Vistla walkthrough

Miron B. Kursa

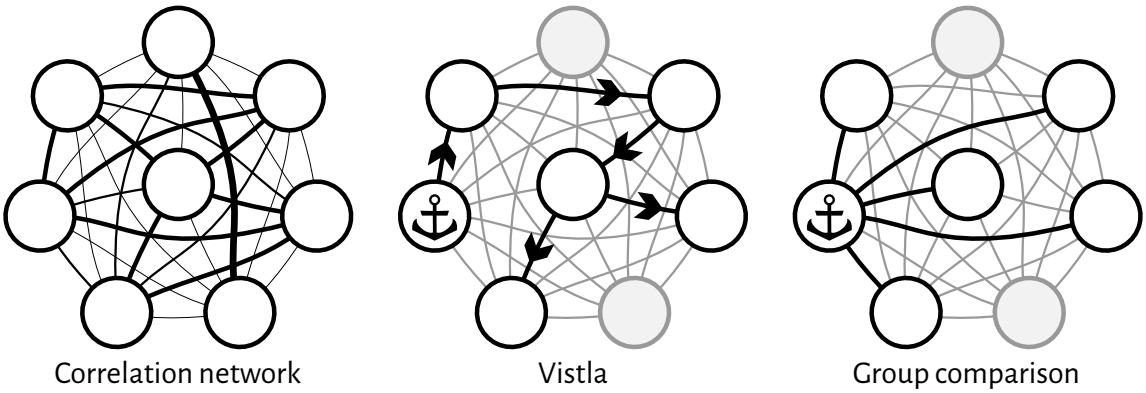
May 26, 2026

1 Motivation

Vistla is a tool for tracing information spread in complex systems. As a typical input, it expects a data frame collecting a number of rows corresponding to observations of said system and columns corresponding to features expressing states of various system's components. One of the features, which we will later call an *anchor*, represents the intervention applied to the system. For instance, such a data set may be a collection of gene expression levels in several biological samples of two groups, one perturbed by a drug administration and the other one left as control. In this case, each feature would be an expression of one gene, and the anchor feature would represent the group.

With basic approaches, one could be interested in analysing correlations between each pair of features, to roughly sketch a network of interactions within the system, or in investigating relations between the anchor and each of the features, to find genes which expression seem to be altered by the drug. The problem with the first approach is that the resulting network is likely to be very dense and hard to comprehend, also due to a fact that simple inference methods may directly join a pair of features that are in fact only connected indirectly.

On the other hand, a set of differently expressed genes is a simpler outcome, obviously strongly connected to the intervention which is a main axis of interest in the experiment. Still, this set lacks any deeper structure, hence requires in-depth analysis for interpretation. Moreover, both approaches will miss edges resulting from more complex, multivariate interactions.



In turn, Vistla provides a middle ground here; anchor feature is used as a starting point from which the algorithm traces sequences of features which seem to conduct information about the intervention and spread it across the system. The union of these paths is a tree, hence a simpler and clearer structure than a generic graph, but still expressing the interactions between features. The tree is rooted in the anchor, our feature of interest, so it also brings the focus of the differential approach.

The path tracing is expressed as an optimisation problem that involves maximising criterion based on information theory; the algorithm is guaranteed to solve it exactly, although the solution may not be unique, especially considering inevitable errors in information estimation (see Section 3.3). The detailed description of the method can be found in [3].

Vistla can also solve the reverse problem, i.e., when the information in the anchor is rather produced than ingested by the system; refer to Section 3.2 for more information.

2 Usage

2.1 Prerequisites

The `vistla` package can be obtained from CRAN, preferably using built-in R package installation mechanisms, for instance with the command

```
> install.packages("vistla")
```

The development version can be obtained from the project repository `https://gitlab.com/mbq/vistla`.
Once the package is installed, it can be loaded with

```
> library(vistla)
```

For this example, we will use the synthetic example dataset, *junction*. It is bundled with the package, we can import it into the global environment with

```
> data(junction)
```

As mentioned earlier, for an input Vistla expects a `data.frame`, in which columns represent components of the system, while rows correspond to individual observations. Depending on the selected information estimator, Vistla may expect different types of data; in this example, we will be using the default maximum likelihood (ML) estimator, which requires discrete input, that is columns that are factors (or logical). We can inspect junction dataset to confirm that it is already in the proper form

```
> head(junction)
```

```
   Y  J A1 A2 A3 B1 B2 B3
1 l2 l1 l1 l1 l1 l1 l1 l1
2 l2 l2 l1 l1 l1 l1 l1 l2
3 l2 l1 l1 l1 l1 l1 l1 l1
4 l2 l2 l1 l1 l2 l1 l1 l2
5 l2 l2 l1 l1 l1 l1 l1 l2
6 l2 l2 l1 l1 l1 l1 l1 l2
```

For other data one may want to use some of the discretisation tools available in base R or on CRAN; for a general ordinal data, including numerical and binary features, one may also use Kendall transformation estimator which is bundled with the package. Refer to Section 3.1 for more information.

Junction was generated from a Bayesian network composed of two chains of binary features with a common start: $Y \to A_1 \to A_2 \to J_A \to A_3$ and $Y \to B_1 \to B_2 \to J_B \to B_3$. In the final data J_A and J_B are not present, however; instead, there is a single junction feature $J = J_A \times J_B$, which retains information from both of them.

2.2 Invocation

Vistla algorithm can be applied using the `vistla` function; besides the dataset, we also need to define the anchor feature, source of information traced by Vistla and the root of the tree it is going to build. In a practical problem this may be, for instance, the intervention or perturbation applied to the system to investigate its response. In the junction data, the anchor is the Y feature. We will use the standard R *formula* interface

```
> vistla(Y~.,data=junction)->vistla_result
```

Printing the obtained object will show a short summary of the top paths

```
> vistla_result
```

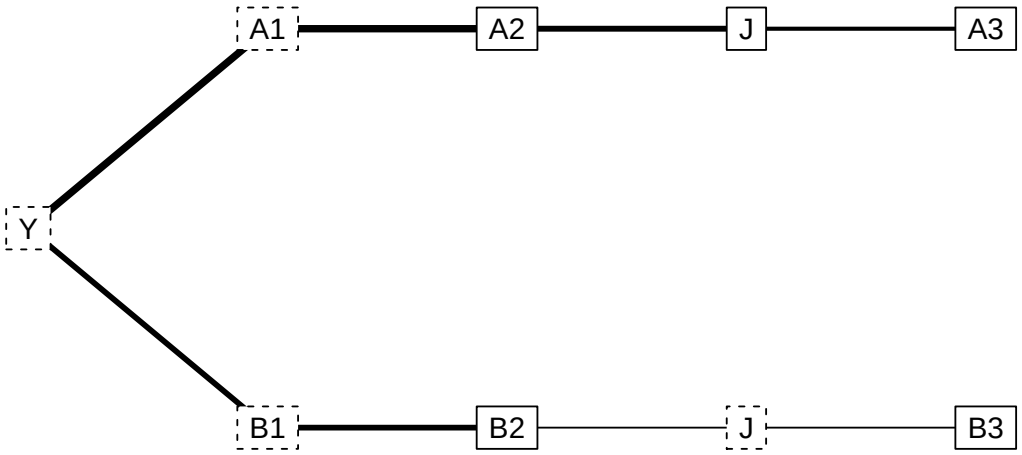
```
        Vistla tree rooted in Y

Paths:
 - A2 (score 0.52) ~ A1
 - J (0.46) ~ A2 ~ A1
 - B2 (0.46) ~ B1
 - A3 (0.4) ~ J ~ A2 ~ A1
 - B3 (0.32) ~ J ~ B2 ~ B1
                 ... ~ Y
```

in the first line we can read that the strongest path leads to A_2, which is connected to the anchor via A_1 and that the score of this path is 0.52 nats. Similarly, the penultimate line reports that the paths to B_3 goes from Y through B_1, B_2 and J, with a score of 0.32 nats.

2.3 Analysis

It is more enlightening, however, to investigate the whole tree; for this, we may use the plotting functionality

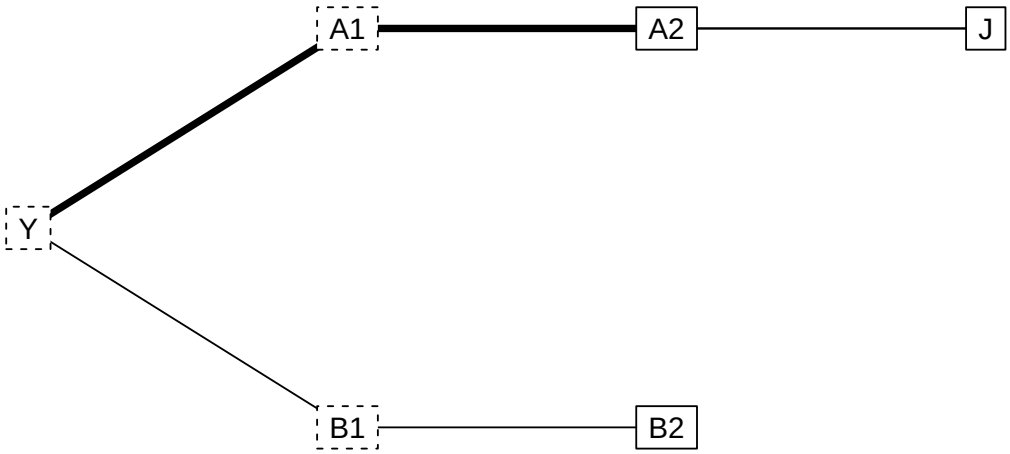
```
> plot(vistla_result)
```



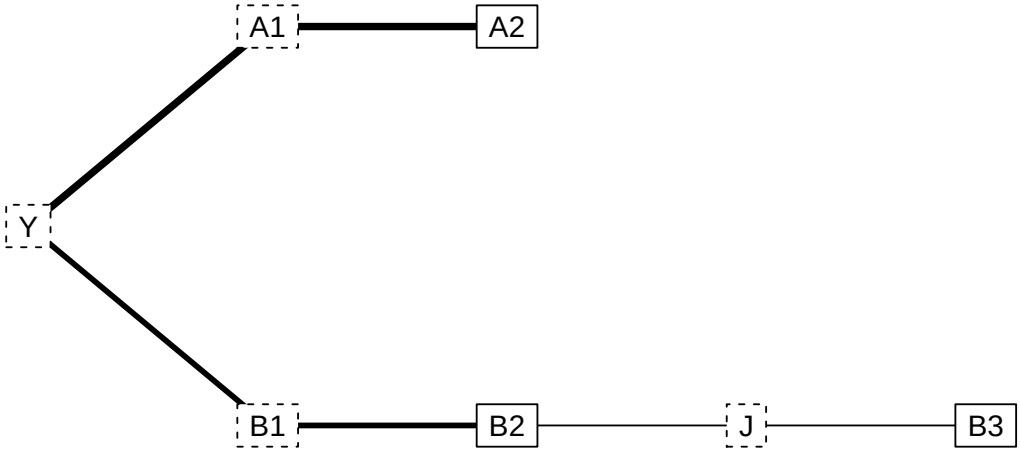
We can see that the algorithm returned the simulated topology exactly, including the bi-modality of J. The dashed boxes denote the *relays*, i.e, features that are not the terminal of the optimal path leading to it. In our example, the optimal path to J leads through A_2, but J is independently a part of the optimal path to B_3 that leads from B_2. A_1 and A_2 are marked as relays because they are directly connected to the anchor and Vistla cannot identify paths shorter than 3 features.

In a practical example, the tree may be too complex to show in full; to this end, we can use the `prune` function to reduce the tree to an interesting part. Pruning supports two filters; one is a score cut-off, and the other is a limitation of *targets*, a set of features paths to which are included. Let us investigate both of these options

```
> #Score cut-off
> plot(prune(vistla_result,score=.45))
```




```
> #Target list limit
> plot(prune(vistla_result,targets=c("A2","B3")))
```



Narrowed target list can be also passed to the `targets` argument of the `vistla` function, similarly score threshold to the `iomin` argument. The end effect is going to be the same as after using `prune`, but it may allow the algorithm to stop earlier and hence reduce the computational time. The gain depends on the data set, and may not be substantial.

Finally, one can extract a particular path with the `path_to` function; it yields the path as a simple text vector or a data frame with scores if requested with the `detailed` argument.

```
> path_to(vistla_result,"B3")

[1] "B3" "J"  "B2" "B1" "Y"

> #Also report scores
> path_to(vistla_result,"B3",detailed=TRUE)

   a  b  c     score
1 B2  J B3 0.3225647
2 B1 B2  J 0.3225647
3  Y B1 B2 0.4592551
```

The whole tree can be exported to the external analysis via `hierarchy` function, returning it in a data frame form, or via `write.dot` function, which generates a Graphviz file that can be imported with most software dedicated to graph analysis and visualisation.

3 Further steps

3.1 Numerical data

To directly analyse numerical data, one may use the Kendall transformation [2] (KT) instead of the default ML estimator. It is a parameter-less method, for ordinal data it is equivalent to Kendall correlation, while for ordinal-binary pairs to Mann-Whitney-Wilcoxon test statistic U or AUROC.

We can demonstrate this on a continuous version of the *chain* dataset, also bundled with the package. It can be loaded with

```
> data(cchain)
```

This dataset was generated from a following causal structure

$$Y \to M_1 \to M_2 \to M_3 \to M_4 \to T,$$

and so consists of 6 continuous features, each with 20 observations.

To invoke Vistla with a different estimator, one needs to use the `estimator` argument

```
> vistla(Y~.,data=cchain,estimator="kt")->vistla_kt
```

Otherwise, the process and the structure of the result is identical. We can now plot the result

```
> plot(vistla_kt)
```

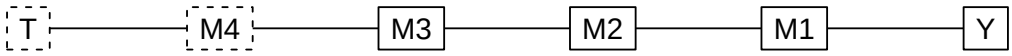
In this case, the result also exactly recreates the simulated network. More portable KT implementation is available in the `praznik` package [1].

KT does not properly estimate information of highly non-monotonic relationships, though; if they are of interest, one should consider some discretisation approach.

3.2 Alternative flows

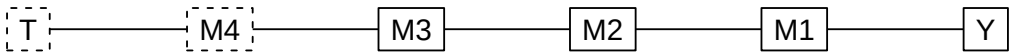
The criterion Vistla is optimising is to some extent customisable via the `flow` argument of the Vistla function; the default value is `"fromdown"`, which polarises the constraints in the outward direction from the anchor. The corresponding reversed option is `"intoup"`, which can be used to trace how information from within the system coalesces into some measured external effect represented by the anchor. Let us try this on `chain`:

```
> vistla(T~.,data=chain,flow="intoup")->vistla_rev
> plot(vistla_rev)
```



One should note that establishing the causal direction without temporal resolution is basically impossible with only data analysis; some external knowledge is essential. In fact, for the chain data, the proper structure is restored regardless of the flow direction.

```
> plot(vistla(T~.,data=chain,flow="fromdown"))
```



The effects of the `flow` are going to be more pronounced for more complex data, though. You can find more information about alternative flows in the package manual (`?flow`).

3.3 Ensemble of vistlas

Let us modify the `chain` data by adding an exact duplicate of the M_3 feature, M_3'

```
> chain_d<-chain
> chain_d$M3p<-chain$M3
```

In the modified set, paths $Y \to \ldots \to M_3 \to M_4 \to T$ and $Y \to \ldots \to M_3' \to M_4 \to T$ have exactly the score, therefore the result of the optimisation is now going to be ambivalent. In such a case, Vistla follows the randomly chosen path and each realisation of the algorithm may yield a different path to T

```
> set.seed(1)
> path_to(vistla(Y~.,data=chain_d),"T")

[1] "T"  "M4" "M3" "M2" "M1" "Y"

> set.seed(7)
> path_to(vistla(Y~.,data=chain_d),"T")

[1] "T"   "M4"  "M3p" "M2"  "M1"  "Y"
```

Such a situation can be manually resolved by re-generating the tree multiple times and looking at the consensus; this functionality is already implemented in the package, however, and it is more efficient and makes better use of parallel execution through OpenMP. It can be accessed using the `ensemble` argument of the `vistla` function. Setting this argument this way simply replicates Vistla given number of times

```
> vistla(Y~.,data=chain_d,ensemble=ensemble(100,resample=FALSE))->vistla_ens
```

The output of the function changes, though; it yields a hierarchy object and scores are replaced with counts of in how many iterations the given path appeared

```
> print(vistla_ens)

        Vistla hierarchy

+Y
| +M1
| | +M2 (100)
| | | +M3 (100)
| | | | +M4 (52)
| | | | | +T (52)
| | | +M3p (100)
| | | | +M4 (48)
| | | | | +T (48)
```

As we expected, each path was taken in roughly 50% of realisations.

Ensembling can also be used to bootstrap the information estimation error; in particular, this is the default behaviour of the `ensemble` constructor. Bootstrapping usually creates a lot of rare paths, hence it is handy to limit the tree with score thresholding and pruning. For small sets, full resample may be too harsh and destabilise the method beyond the point of usefulness; in this case random sub-sampling of the input can be used, by setting the `resample` argument of `ensemble` to a number interpreted as a fraction of observations to use in each realisation.

Refer to the package manual (`?ensemble`) for more information.

References

[1] Miron Bartosz Kursa. Praznik: High performance information-based feature selection. *SoftwareX*, 16:100819, December 2021.

[2] Miron Bartosz Kursa. Kendall transformation brings a robust categorical representation of ordinal data. *Scientific Reports*, 12(8341), May 2022.

[3] Miron Bartosz Kursa. Vistla: identifying influence paths with information theory. *Bioinformatics*, btaf036, January 2025.

Have fun!