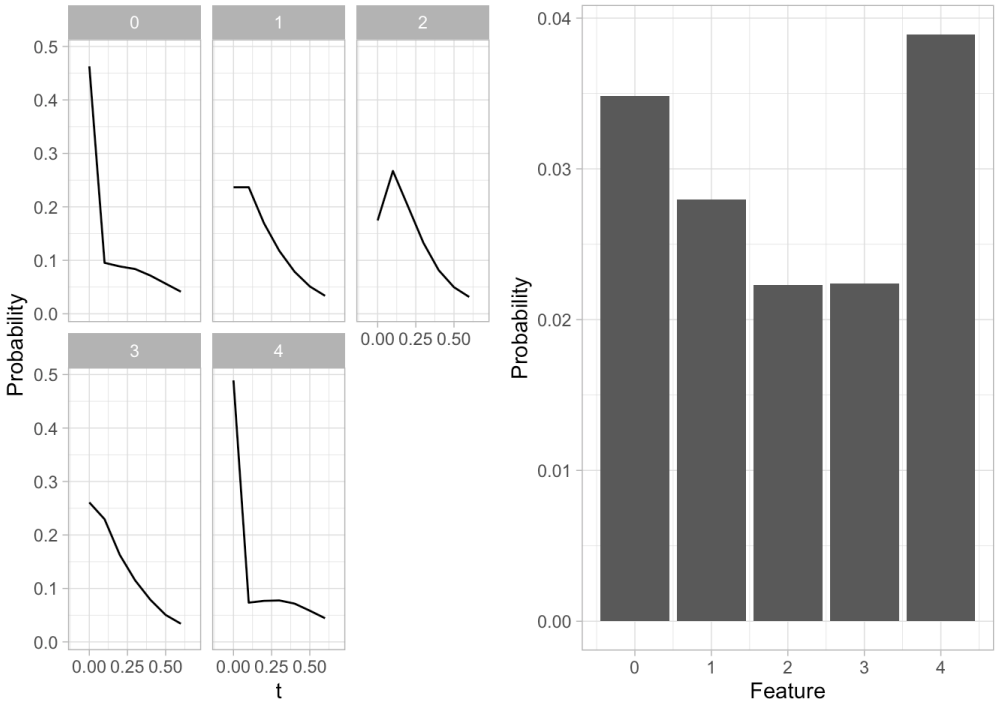
```
ggarrange(plotHypercube.motifseries(my.post, t.set=c(0,0.1,0.5,1,5)))
```



Timing histograms. Histograms of acquisition time for each feature, and probability that each feature is not acquired within a given time threshold (here, 1). By default this would be plotted on a log timescale; we'll linearise for this simple system.

```
plotHypercube.timehists(my.post, t.thresh=1, log.time=FALSE)
```

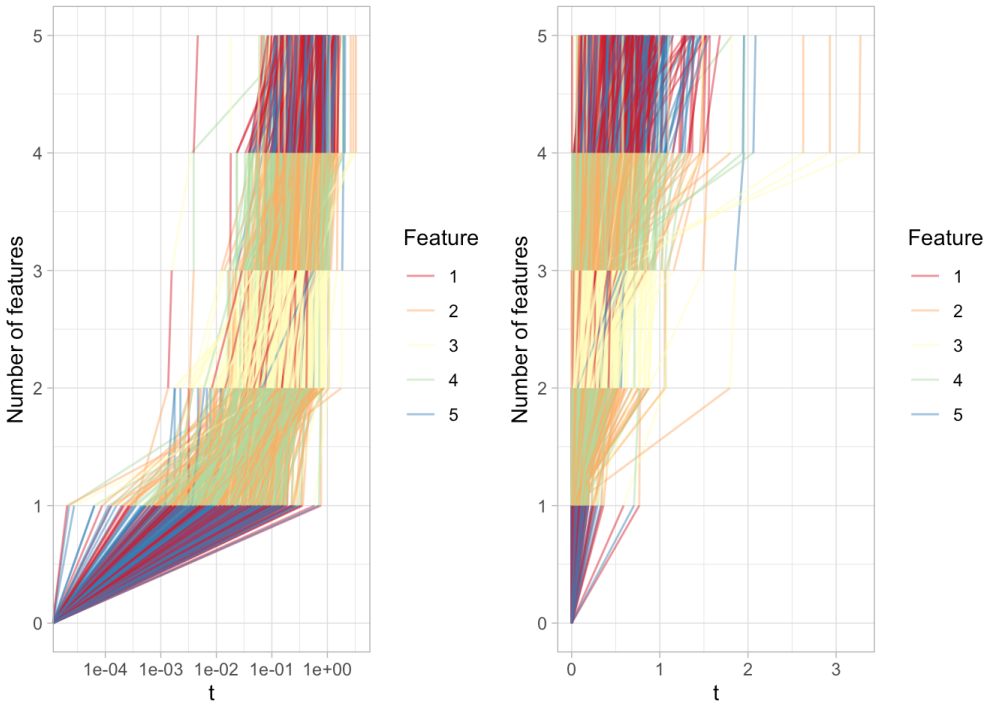
```
## Warning: Removed 6 rows containing missing values (`geom_line()`).
```



Time series plot. Each step up the vertical axis corresponds to the acquisition of a new feature. The horizontal axis gives the time of the corresponding event, and the colour labels the feature that was acquired. Plotted with and without logged time axis.

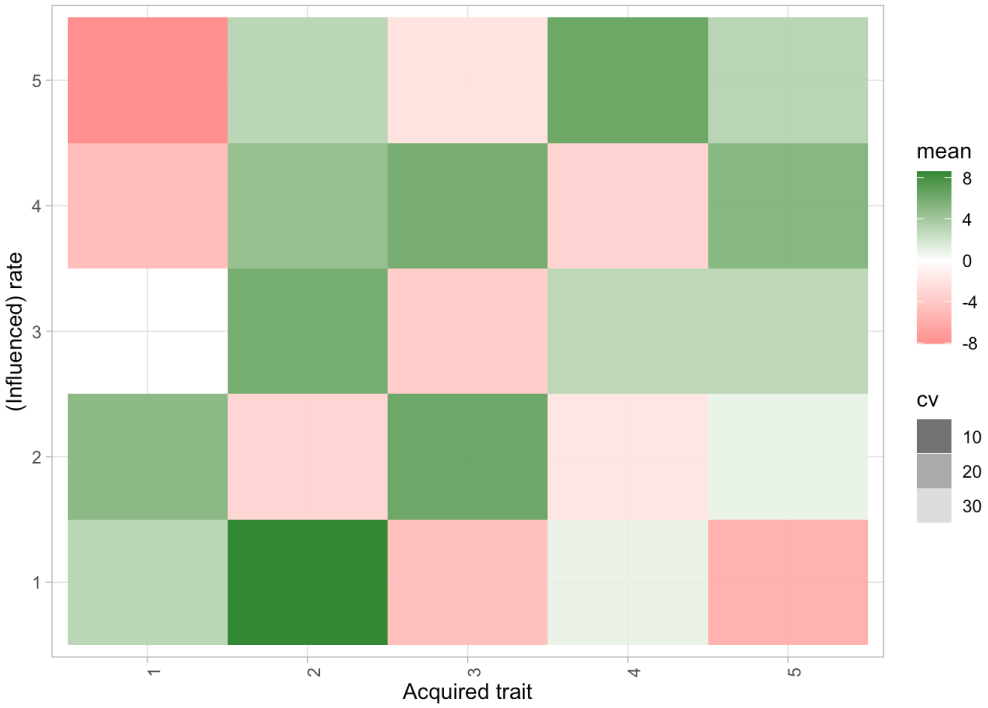
```
ggarrange(plotHypercube.timeseries(my.post),
          plotHypercube.timeseries(my.post, log.time=FALSE))
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```



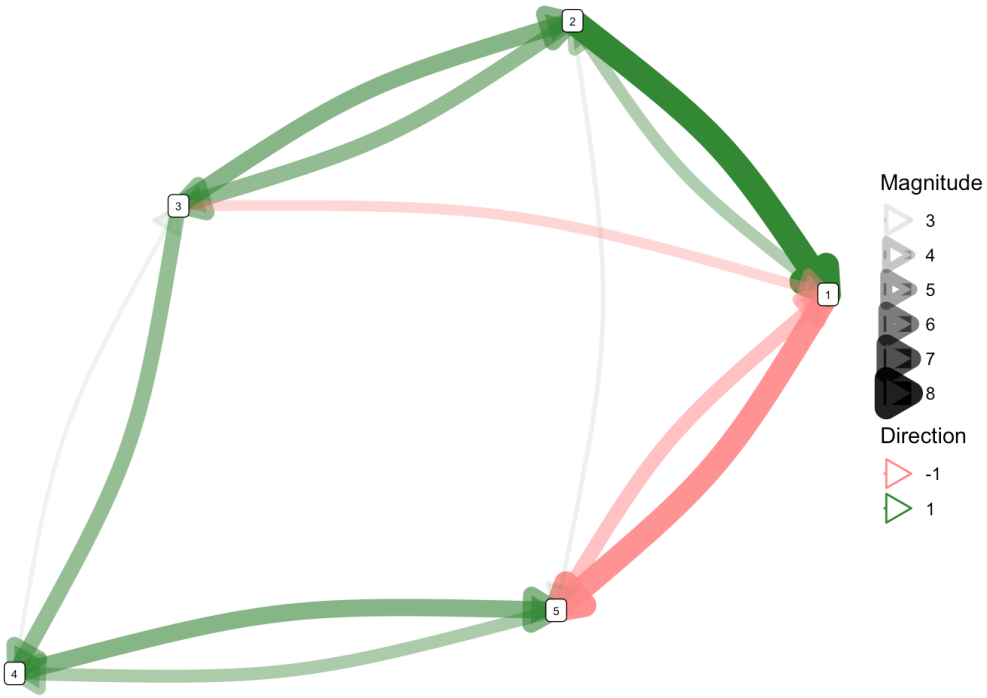
Influences between features. How each acquired trait (horizontal axis) influences the propensity for each other trait (vertical axis) to be acquired. Red is repression, blue is promotion; the transparency gives the width of the associated posterior. Here we see that 1 and 5 strongly cross-repress (being the first steps on the two competing pathways) and each feature promotes the next step on its pathway (wrapping the diagonal).

```
plotHypercube.influences(my.post)
```



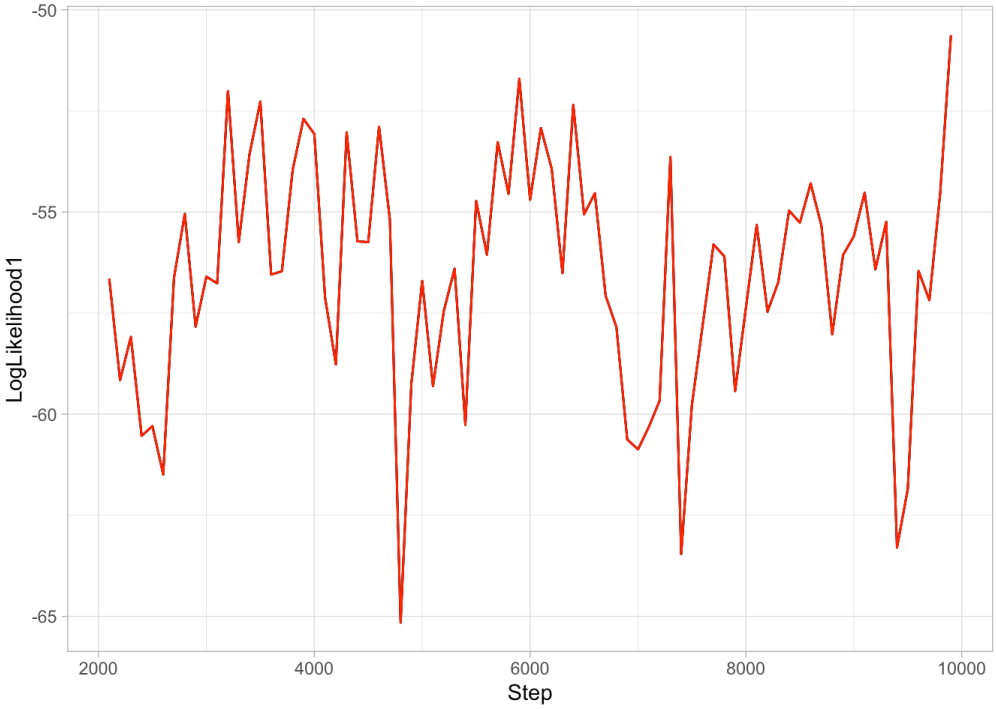
This can be perhaps more intuitively, and certainly more generally, be represented as a graph of influences: here we also impose a threshold on the coefficient of variation of the associated posterior, to report only the more “reliable” influences.

```
plotHypercube.influencegraph(my.post, cv.thresh = 0.7)
```



Likelihood trace. Last in this list, but perhaps the first to view, the trace of likelihood estimates through the course of the MCMC search (or optimisation process; see below) offers a first diagnostic about the performance of the inference process. There are two black lines and one red. The red line shows the likelihood estimate that is actually used in the MCMC process. The two black lines show periodic, independent recalculations of the estimate. In the perfect case they will totally overlap and resemble uncorrelated white noise with no systematic trend.

```
plotHypercube.lik.trace(my.post)
```



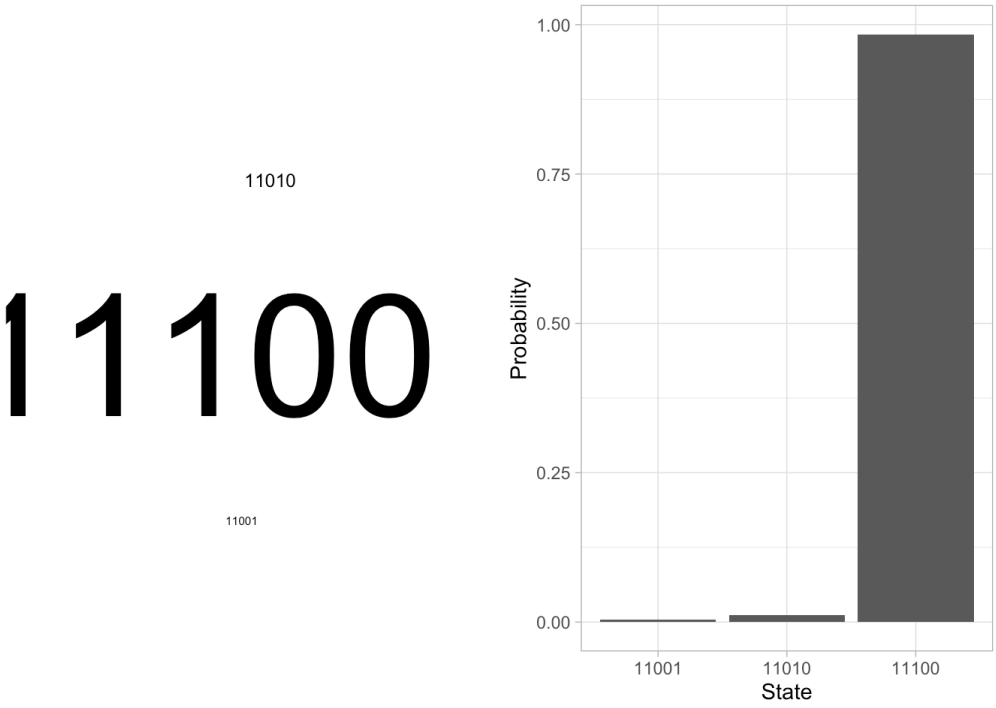
Some things that can go wrong:

- The two black lines look very different. The path sampling process is not given consistent estimates for the likelihood. Use more walkers.
- Red line is systematically above black line. A rare extreme estimate of the likelihood has been fixed and has frozen the simulation. Use more walkers as above for more precise estimation; consider using auxiliary pseudo-marginal MCMC.
- Systematic trend in likelihood: gradual increase, changing the baseline level. Chain has not equilibrated and is taking a long time to locate good parameter sets; consider running longer chains, or increasing step size (but see below).
- Systematic trend in likelihood: punctuated increases. Chain has not equilibrated and is getting stuck in local optima; consider running longer chains, or decreasing step size (but see above).

Predictions of unseen and future behaviour

We can use the inferred transition network to make predictions about what will happen next in a given state. Let's query the inferred hypercube to see what will happen next when we are part of the way down one pathway.

```
prediction.step = predictNextStep(my.post, c(1,1,0,0,0))
plotHypercube.prediction(prediction.step)
```



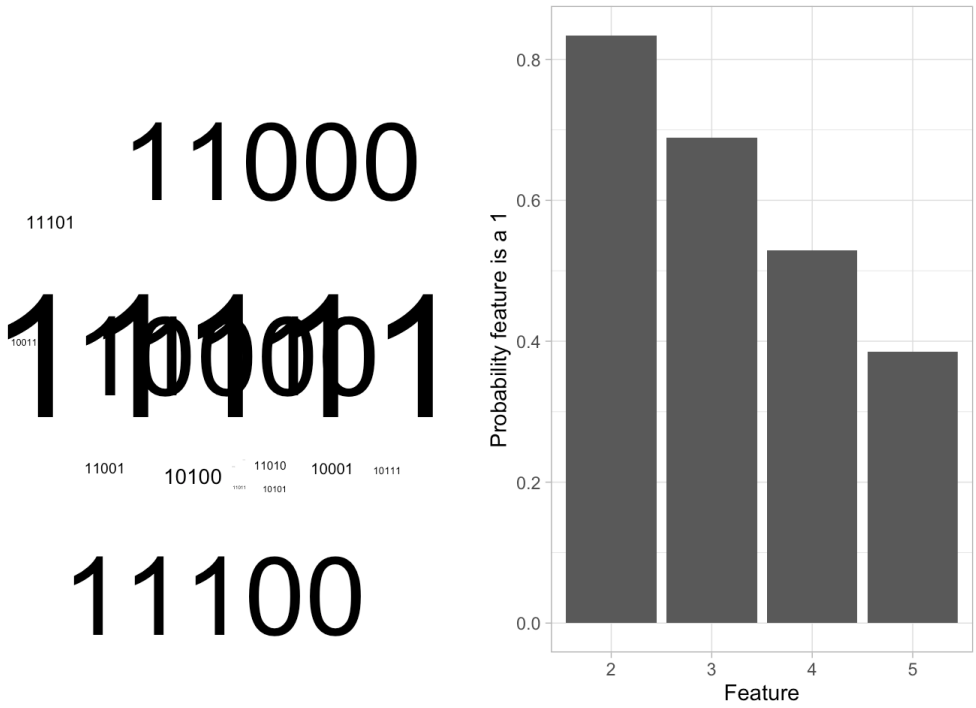
The graphic reports the likely next steps, as a word cloud and a set of probabilities. Here, we're making the strong prediction that the next step after 11000 is 11100.

We can also make predictions about hidden values in new observations. First let's mask an observation and ask HyperTraPS to "fill in the blanks"

of 1????.

```
prediction.hidden = predictHiddenVals(my.post, c(1,2,2,2,2))
plotHypercube.prediction(prediction.hidden)
```

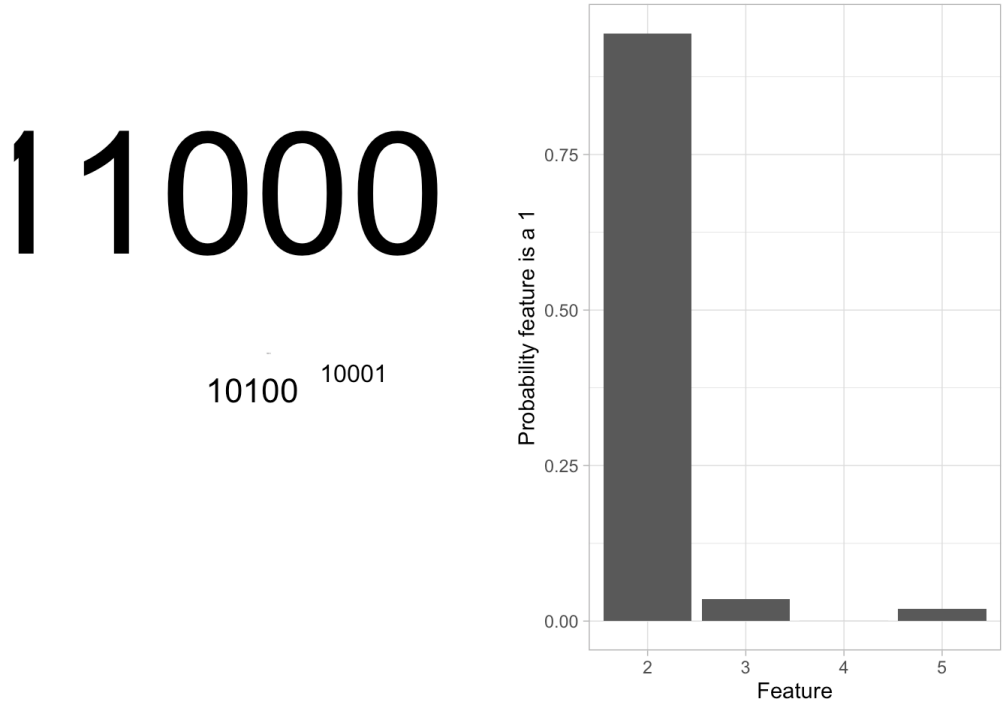
```
## Warning in wordcloud_boxes(data_points = points_valid_first, boxes = boxes, :
## Some words could not fit on page. They have been placed at their original
## positions.
```



Here, we see a word cloud of possible states that could correspond to our observation, and a set of probabilities associated with each *feature* being 1. That is, it's highly likely that feature 2 is 1, and less likely that feature 5 is 1. But doesn't this assume something about how far we've come on the hypercube?

Yes – by default this assumes that all "levels" of the hypercube – all counts of "1" that are possible, given our uncertainty – are equally likely. But we might be more interested in specifying something about how many features we're likely to have acquired. We can do this by specifying weights for each "level":

```
prediction.hidden = predictHiddenVals(my.post, c(1,2,2,2,2), level.weight=c(0,0,1,0,0,0))
plotHypercube.prediction(prediction.hidden)
```



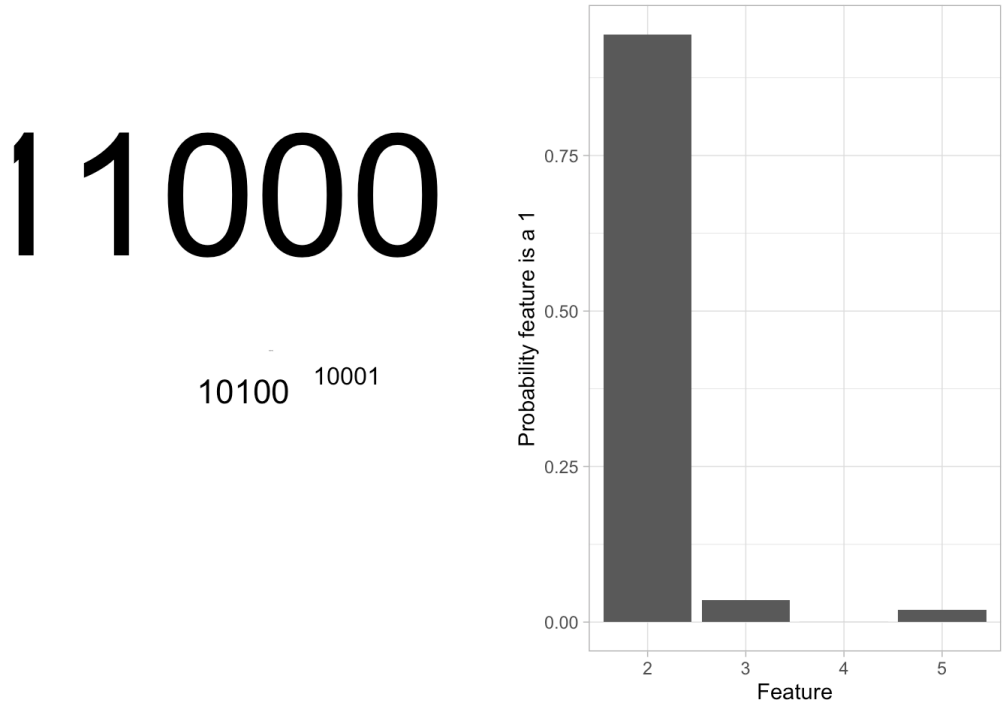
Here, we're saying that we believe there's only one more "1" in the ?s – and we see a correspondingly strong prediction about where that "1" is.

We can be more agnostic, and the predictions are shaped accordingly, like below. We might want our prediction to mirror how many “levels” we saw in our original data, for example.

```
predictHiddenVals(my.post, c(1,2,2,2,2), level.weight=c(0,0,1,1,1,0))
```

```
## $state.probs
##    state level     raw.prob   level.prob         prob
## 1  10000     1 4.137175e-01 1.000000e+00 0.000000e+00
## 2  11000     2 3.995272e-01 9.448435e-01 3.132204e-01
## 3  10100     2 1.518164e-02 3.590312e-02 1.190207e-02
## 4  11100     3 4.082266e-01 9.583788e-01 3.200405e-01
## 5  10010     2 3.734453e-05 8.831623e-05 2.927728e-05
## 6  11010     3 4.692642e-03 1.101674e-02 3.678926e-03
## 7  10110     3 2.377989e-05 5.582718e-05 1.864290e-05
## 8  11110     4 4.117017e-01 9.647575e-01 3.227649e-01
## 9  10001     2 8.103965e-03 1.916510e-02 6.353327e-03
## 10 11001     3 7.296222e-03 1.712907e-02 5.720074e-03
## 11 10101     3 2.624863e-03 6.162296e-03 2.057834e-03
## 12 11101     4 1.081964e-02 2.535411e-02 8.482357e-03
## 13 10011     3 3.091286e-03 7.257300e-03 2.423499e-03
## 14 11011     4 7.843709e-04 1.838049e-03 6.149292e-04
## 15 10111     4 3.435410e-03 8.050338e-03 2.693285e-03
## 16 11111     5 1.000000e+00 1.000000e+00 0.000000e+00
##
## $locus.probs
##   locus      prob
## 1     2 0.9745221
## 2     3 0.6679596
## 3     4 0.3322235
## 4     5 0.0283453
```

```
plotHypercube.prediction(prediction.hidden)
```



Uncertainty in observation times

HyperTraPS-CT allows uncertainty in transition timings to be captured. In the run above, we specified a precise timing for each transition, by giving a zero-width time window for each observation (start times = end times). This time window is flexible. We can allow it to have infinite width, in which case absolute timings are meaningless and we just infer the orderings of events. Or we can specify a finite time window, allowing a transition to take any time within that window, to capture uncertainty in observation timings.

```
# precisely specified timings, as above
my.post.time.precise = HyperTraPS(m.2, initialstates = m.1,
                                  starttimes = times, endtimes = times,
                                  limited_output = 1,
                                  featurenames = c("A", "B", "C", "D", "E"));
```

```
## One likelihood estimation took 1.222000e-03 seconds.
## Initial likelihood is -9.895393e+01
## Second guess is -9.895393e+01
## This code (1000 steps) will probably take around 1.222 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

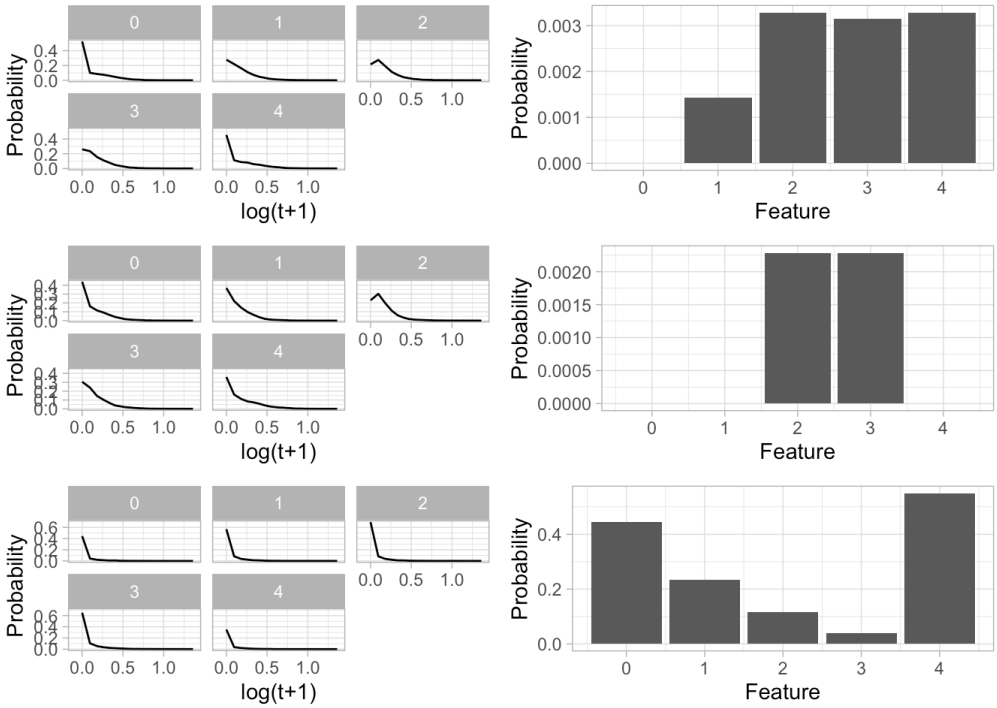
```
# infinite width time window for transitions (equivalent to just inferring ordering)
my.post.time.inf = HyperTraPS(m.2, initialstates = m.1,
                              starttimes = times*0, endtimes = times*Inf,
                              limited_output = 1,
                              featurenames = c("A", "B", "C", "D", "E"));
```

```
## One likelihood estimation took 1.017000e-03 seconds.
## Initial likelihood is -4.787492e+01
## Second guess is -4.787492e+01
## This code (1000 steps) will probably take around 1.017 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

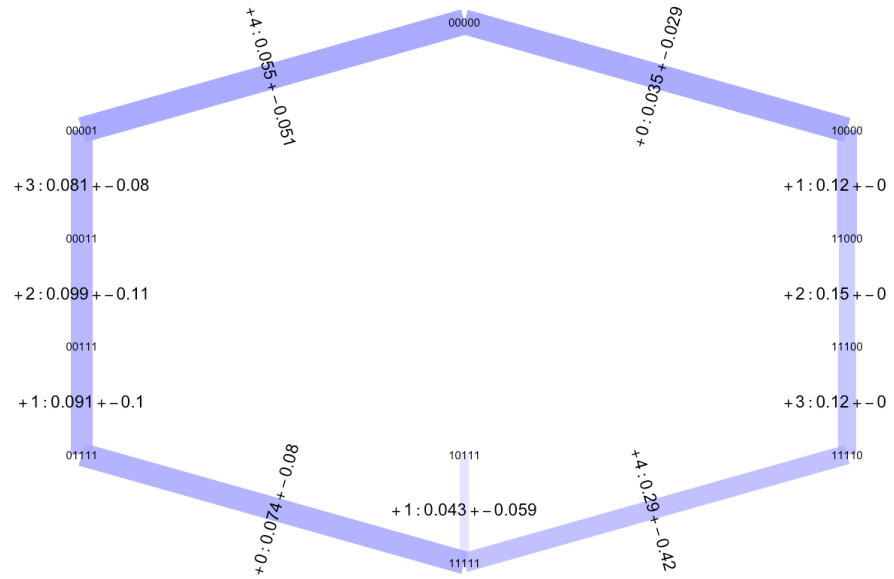
```
# finite time window for each uncertain transition time
my.post.time.uncertain = HyperTraPS(m.2, initialstates = m.1,
                    starttimes = times*0.25, endtimes = times*4,
                    limited_output = 1,
                    featurenames = c("A", "B", "C", "D", "E"));
```

```
## One likelihood estimation took 1.134000e-03 seconds.
## Initial likelihood is -5.456530e+01
## Second guess is -5.456530e+01
## This code (1000 steps) will probably take around 1.134 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

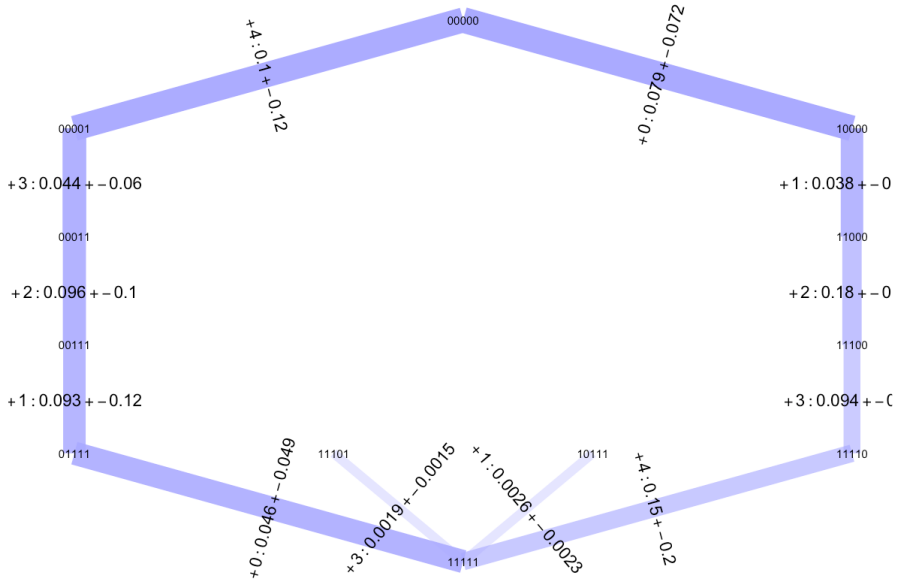
```
ggarrange(plotHypercube.timehists(my.post.time.precise, t.thresh=3),
          plotHypercube.timehists(my.post.time.uncertain, t.thresh=3),
          plotHypercube.timehists(my.post.time.inf, t.thresh=3),
          nrow=3)
```



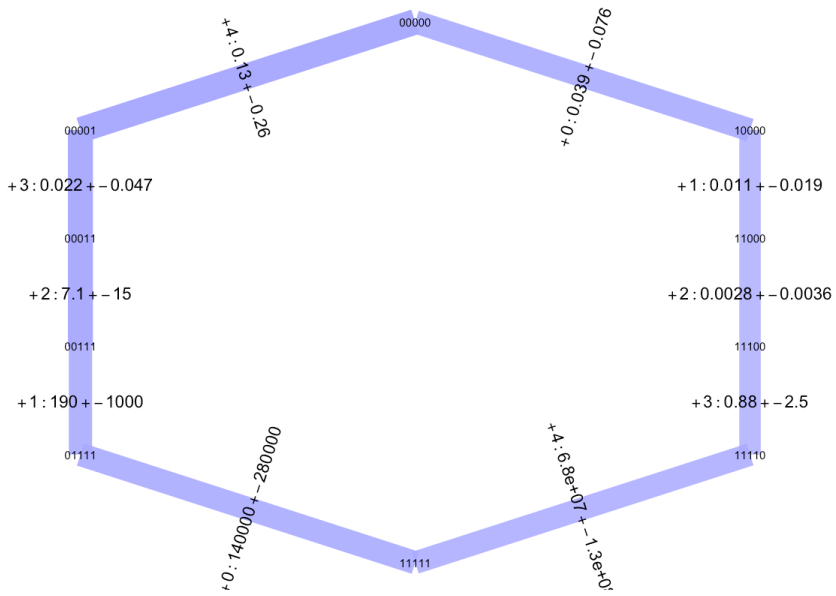

```
plotHypercube.sampledgraph2(my.post.time.precise, thresh=0.1, use.arc=FALSE, edge.label.size=3) + theme(legend.po
sition="none") + expand_limits(x = c(-0.1, 1.1))
```




```
plotHypercube.sampledgraph2(my.post.time.uncertain, thresh=0.1, use.arc=FALSE, edge.label.size=3) + theme(legen
d.position="none") + expand_limits(x = c(-0.1, 1.1))
```




```
plotHypercube.sampledgraph2(my.post.time.inf, thresh=0.1, use.arc=FALSE, edge.label.size=3) + theme(legend.positi
on="none") + expand_limits(x = c(-0.1, 1.1))
```



The summary plots here reflects the different inferences about acquisition timescales that come from the approaches with different time windows.

Input/output between R data structure and file format

We can write the output of HyperTraPS to a set of files for storage and later retrieval

```
writeHyperinf(my.post, "simpledemo", my.post$L, postlabel = "simpledemo", fulloutput=TRUE)
```

We can retrieve output from files, which may have been previously written as above, or may have come from running HyperTraPS at the command line.

```
my.post.r = readHyperinf("simpledemo", postlabel = "simpledemo", fulloutput=TRUE)
```

Other functional examples

HyperTraPS allows substantial flexibility in how evolutionary pathways are inferred, how likelihoods are estimated, and other aspects of the approach.

If we want to sacrifice some accuracy in estimating likelihood for computational speed, we can run an example with fewer walkers sampling pathways (`walkers`)

```
my.post.sparse = HyperTraPS(m.2, initialstates = m.1,
                            starttimes = times, endtimes = times,
                            featurenames = c("A", "B", "C", "D", "E"),
                            walkers = 2,
                            limited_output = 1)
```

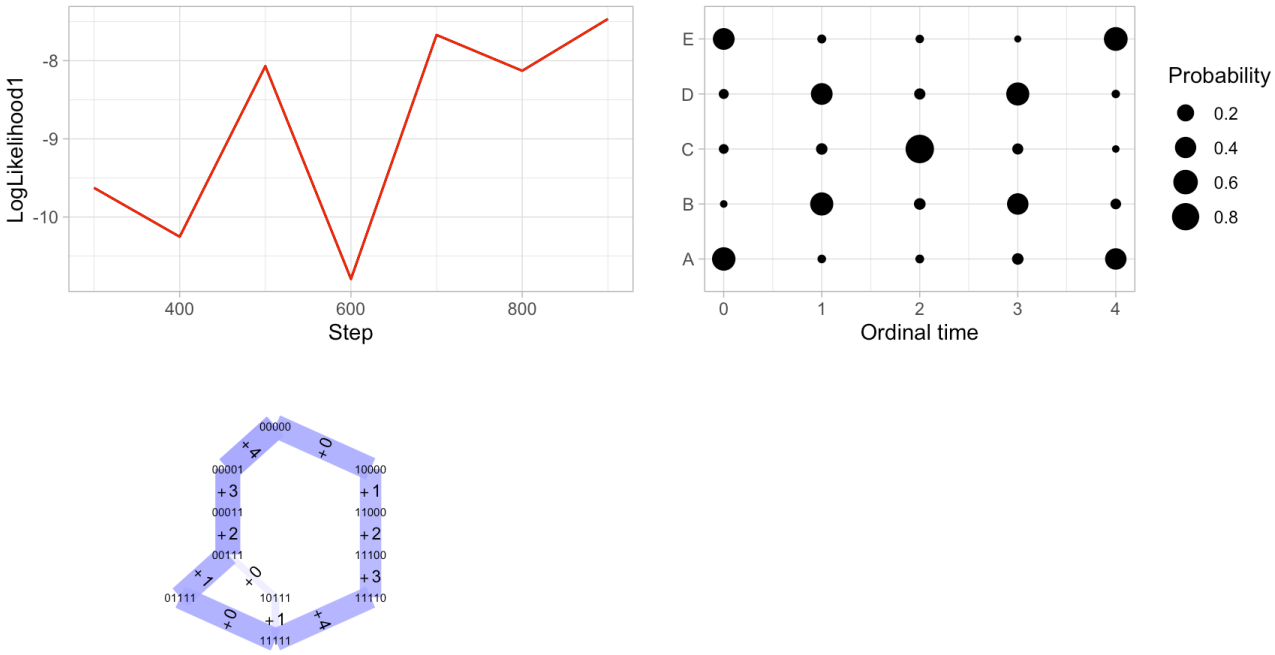
```
## One likelihood estimation took 5.100000e-05 seconds.
## Initial likelihood is -9.895393e+01
## Second guess is -9.895393e+01
## This code (1000 steps) will probably take around 0.051 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

The original, discrete-time HyperTraPS approach is recovered if we don't use timing information

```
my.post.dt = HyperTraPS(m.2, initialstates = m.1,
                        featurenames = c("A", "B", "C", "D", "E"),
                        limited_output = 1)
```

```
## One likelihood estimation took 8.870000e-04 seconds.
## Initial likelihood is -4.787492e+01
## Second guess is -4.787492e+01
## This code (1000 steps) will probably take around 0.887 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

```
plotHypercube.summary(my.post.dt, continuous.time = FALSE)
```

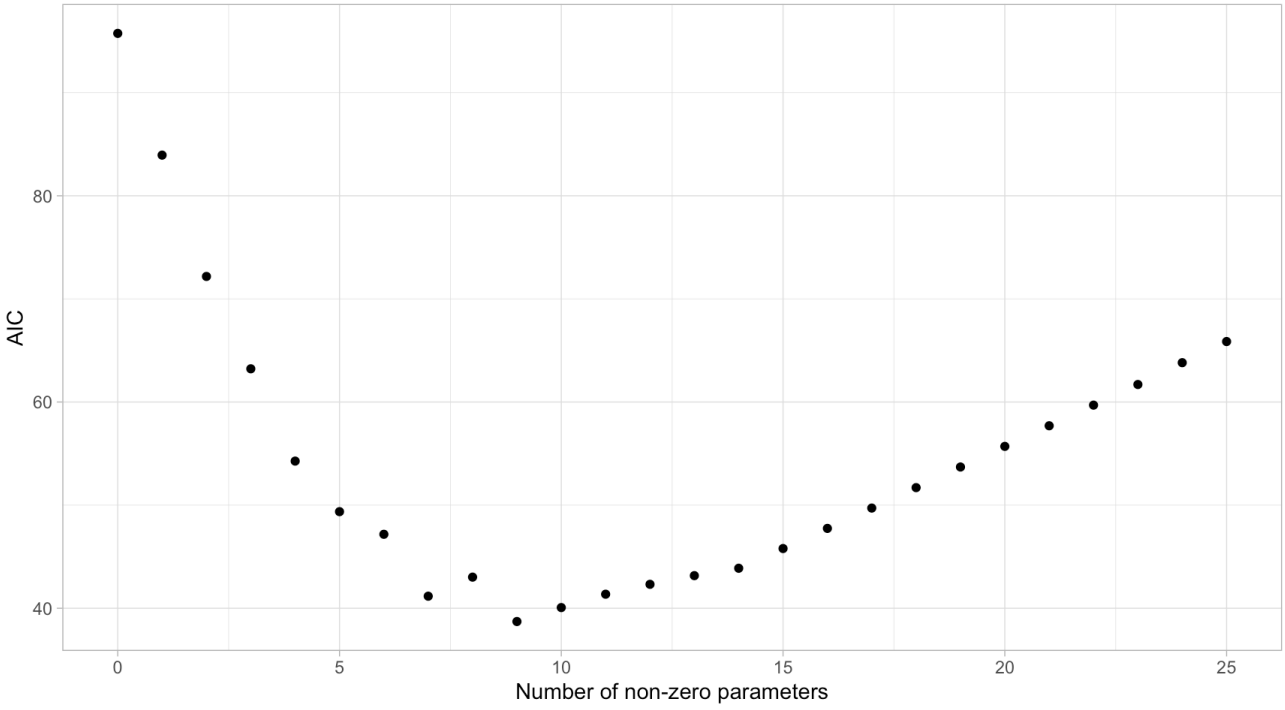


We can ask HyperTraPS to perform stepwise regularisation after fitting a model, pruning extraneous parameters (`regularise`). The regularisation plot shows how the information criterion behaves as we prune back parameters – the minimum gives us our optimal model.

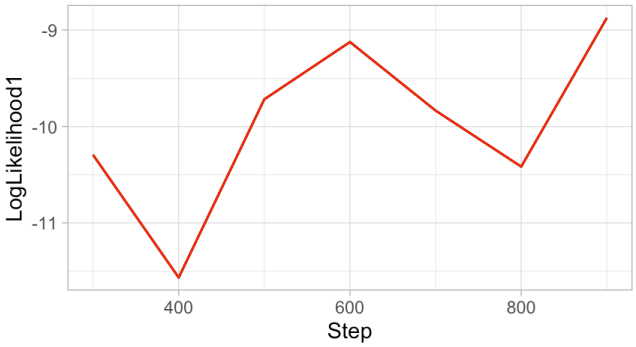
```
my.post.regularise = HyperTraPS(m.2, initialstates = m.1, regularise = 1,
                                walkers = 20,
                                limited_output = 1)
```

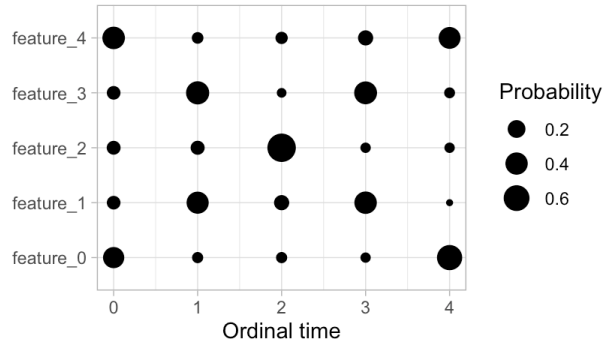
```
## One likelihood estimation took 1.260000e-04 seconds.
## Initial likelihood is -4.787492e+01
## Second guess is -4.787492e+01
## This code (1000 steps) will probably take around 0.126 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

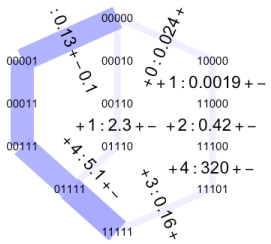
```
plotHypercube.regularisation(my.post.regularise)
```

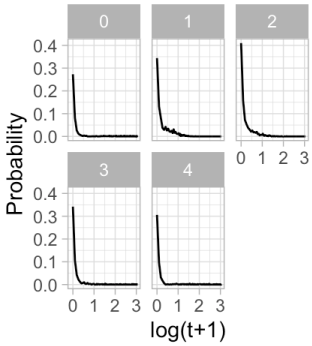


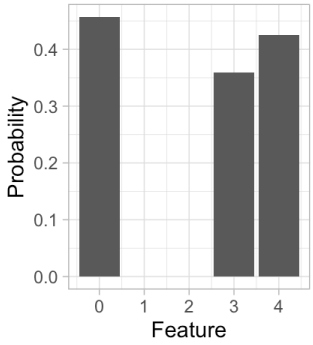

```
plotHypercube.summary(my.post.regularise)
```









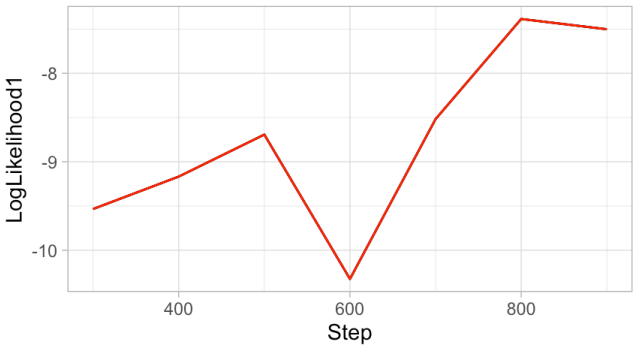


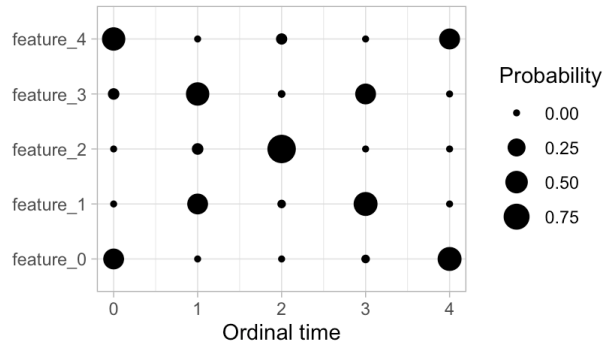
We can use simulated annealing to get a maximum likelihood estimate rather than a Bayesian picture (`sa`)

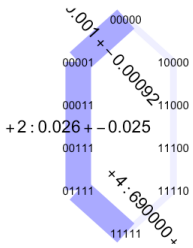
```
my.post.sa = HyperTraPS(m.2, initialstates = m.1, sa = 1,
                        limited_output = 1)
```

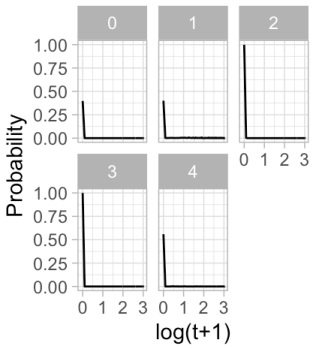
```
## One likelihood estimation took 1.020000e-03 seconds.
## Initial likelihood is -4.787492e+01
## Second guess is -4.787492e+01
## This code (1000 steps) will probably take around 1.020 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

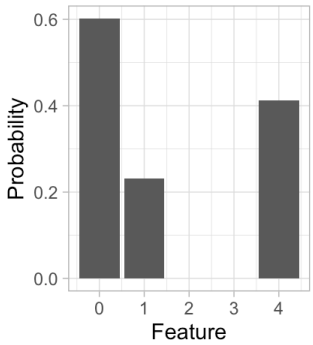
```
plotHypercube.summary(my.post.sa)
```









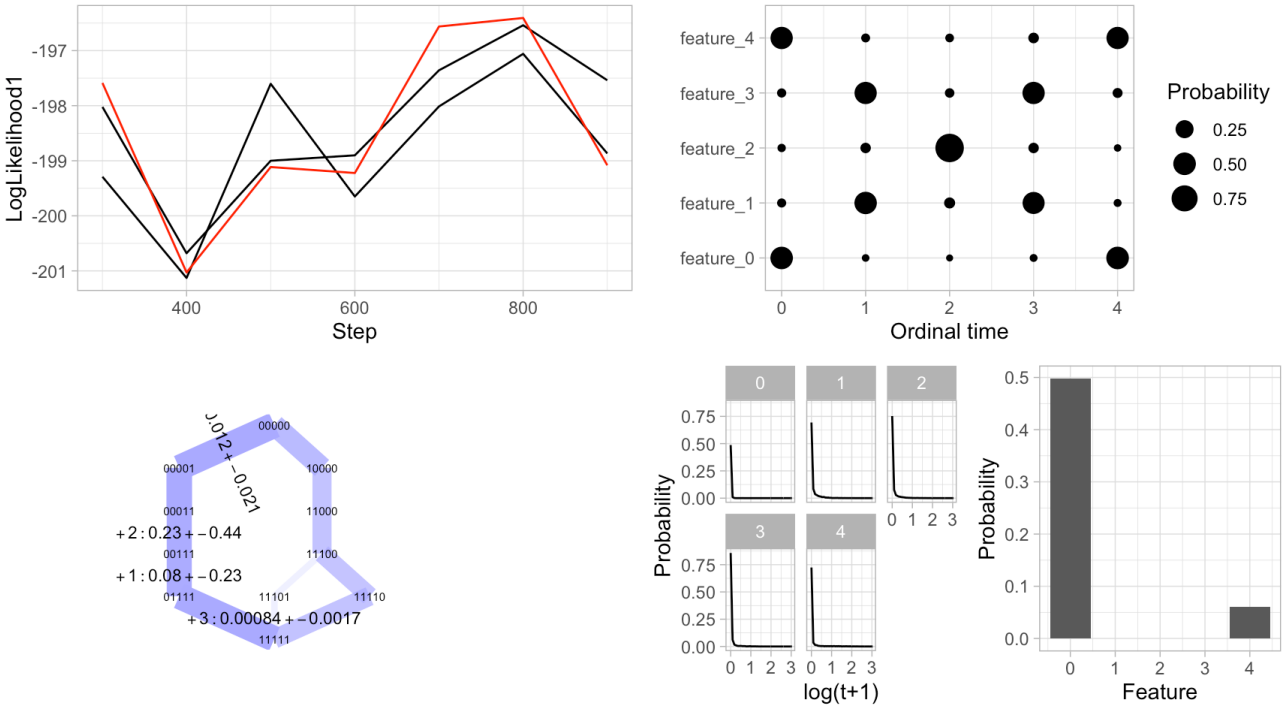


We can also use “phenotype landscape inference” – an unbiased sampling method, unlike the (corrected) bias sampling in HyperTraPS (`PLI`). This takes longer but is more flexible with uncertain data

```
my.post.pli = HyperTraPS(m.2, initialstates = m.1, pli = 1,
                         limited_output = 1)
```

```
## One likelihood estimation took 6.257000e-03 seconds.
## Initial likelihood is -2.912900e+02
## Second guess is -2.906914e+02
## This code (1000 steps) will probably take around 6.257 seconds (0.002 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

```
plotHypercube.summary(my.post.pli)
```

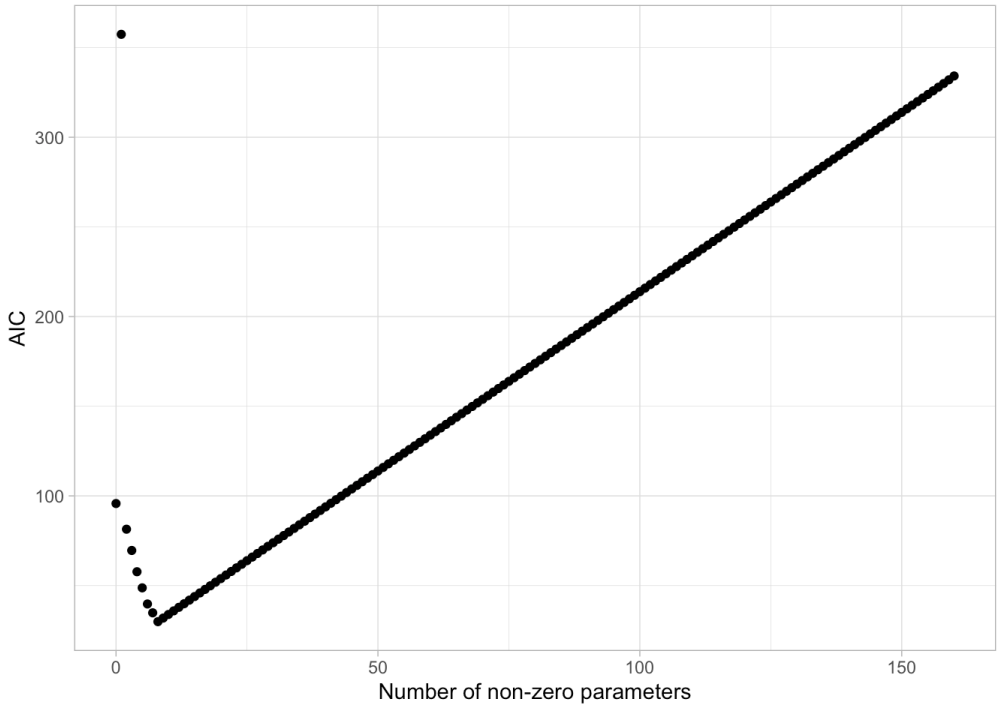


HyperTraPS supports different parameter structures (`model`). By default we use an "L2" parameterisation, where each feature can individually influence the acquisition of every other feature. "L1" has features completely independent; "L0" has all features identical. "L3" allows *pairs* of features to influence each feature's acquisition; "L4" allows *triplets* of features to influence each feature's acquisition. This approach, labelled -1, allows every edge on the hypercube to have its own independent parameters – corresponding to the most flexible possible picture, where arbitrary sets of features influence other features. We then regularise as above (`regularise`) to remove extraneous parameters

```
my.post.bigmodel.regularise = HyperTraPS(m.2, initialstates = m.1, model = -1,
                                         regularise = 1, walkers = 20,
                                         limited_output = 1)
```

```
## One likelihood estimation took 1.150000e-04 seconds.
## Initial likelihood is -4.787492e+01
## Second guess is -4.787492e+01
## This code (1000 steps) will probably take around 0.115 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

```
plotHypercube.regularisation(my.post.bigmodel.regularise)
```



Here's another example of model choice. The data is now generated by a process where *pairs* of features influence the acquisition of other features. To capture these interactions, an "L^2" picture (`model = 2` , where each feature *individually* influences each other feature) is insufficient – an "L^3" picture (`model = 3`) allowing pair influence is needed. The full parameterisation, where every edge on the hypercube transition network (`model = -1`) has an independent parameter, can also capture this behaviour.

```
logic.mat = readLines("RawData/old-hi-order.txt")
logic.mat = do.call(rbind, lapply(strsplit(logic.mat, ""), as.numeric))
logic.mat = rbind(logic.mat, logic.mat)
logic.mat.i = readLines("RawData/old-hi-order-init.txt")
logic.mat.i = do.call(rbind, lapply(strsplit(logic.mat.i, ""), as.numeric))
logic.mat.i = rbind(logic.mat.i, logic.mat.i)
logic.starts = logic.mat.i
logic.ends = logic.mat

logic.post.m1 = HyperTraPS(logic.ends, initialstates = logic.starts, length = 4, model = -1, walkers = 20, limite
d_output = 1)
```

```
## One likelihood estimation took 1.270000e-04 seconds.
## Initial likelihood is -6.103342e+01
## Second guess is -6.103342e+01
## This code (10000 steps) will probably take around 1.270 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 - 1000 - 1100 - 1200 - 1300 - 1400 - 1500 - 1600 - 170
0 - 1800 - 1900 - 2000 - 2100 - 2200 - 2300 - 2400 - 2500 - 2600 - 2700 - 2800 - 2900 - 3000 - 3100 - 3200 - 3300
- 3400 - 3500 - 3600 - 3700 - 3800 - 3900 - 4000 - 4100 - 4200 - 4300 - 4400 - 4500 - 4600 - 4700 - 4800 - 4900 -
5000 - 5100 - 5200 - 5300 - 5400 - 5500 - 5600 - 5700 - 5800 - 5900 - 6000 - 6100 - 6200 - 6300 - 6400 - 6500 - 6
600 - 6700 - 6800 - 6900 - 7000 - 7100 - 7200 - 7300 - 7400 - 7500 - 7600 - 7700 - 7800 - 7900 - 8000 - 8100 - 82
00 - 8300 - 8400 - 8500 - 8600 - 8700 - 8800 - 8900 - 9000 - 9100 - 9200 - 9300 - 9400 - 9500 - 9600 - 9700 - 980
0 - 9900 -
```

```
logic.post.1 = HyperTraPS(logic.ends, initialstates = logic.starts, length = 4, model = 1, walkers = 20, limited_
output = 1)
```

```
## One likelihood estimation took 1.350000e-04 seconds.
## Initial likelihood is -6.103342e+01
## Second guess is -6.103342e+01
## This code (10000 steps) will probably take around 1.350 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 - 1000 - 1100 - 1200 - 1300 - 1400 - 1500 - 1600 - 170
0 - 1800 - 1900 - 2000 - 2100 - 2200 - 2300 - 2400 - 2500 - 2600 - 2700 - 2800 - 2900 - 3000 - 3100 - 3200 - 3300
- 3400 - 3500 - 3600 - 3700 - 3800 - 3900 - 4000 - 4100 - 4200 - 4300 - 4400 - 4500 - 4600 - 4700 - 4800 - 4900 -
5000 - 5100 - 5200 - 5300 - 5400 - 5500 - 5600 - 5700 - 5800 - 5900 - 6000 - 6100 - 6200 - 6300 - 6400 - 6500 - 6
600 - 6700 - 6800 - 6900 - 7000 - 7100 - 7200 - 7300 - 7400 - 7500 - 7600 - 7700 - 7800 - 7900 - 8000 - 8100 - 82
00 - 8300 - 8400 - 8500 - 8600 - 8700 - 8800 - 8900 - 9000 - 9100 - 9200 - 9300 - 9400 - 9500 - 9600 - 9700 - 980
0 - 9900 -
```

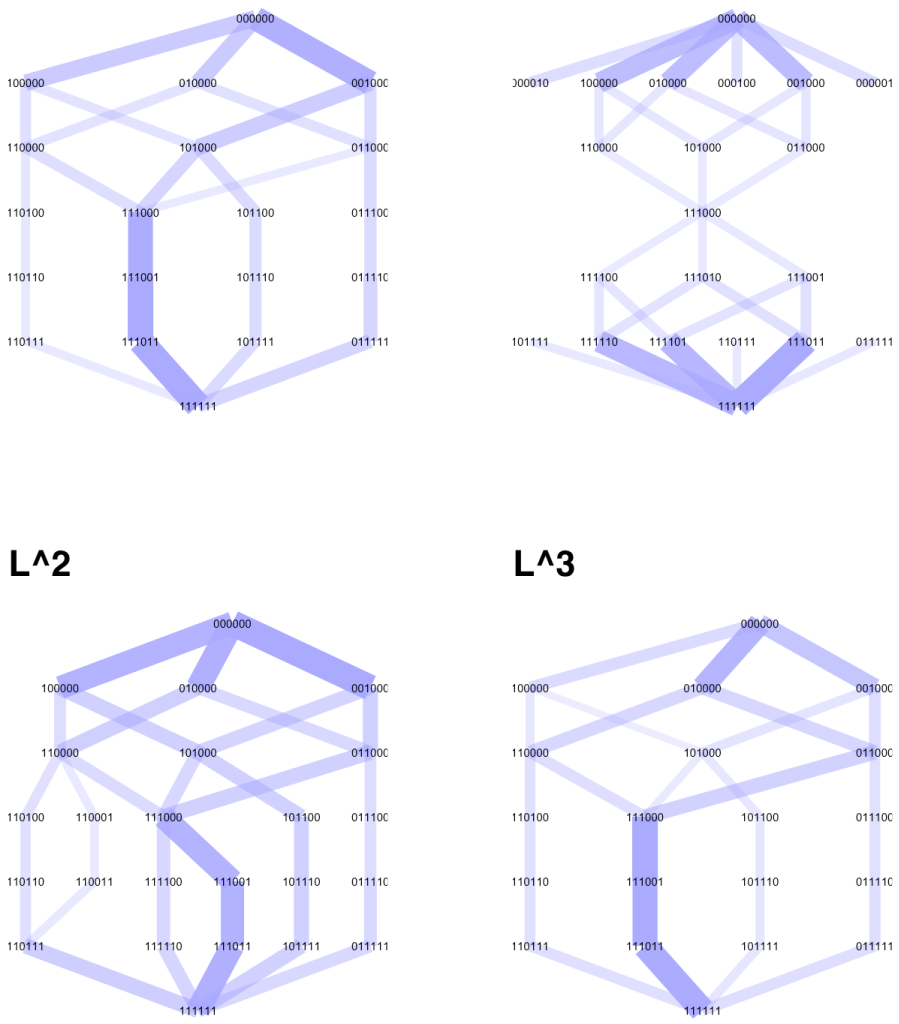
```
logic.post.2 = HyperTraPS(logic.ends, initialstates = logic.starts, length = 4, model = 2, walkers = 20, limited_
output = 1)
```

```
## One likelihood estimation took 1.380000e-04 seconds.
## Initial likelihood is -6.103342e+01
## Second guess is -6.103342e+01
## This code (10000 steps) will probably take around 1.380 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 - 1000 - 1100 - 1200 - 1300 - 1400 - 1500 - 1600 - 170
0 - 1800 - 1900 - 2000 - 2100 - 2200 - 2300 - 2400 - 2500 - 2600 - 2700 - 2800 - 2900 - 3000 - 3100 - 3200 - 3300
- 3400 - 3500 - 3600 - 3700 - 3800 - 3900 - 4000 - 4100 - 4200 - 4300 - 4400 - 4500 - 4600 - 4700 - 4800 - 4900 -
5000 - 5100 - 5200 - 5300 - 5400 - 5500 - 5600 - 5700 - 5800 - 5900 - 6000 - 6100 - 6200 - 6300 - 6400 - 6500 - 6
600 - 6700 - 6800 - 6900 - 7000 - 7100 - 7200 - 7300 - 7400 - 7500 - 7600 - 7700 - 7800 - 7900 - 8000 - 8100 - 82
00 - 8300 - 8400 - 8500 - 8600 - 8700 - 8800 - 8900 - 9000 - 9100 - 9200 - 9300 - 9400 - 9500 - 9600 - 9700 - 980
0 - 9900 -
```

```
logic.post.3 = HyperTraPS(logic.ends, initialstates = logic.starts, length = 4, model = 3, walkers = 20, limited_
output = 1)
```

```
## One likelihood estimation took 2.290000e-04 seconds.
## Initial likelihood is -6.103342e+01
## Second guess is -6.103342e+01
## This code (10000 steps) will probably take around 2.290 seconds (0.001 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 - 1000 - 1100 - 1200 - 1300 - 1400 - 1500 - 1600 - 170
0 - 1800 - 1900 - 2000 - 2100 - 2200 - 2300 - 2400 - 2500 - 2600 - 2700 - 2800 - 2900 - 3000 - 3100 - 3200 - 3300
- 3400 - 3500 - 3600 - 3700 - 3800 - 3900 - 4000 - 4100 - 4200 - 4300 - 4400 - 4500 - 4600 - 4700 - 4800 - 4900 -
5000 - 5100 - 5200 - 5300 - 5400 - 5500 - 5600 - 5700 - 5800 - 5900 - 6000 - 6100 - 6200 - 6300 - 6400 - 6500 - 6
600 - 6700 - 6800 - 6900 - 7000 - 7100 - 7200 - 7300 - 7400 - 7500 - 7600 - 7700 - 7800 - 7900 - 8000 - 8100 - 82
00 - 8300 - 8400 - 8500 - 8600 - 8700 - 8800 - 8900 - 9000 - 9100 - 9200 - 9300 - 9400 - 9500 - 9600 - 9700 - 980
0 - 9900 -
```

```
ggarrange(plotHypercube.graph(logic.post.m1) + ggtitle("All edges") + theme(legend.position="none"),
          plotHypercube.graph(logic.post.1) + ggtitle("L") + theme(legend.position="none"),
          plotHypercube.graph(logic.post.2)+ ggtitle("L^2") + theme(legend.position="none"),
          plotHypercube.graph(logic.post.3)+ ggtitle("L^3") + theme(legend.position="none"))
```



We can see that the structures inferred by the “full” and “L^3” models are the same, while the “L^2” (and inappropriate “L^1”) will either introduce extraneous transitions or fail to capture the true ones.

Prior information

The Bayesian implementations of HyperTraPS-CT allow prior information to be included in the inference process. We can do this by placing bounds on the values that the parameters in our model are allowed to take. Here, we first assign wide priors (-10 to 10 in log space) to all the 25 parameters in our model. Then we go through the parameters corresponding to the base rates for each feature: these are on the diagonal of the parameter matrix. We restrict these values to be -10, except for the base rates for feature 1 and feature 5. We thus enforce feature 1 > feature 5 >> other features.

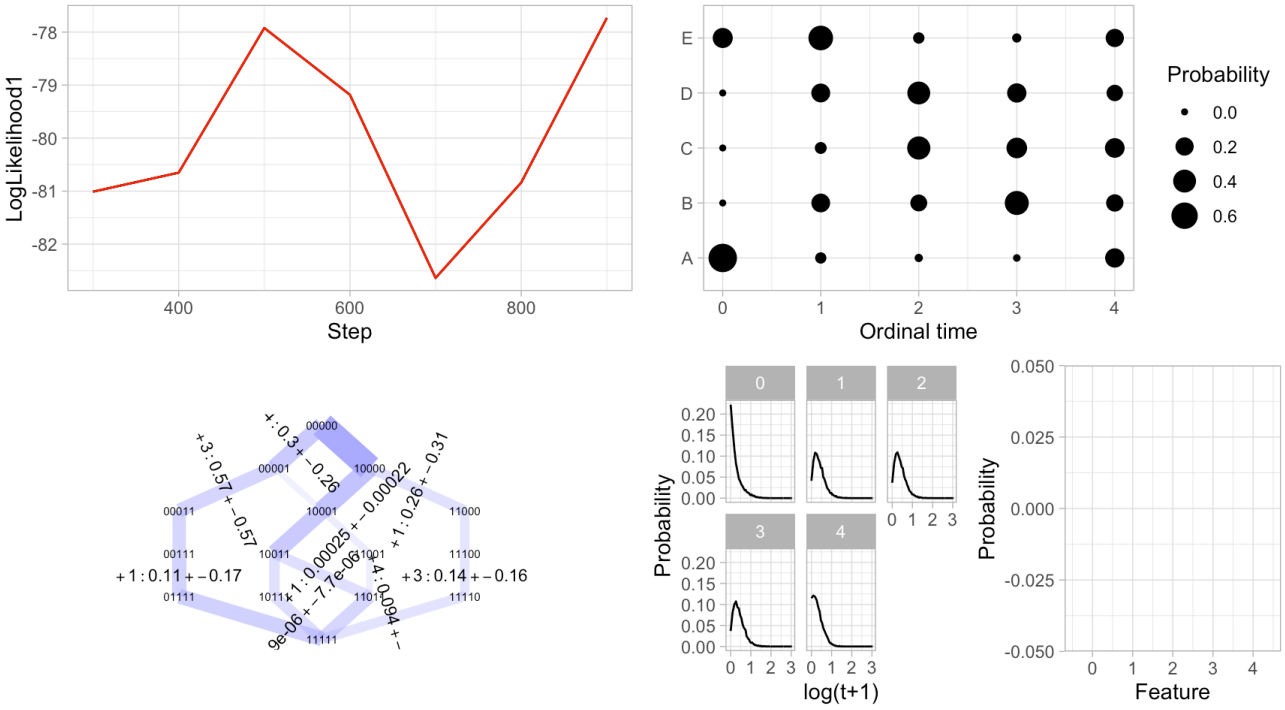
```
priors = matrix(0, ncol=2, nrow=5*5)
priors[,1] = -10
priors[,2] = 10
for(i in 0:4) {
  priors[i*5+i+1,1] = -10
  priors[i*5+i+1,2] = -10
}
priors[0*5+0+1,1] = 1
priors[0*5+0+1,2] = 1
priors[4*5+4+1,1] = 0
priors[4*5+4+1,2] = 0
```

Running and visualising the posteriors shows this effect:

```
my.post.priors = HyperTraPS(m.2, initialstates = m.1,
                    starttimes = times, endtimes = times,
                    priors = priors,
                    featurenames = c("A", "B", "C", "D", "E"),
                    limited_output = 1)
```

```
## One likelihood estimation took 1.119000e-03 seconds.
## Initial likelihood is -4.143904e+02
## Second guess is -4.143904e+02
## This code (1000 steps) will probably take around 1.119 seconds (0.000 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

```
plotHypercube.summary(my.post.priors)
```



Scientific examples

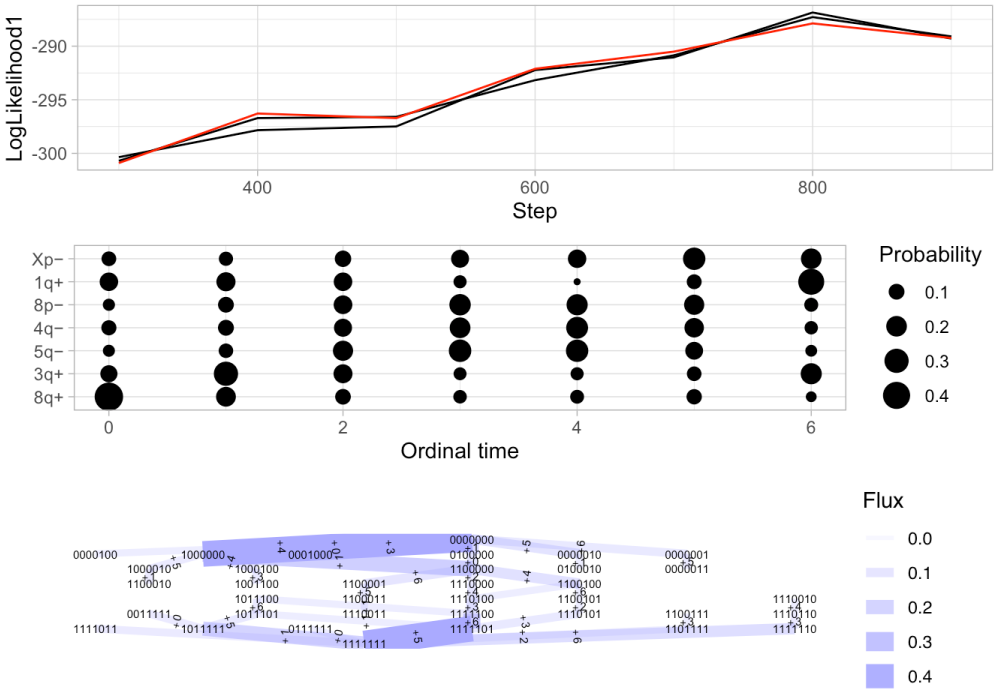
A set of short-form examples from past studies – these should run in a few minutes and give approximations to the original results. In each case we read the observed states from a file (these are present in different formats, so different curation steps are involved in each case), and feature labels from another file. Then we run HyperTraPS and plot some summaries of the output.

1. Ovarian cancer case study: traits are chromosomal aberrations, observations are independent patient samples.

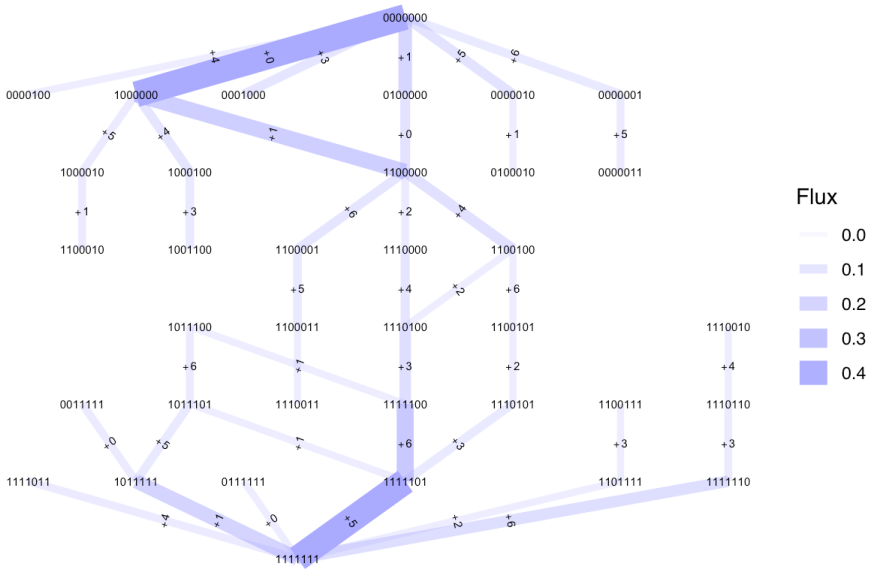
```
cgh.mat = readLines("RawData/ovarian.txt")
cgh.mat = do.call(rbind, lapply(strsplit(cgh.mat, ""), as.numeric))
cgh.names = as.vector(read.table("RawData/ovarian-names.txt", sep=","))[[1]]

my.post.cgh = HyperTraPS(cgh.mat,
                         length = 3,
                         featurenames = cgh.names,
                         limited_output = 1)
```

```
## One likelihood estimation took 9.950000e-03 seconds.
## Initial likelihood is -3.213415e+02
## Second guess is -3.213415e+02
## This code (1000 steps) will probably take around 9.950 seconds (0.003 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 -
```

```
ggarrange(plotHypercube.lik.trace(my.post.cgh),
          plotHypercube.bubbles(my.post.cgh, reorder=TRUE),
          plotHypercube.sampledgraph2(my.post.cgh, no.times=TRUE), nrow=3)
```




```
plotHypercube.sampledgraph2(my.post.cgh, no.times=TRUE)
```



2. C4 photosynthesis case study: traits are physical/genetic features associated with C4, observations are (incomplete) phylogenetically independent intermediate species.

This case study involves some uncertain data, which are present as “2”s in the source data – labelling features which may be 0 or 1.

```
c4.mat = as.matrix(read.table("RawData/c4-curated.csv", sep=","))
c4.names = as.vector(read.table("RawData/c4-trait-names.txt", sep=","))[[1]]

my.post.c4 = HyperTraPS(c4.mat,
                        length = 4,
                        featurenames = c4.names,
                        limited_output = 1)
```

```
## One likelihood estimation took 3.960600e-02 seconds.
## Initial likelihood is -2.694082e+02
## Second guess is -2.695834e+02
## This code (10000 steps) will probably take around 396.060 seconds (0.110 hours) to complete.
##
## 0 - 100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900 - 1000 - 1100 - 1200 - 1300 - 1400 - 1500 - 1600 - 170
0 - 1800 - 1900 - 2000 - 2100 - 2200 - 2300 - 2400 - 2500 - 2600 - 2700 - 2800 - 2900 - 3000 - 3100 - 3200 - 3300
- 3400 - 3500 - 3600 - 3700 - 3800 - 3900 - 4000 - 4100 - 4200 - 4300 - 4400 - 4500 - 4600 - 4700 - 4800 - 4900 -
5000 - 5100 - 5200 - 5300 - 5400 - 5500 - 5600 - 5700 - 5800 - 5900 - 6000 - 6100 - 6200 - 6300 - 6400 - 6500 - 6
600 - 6700 - 6800 - 6900 - 7000 - 7100 - 7200 - 7300 - 7400 - 7500 - 7600 - 7700 - 7800 - 7900 - 8000 - 8100 - 82
00 - 8300 - 8400 - 8500 - 8600 - 8700 - 8800 - 8900 - 9000 - 9100 - 9200 - 9300 - 9400 - 9500 - 9600 - 9700 - 980
0 - 9900 -
```

```
plotHypercube.bubbles(my.post.c4, reorder=TRUE)
```