Journal of Statistical Software

April 2022, Volume 102, Issue 6. doi: 10.18637/jss.v102.i06

The R Package stagedtrees for Structural Learning of Stratified Staged Trees

Federico Carli
Università degli Studi
di Genova

Manuele Leonelli
IE University

Eva Riccomagno
Università degli Studi
di Genova

Gherardo Varando
Universitat de València

Abstract

stagedtrees is an R package which includes several algorithms for learning the structure of staged trees and chain event graphs from data. Score-based and clustering-based algorithms are implemented, as well as various functionalities to plot the models and perform inference. The capabilities of **stagedtrees** are illustrated using mainly two datasets both included in the package or bundled in R.

Keywords: chain event graphs, graphical models, R, staged trees, structure learning algorithms.

1. Introduction

In the past twenty years there has been an explosion in the use of graphical models to represent the relationships between a vector of random variables and perform distributed inference which takes advantage of the underlying graphical representations. Bayesian networks (BNs, Darwiche 2009; Fenton and Neil 2012) are nowadays the most used graphical models, with applications to a wide array of domains and implementation in various software: for instance, the R packages (R Core Team 2022) **bnlearn** by Scutari (2010, 2017) and **gRain** by Højsgaard (2012), among others.

However, BNs can only represent symmetric conditional independences which in practical applications may not be fully justified. For this reason, a variety of models that can take into account the asymmetric nature of real-world data have been proposed; for example,

context-specific BNs (Boutilier, Friedman, Goldszmidt, and Koller 1996), labeled directed acyclic graphs (Pensar, Nyman, Koski, and Corander 2015) and probabilistic decision graphs (Jaeger, Nielsen, and Silander 2006). Unlike most of its competitors, the chain event graph (CEG, Collazo, Görgen, and Smith 2018; Smith and Anderson 2008; Riccomagno and Smith 2004, 2009) can capture all (context-specific) conditional independences in a unique graph, obtained by a coalescence over the vertices of an appropriately constructed probability tree, called staged tree.

CEGs have been used for cohort studies (Barclay, Hutton, and Smith 2013), causal analysis (Thwaites, Smith, and Riccomagno 2010; Thwaites 2013) and case-control studies (Keeble, Thwaites, Barber, Law, and Baxter 2017a; Keeble, Thwaites, Baxter, Barber, Parslow, and Law 2017b). Structure learning algorithms have been defined in the literature (Barclay, Hutton, and Smith 2014; Collazo and Smith 2016; Cowell and Smith 2014; Silander and Leong 2013). The user's toolbox to efficiently and effectively perform uncertainty reasoning with CEGs further includes methods for inference and probability propagation (Görgen, Leonelli, and Smith 2015; Thwaites, Smith, and Cowell 2008), the exploration of equivalence classes (Görgen and Smith 2018; Görgen, Bigatti, Riccomagno, and Smith 2018), causal discovery (Leonelli and Varando 2021) and robustness studies (Leonelli 2019; Wilkerson and Smith 2019). The model class of CEGs and staged trees have been further extended to model dynamic problems with recursively updated probabilities (Barclay, Collazo, Smith, Thwaites, and Nicholson 2015; Freeman and Smith 2011b), decision problems under the framework expected utility maximization (Thwaites and Smith 2017) and Bayesian games (Thwaites and Smith 2018).

The R package **stagedtrees** implements some algorithms for learning staged trees and CEGs from data and is freely available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=stagedtrees`. The package also provides inferential and visualization functions for such models as well as descriptive and summary statistics about the graph structure. The only other software available to learn such models is the **ceg** package (Collazo and Taranti 2017), including one learning algorithm (*Agglomerative Hierarchical Clustering*, Freeman and Smith 2011a).

2. Staged trees and chain event graphs

Many statistical graphical models represent a random vector of interest in terms of undirected or directed acyclic graphs. In particular, BNs are directed acyclic graphs where each vertex corresponds to a random variable and a missing edge between two nodes represents conditional independence. Conversely, staged trees are directed trees equipped with probabilites where atomic events coincide with root-to-leaf paths.

A directed tree $\mathcal{T} = (V, E)$ is a tree with vertex set V and edge set E, where each vertex except for the root has one parent only, all non-leaf vertices have at least two children and all edges point away from the root. For $v, v' \in V$ let $e = (v, v') \in E$ be the edge pointing from v to v'. For a non-leaf v, let $E(v) = \{v' \in V : (v, v') \in E\}$ and call $\mathcal{F}(v) = (v, E(v))$ a floret of the tree. Let Θ be a non-empty set of labels and $\theta : E \to \Theta$ be a function such that for any non-leaf $v \in V$ the labels in $\theta(E(v))$ are all distinct. The set $\theta(E(v))$ is denoted by $\boldsymbol{\theta}_v$ and is called the set of floret labels. Next assume $\Theta \subseteq [0, 1]$. If $\sum_{e \in E(v)} \theta(e) = 1$ for all non-leaf v, then $\mathcal{T}$ together with the $\boldsymbol{\theta}_v$'s is called a *probability tree* and $\theta(e)$ is the probability

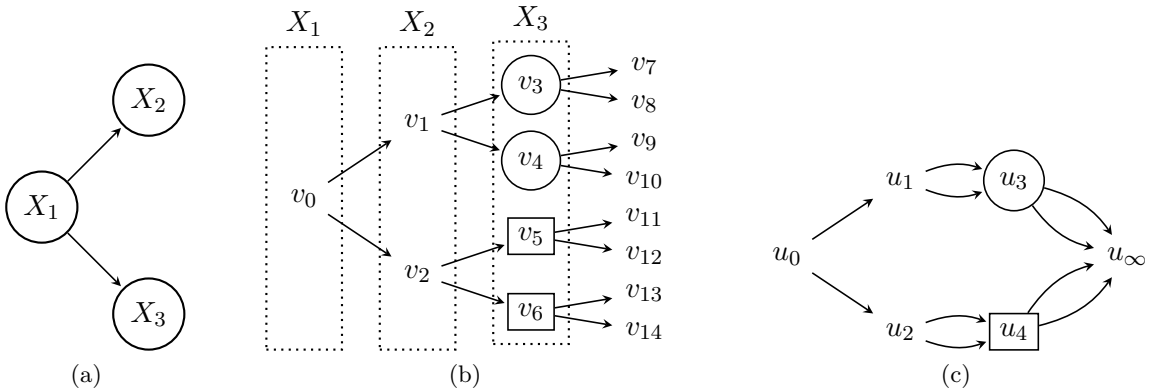

Figure 1: Illustration of the construction of staged tree and CEG from a BN for three binary random variables. The BN in Figure 1a is represented by the $\boldsymbol{X}$-compatible tree in Figure 1b where the edges emanating from v_0 represent the outcomes of X_1; the edges emanating from v_1 and v_2 represent the outcomes of X_2 conditionally on the outcome of X_1; the edges emanating from $v_3, \ldots, v_6$ represent the outcomes of X_3 conditionally on X_1 and X_2. The conditional independence of the BN coincides with the staging $\{v_3, v_4\}$ and $\{v_5, v_6\}$ (vertices not framed are in their own stage). The staged tree in Figure 1b is transformed into the CEG in Figure 1c using the positions $u_0 = \{v_0\}$, $u_1 = \{v_1\}$, $u_2 = \{v_2\}$, $u_3 = \{v_3, v_4\}$, $u_4 = \{v_5, v_6\}$ and $u_\infty = \{v_7, \ldots, v_{14}\}$.

of the edge $e \in E$. Each root-to-leaf path λ in $\mathcal{T}$, equivalently each leaf vertex, is associated to an atom in a discrete probability space and the atomic probabilities can be defined as $\prod_{e \in \lambda} \theta(e)$. Throughout, edges on a root-to-leaf path λ are ordered from the closest to the root to the closest to the leaf. The atomic probabilities together with Θ give the statistical model associated to the tree.

Definition 1 *A probability tree where for some $v, v' \in V$ $\boldsymbol{\theta}_v = \boldsymbol{\theta}_{v'}$, is called a* staged tree. *The vertices v and v' are said to be in the same stage.*

Although not strictly required, a probability tree can represent the joint probability distribution of a discrete random vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ taking values in a product space $\mathbb{X} = \times_{i=1}^{n} \mathbb{X}_i$, where $\mathbb{X}_i$ is the finite sample space of X_i, $i = 1, \ldots, n$.

Recall that for $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{X}$ the joint probability can be factorized according to the chain rule of probabilities

$$p(\boldsymbol{x}) = \prod_{i=2}^{n} p(x_i | \boldsymbol{x}^{i-1}) \, p(x_1), \tag{1}$$

where $\boldsymbol{x}^{i-1} = (x_1, \ldots, x_{i-1}) \in \times_{j=1}^{i-1} \mathbb{X}_j$. This sequential factorization can be represented by a probability tree as the one in Figure 1b where the probabilities on the right-hand-side of Equation 1 are associated to the edges emanating from the non-leaf vertices.

Definition 2 *A probability tree $\mathcal{T}$ is called $\boldsymbol{X}$-compatible if for each $\boldsymbol{x} \in \mathbb{X}$ there exists a unique root-to-leaf path $\lambda = (e_1, \ldots, e_n)$ such that $\theta(e_1) = p(x_1)$ and $\theta(e_i) = p(x_i | \boldsymbol{x}^{i-1})$ for $i = 2, \ldots, n$.*

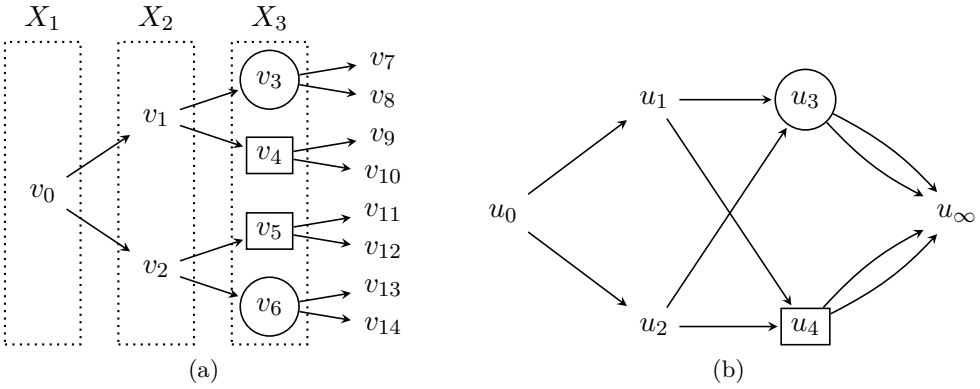

Figure 2: Staged tree and CEG for three binary random variables with stages $\{v_0\}, \{v_1\}, \{v_2\}, \{v_3, v_6\}, \{v_4, v_5\}$ and positions $u_0 = \{v_0\}$, $u_1 = \{v_1\}$, $u_2 = \{v_2\}$, $u_3 = \{v_3, v_6\}$, $u_4 = \{v_4, v_5\}$ and $u_\infty = \{v_7, \ldots, v_{14}\}$.

An $\boldsymbol{X}$-compatible tree has as many leaves as elements in $\mathbb{X}$. All vertices such that the length of the path from the root to them is i are associated to the same random variable X_{i+1}, $i = 1, \ldots, n-1$, and are said to be in the same *stratum*.

Conditional independence statements embedded in BNs then correspond to equalities between probabilities on the right-hand-side of Equation 1. This can be captured in probability trees by identifying some of the floret probability values.

For example, the BN in Figure 1a implies that X_3 is conditionally independent of X_2 given X_1, $p(x_3|x_2, x_1) = p(x_3|x_1)$ for all $x_i \in \mathbb{X}_i$, $i = 1, 2, 3$. The same conditional independence is embedded in the staged tree in Figure 1b by the staging $\{v_3, v_4\}$ and $\{v_5, v_6\}$ so that $\boldsymbol{\theta}_{v_3} = \boldsymbol{\theta}_{v_4}$ and $\boldsymbol{\theta}_{v_5} = \boldsymbol{\theta}_{v_6}$. By construction, all BNs have a staged tree representation such that vertices in the same stage must be in the same stratum as in Figure 1. Only staged trees with this property are implemented in the **stagedtrees** package.

Definition 3 *An $\boldsymbol{X}$-compatible staged tree is called stratified if all non-leaf vertices in the same stage are in the same stratum.*

The class of stratified staged trees is much larger than the one of BNs over the same variables: for instance, the staged tree with staging $\{v_3, v_6\}$ and $\{v_4, v_5\}$ in Figure 2a does not have a BN representation over the same X variables. In stratified staged trees the root vertex forms a stage by its own.

Staged trees are very expressive and flexible but, as the number of variables increases, they cannot succinctly visualize their staging. For this reason, Smith and Anderson (2008) devised a coalescence of the tree by merging some of its vertices in the same stage and therefore reducing the size of the graphical representation. The resulting graph is called a CEG, which represents the exact same probability model as the original staged tree (Collazo *et al.* 2018). The construction of a CEG from a staged tree is illustrated next.

Given a probability tree $\mathcal{T}$, a subtree $\mathcal{T}(v)$ rooted at $v \in V$ is the tree with v-to-leaf paths of $\mathcal{T}$ and the same edge probabilities. Two vertices v, v' in the same stage are said to be in the same position if the subtrees $\mathcal{T}(v)$ and $\mathcal{T}(v')$ are equal. For instance, the vertices v_3 and v_4 in Figure 1b are in the same stage but also in the same position. Therefore, for vertices in the

same position the full downstream stage structure is identical, and not only the immediate floret probabilities. Positions give a coarser partition U of the vertex set of a staged tree than stages do. Hereby, all leaves are trivially in the same position denoted by u_∞.

The CEG is the graph obtained from a staged tree $\mathcal{T} = (V, E)$ having a vertex for each set in U and edge set F so constructed: if there exist edges $e = (v, v')$, $e' = (w, w') \in E$ and v, w are in the same position then there exist corresponding edges $f, f' \in F$. If also v', w' are in the same position then the labels associated to f and f' are equal and are probabilities inherited from $\mathcal{T}$. The process of constructing a CEG is illustrated in Figure 1.

3. Package implementation

3.1. Creating staged trees and CEGs

The main object class implemented in the **stagetrees** package is `sevt` representing a staged tree model. Given a dataset, either in `data.frame` or `table` format, a staged tree which is compatible with the variables in the dataset can be constructed using the functions `full` or `indep`. The function `full` returns a `sevt` object which defines in R a staged tree where each vertex is in a different stage. It corresponds to the saturated statistical model, where the number of free parameters equals the number of edges minus the number of non leaf vertices, equivalently the number of leaves minus one. Conversely, `indep` returns a tree where all vertices in the same stratum are in the same stage, corresponding to a model where all variables are marginally independent of each other.

Worth-mentioning arguments of these two functions are: `order`, which selects the order of the variables in the tree; `join_unobserved` which collapses parts of the tree where no observations are collected (by default set to `TRUE`); `lambda`, which implements a Laplace smoothing (Russell and Norvig 2016) to address possible zero counts, especially if `join_unobserved` is set to `FALSE`.

Furthermore, a `bn.fit` (or `bn`) object created with the **bnlearn** package could be turned into a `sevt` object with `as_sevt` modelling the same conditional independences. A staged tree can be converted into a CEG model using the `ceg` function. The usual `print`, `summary` and `plot` functions provide basic information, more detailed information and the graphical representation of the model, respectively.

3.2. Structure learning algorithms

stagedtrees implements a variety of structure learning algorithms. These can be grouped into two categories:

- Score-based algorithms using various heuristics to maximize a score function. The default value of `score` is the negative BIC, but any other can be defined by the user:
  - A hill-climbing score optimization implemented in `stages_hc` which, for each stratum, at each iteration searches for the vertex to move either to a different or a new stage maximizing a score until no score improvement is found.
  - A backward hill-climbing `stages_bhc` which searches the joining of two stages maximizing a score until no score improvement is found.

  - A fast backward hill-climbing `stages_fbch` which joins two stages whenever the joining improves the score until no improvement is possible.
  - A random backward hill-climbing `stages_bhcr` which at each iteration randomly selects a stratum and two stages and joins the stages if the score is increased. The procedure is repeated until the number of iterations reaches `max_iter`.

- Clustering-based algorithms, where stages are created by clustering the probability distribution of florets:

  - Backward joining of stages `stages_bj` which iteratively joins stages if the distance between their floret probabilities is less then a given threshold value (`thr`) (the distance can be chosen with the `distance` argument, the default being the symmetrized Kullback-Leibler `"kullback"`).
  - Hierarchical clustering of stages `stages_hclust` which creates a user-defined number `k` of stages in each stratum. The function inherits all arguments of the standard `hclust` function from the **stats** package.
  - Clustering of stages using the k-means algorithm `stages_kmeans`, again creating a user-defined number `k` of stages in each stratum. The function inherits all arguments of the standard `kmeans` function from the **stats** package.

The starting model of any structure learning algorithm has to be a staged tree which, for instance, may be constructed directly from a dataset using `full` or `indep`. Different structure learning algorithm can be easily combined since the starting model for any algorithm could be also an already estimated model with another structure learning algorithm. Furthermore, model search can be computed over a subset of strata specified by `scope`.

All above algorithms work with a fixed ordering of the variables, which can be set with the argument `order`. For learning a staged tree model from data with an optimal variable ordering, the function `search_best` can be used, which implements the dynamic programming algorithm of Silander and Leong (2013) and Cowell and Smith (2014). The search of the optimal order, by optimizing a model selection criterion of choice (e.g., BIC), can be coupled with any model search algorithms mentioned above, which can be set with the argument `alg`.

3.3. Querying the model

stagedtrees provides an array of functions to explore and perform inference over a learned model:

- `stndnaming` standardly renames stages. It assigns them increasing numbers from 1 to the number of different stages, for each stratum in the tree.

- `subtree` enables for the construction of a subtree having as root any vertex of the tree. This can be achieved specifying the `path` starting from the root and ending at that vertex.

- `summary` returns for each stratum all the estimated stages, the number of paths and observations starting from the root that arrives to each stage and their corresponding probability distributions.

- `compare_stages` checks if the staging structure of two staged trees with the same order of variables are equal and returns a plot where nodes in different stages are colored in red.
- `sample_from` generates observations according to the probability distributions defined by the staged tree given in input. This can be used to perform simulation studies over a learned model.
- `get_stage` retrieves the stage associated to a given `path` from the root. To be used in combination with `summary` and/or `plot` for a more helpful use.
- `get_path` gives all the paths that starting from the root arrive to a given stratum (`var`) and `stage`.
- `prob` computes the (conditional) probability (or its logarithm if `log` = TRUE) of any event of interest (`x`) and can be used to derive all atomic probabilities.
- `confint` provides confidence intervals for floret probabilities. By default, it computes the confidence intervals for the probability parameters for all stages in the staged tree given in input; confidence intervals can be computed only for a single variable by specifying it in `parm`. Five methods are available: `wald`, `waldcc`, `wilson`, `goodman` and `quesenberry-hurst` (see e.g., Möstel, Pfeuffer, and Fischer 2020).
- `lr_test` performs a likelihood ratio test between nested staged trees.

3.4. Plotting

stagedtrees contains simple plotting functions to enable a visual exploration and visualization of the generated models.

- `plot.sevt` is a dependencies-free plotting method for the `sevt` class; users can specify stage-colouring, node and edge size and labels appearance.
- `plot.ceg` is a simple plotting function for chain event graph objects (class `ceg`) using the **igraph** package.
- `barplot` automatically generates barplots to visualize the floret probabilities for each stage of a specified variable (`var`).

4. Usage of the stagedtrees package

The well-known Titanic dataset (Dawson 1995), which provides information on the fate of the Titanic passengers and available from the **datasets** package bundled in R, is used to exemplify the usage of **stagedtrees**. **stagedtrees** and its dependencies (the **graphics** and **stats** packages bundled in R) are available from CRAN, as the suggested packages **bnlearn** (Scutari 2010, 2017) and **igraph** (Csárdi and Nepusz 2006, needed only to plot CEGs).

4.1. Learning the stage structure from a dataset

The `Titanic` dataset can be loaded into a `table` of the same name with the call to `data`.

```
R> data("Titanic", package = "datasets")
R> str(Titanic)

 'table' num [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
 - attr(*, "dimnames")=List of 4
  ..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
  ..$ Sex     : chr [1:2] "Male" "Female"
  ..$ Age     : chr [1:2] "Child" "Adult"
  ..$ Survived: chr [1:2] "No" "Yes"
```

`Titanic` includes four categorical variables: `Sex`, `Age` and `Survived` are binary and `Class` has four levels. Initial staged trees where all vertices within a stratum are either in the same or in different stages can be constructed using the `indep` and `full` functions, respectively. The argument `order` can be set to choose the order of the variables in the tree. Since our aim is to assess how the probability of survival is affected by the other factors, we fix `Survived` as the last variable in the order. We refer to Section 6 for an illustration of an automatic choice of an optimal order from data.

```
R> library("stagedtrees")
R> order <- c("Class", "Sex", "Age", "Survived")
R> m.full <- full(Titanic, name_unobserved = "na", order = order)
R> m.indep <- indep(Titanic, name_unobserved = "na", order = order)
R> m.full

Staged event tree (fitted)
Class[4] -> Sex[2] -> Age[2] -> Survived[2]
'log Lik.' -5151.517 (df=30)

R> m.indep

Staged event tree (fitted)
Class[4] -> Sex[2] -> Age[2] -> Survived[2]
'log Lik.' -5773.349 (df=7)
```

The printing of `m.full` and `m.indep` gives information about the order of the variables in the tree, the value of the log-likelihood function and the number of free parameters, whilst `plot` displays the stratified staged tree with stages coloured within each stratum as shown in Figure 3. The plot of `m.full` is depicted using the `Dynamic` palette from the **colorspace** package (Zeileis, Fisher, Hornik, Ihaka, McWhite, Murrell, Stauffer, and Wilke 2020), since the default palette has only eight colors and thus stages for the last variable would be impossible to graphically distinguish.

```
R> library("colorspace")
R> plot(m.full, col = \(s) qualitative_hcl(length(s), "Dynamic"))
R> plot(m.indep)
```

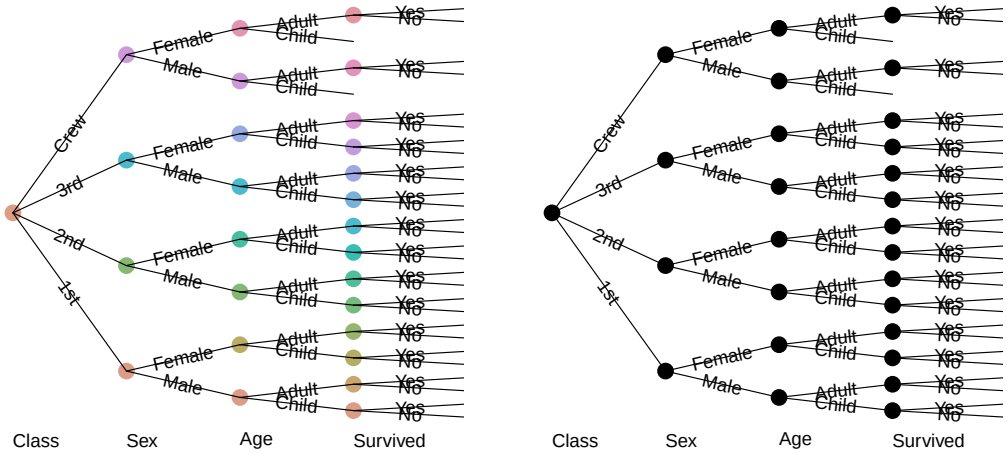

Figure 3: Left: Staged tree `m.full` where all vertices in the same stratum are in a different stage: there are 29 different stages. Colors in different strata can be equal. Right: Staged tree `m.indep` where all vertices in the same stratum are in the same stage: there are 4 different stages. The labels at the bottom denote the variable associated to a stratum.

Notice that there are no crew members, either male or female, who are children and this is correctly reflected in the trees in Figure 3 since the subtree associated to such events are collapsed (by default the argument `join_unobserved` is set to `TRUE`). The name of these collapsed vertices is set to `"na"` with the argument `name_unobserved`.[1]

Using the staged tree `m.full` or `m.indep` as starting point, structural learning algorithms can be used to infer the staging structure from the data. The hill-climbing algorithm implemented in `stages_hc` can receive in input both `m.full` and `m.indep` (since it embeds also a splitting stage move). Whilst backward algorithms (implemented in `stages_bhc`, `stages_fbhc` and `stages_bhcr`) and clustering algorithms (implemented in `stages_bj`, `stages_hclust` and `stages_kmeans`) start from the `m.full` tree. For illustration purposes, the `stages_hc` function is used with the `m.indep` tree, whilst `stages_bj` is used with `m.full`. Stages are renamed with the function `stndnaming`.

```
R> mod1 <- stndnaming(stages_hc(m.indep))
R> mod2 <- stndnaming(stages_bj(m.full, thr = 0.1))
```

The `stages_hc` function has BIC as a default score, while the default distance for `stages_bj` is the symmetrized Kullback-Leibler divergence, with threshold 0.1 in this example. The learned `mod1` and `mod2` are plotted in Figure 4 where vertices report the stage numbers. Both staged trees suggest that the variables are dependent in a non-symmetric fashion. Let's consider `mod1` for illustration. Stage 1 for the variable `Sex` suggests that the distribution of `Sex` is the same for travelers in `Class = 1st,2nd`. In the terminology of Pensar, Nyman, Lintusaari, and Corander (2016) such an equality is usually referred to as *partial independence*. Similarly, the vertices in stage 3 in the upper quarter of the variable `Age` suggest that `Age` is independent of `Sex` for `Class = crew`: this is usually called a *context-specific independence*.

[1]By default the name of the stage for unobserved situations is `"UNOBSERVED"`.

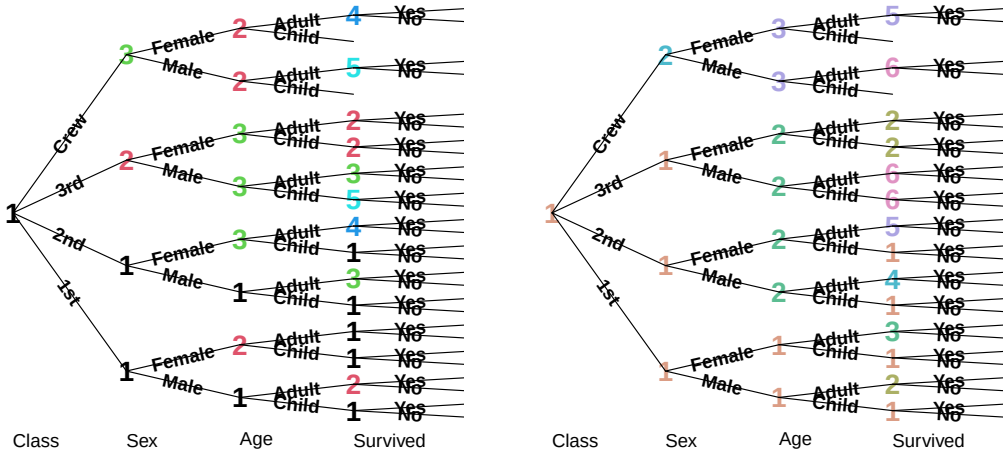

Figure 4: Staged trees `mod1` (left) and `mod2` (right) learned using the `stages_hc` and the `stages_bj` algorithms, respectively.

The stage structures of the two models are quite different and may be affected by the choice of threshold in `mod2`. However, they also share some common features: for instance, both state that the distribution of Male/Female is the same for passengers in the first and second class. Since all structural learning algorithms take as input a staged tree, it is possible to refine a learned model: for instance the model `mod2` learned using a backward algorithm may be refined using a standard hill climbing algorithm.

```
R> mod3 <- stndnaming(stages_hc(mod2))
R> plot(mod3, ignore = NULL,
+     cex_label_nodes = 1.5, cex_nodes = 0, font = 2)
```

The resulting staged tree is reported in Figure 5. For illustrative purpose we report there the full tree (by setting `ignore = NULL`). The two staged tree structures in `mod1` and `mod3` are compared through the `compare_stages` function, whose output highlights in red the nodes in different stages. Different methods can be used to compare two staged tree structures, here the `"stages"` method is used: it checks if the same exact stages are present in both models.

```
R> compare_stages(mod1, mod3, method = "stages", plot = TRUE)

[1] FALSE
```

Figure 5 shows that the two models have the same stage structure over the `Sex` and `Survived` variables, but they highly differ over `Age`. The model selection criteria `AIC` and `BIC` can be used to choose the best fitting model.

```
R> cbind(AIC(mod1, mod2, mod3), BIC = BIC(mod1, mod2, mod3)$BIC)

     df      AIC      BIC
mod1 15 10364.49 10449.94
mod2 15 10390.37 10475.82
mod3 15 10365.02 10450.47
```

Figure 5: Staged event tree `mod3` (left) and output of the `compare_stages` function between models `mod1` and `mod3` (right). Vertices depicted by a red dot in the right plot correspond to vertices for which the staging structure differs.

According to both criteria, `mod1` is the best fitting model among those tried. It is not surprising that `mod2` obtains the worst BIC scores since it was estimated with the `stages_bj` function that joins stages following a distance based heuristic and thus not the minimization of the BIC score.

4.2. Bayesian networks as staged trees

stagedtrees has the capability of translating a BN learned with the **bnlearn** package into a staged tree. To use **bnlearn** the dataset `Titanic` needs to be converted into a data frame.

```
R> titanic.df <- as.data.frame(Titanic)
R> titanic.df <- titanic.df[rep(row.names(titanic.df), titanic.df$Freq), 1:4]
```

The `hc` function of **bnlearn** can be used to learn the graph of the BN reported in Figure 6 left.

```
R> library("bnlearn")
R> mod.bn <- bnlearn::hc(titanic.df)
R> plot(mod.bn)
```

`bn.fit` returns an object of class `bn.fit` which can be turned into an object of class `sevt` using the `as_sevt` function. `sevt_fit` is used to compute also the stage probability distributions. Below the R code.

```
R> mod.bn <- bn.fit(mod.bn, titanic.df)
R> bn.tree <- sevt_fit(as_sevt(mod.bn), data = titanic.df, lambda = 0)
R> plot(bn.tree)
```

The learned BN embeds only one conditional independence statement: `Age` and `Sex` are conditionally independent given `Class` and `Survived`. This is represented in Figure 6 right by

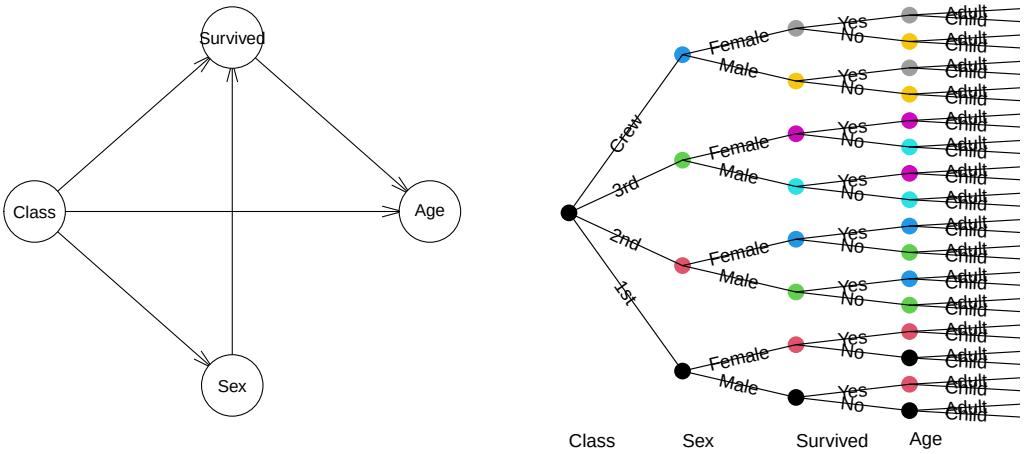

Figure 6: Left: BN model learned using the `hc` function of **bnlearn**. Right: associated staged event tree.

the highly symmetric staging structure over the variable `Age`. Notice that this tree, since it is representing the associated BN, does not collapse subtrees where there are no associated observations in the dataset. However this can be achieved by using the function `join_unobserved`. It is also worth noticing that the order of the variables chosen by **bnlearn** is different to the one used for `mod1`, `mod2` and `mod3`. Therefore, it is not possible to use `compare_stages` to compare `bn.tree` with `mod1`, `mod2` or `mod3`.

The staged tree corresponding to the associated learned BN could be used as the starting point of any of our structure learning algorithms, as below and also in Barclay *et al.* (2013). As an illustration, we use here the `stages_hclust` function specifying that in each stratum there should be 2 stages.

```
R> mod4 <- stages_hclust(bn.tree, k = 2)
R> plot(mod4, col = \(s) c("red3", "blue3"))
```

The staged tree `mod4`, which is displayed in Figure 7 left, is coalesced into the more compact CEG representation shown in Figure 7 right. This can be achieved by the `ceg` function which takes as input `mod4` and the `plot` method for `ceg` objects, which requires the suggested **igraph** package (Csárdi and Nepusz 2006).

```
R> library("igraph")
R> plot(ceg(mod4), col = \(s) c("red3", "blue3"))
```

Vertices in the last stratum are coalesced into two positions, whilst vertices in the penultimate stratum are coalesced into four positions, thus reducing the overall number of vertices of the underlying graphical representation. We refer to Section 7 for a discussion of the CEG plotting capabilities.

4.3. Querying the model

Chosen a model, the focus is on using it to perform inference and understanding the relation-

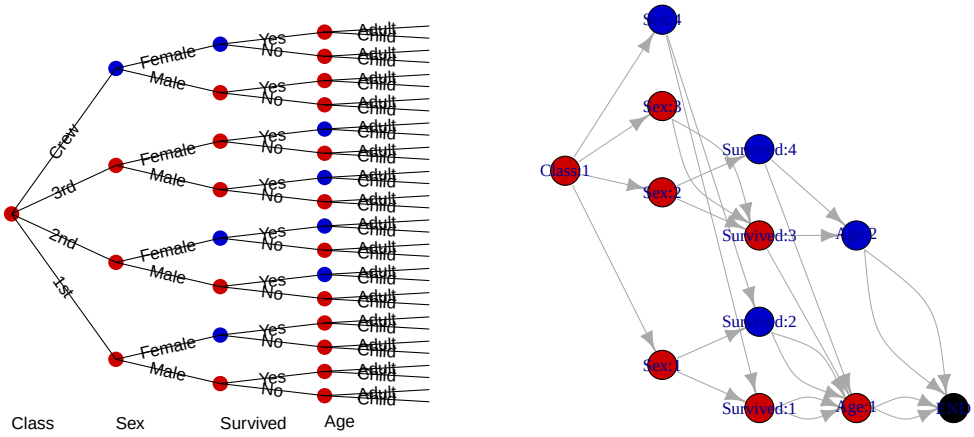

Figure 7: Staged event tree `mod4` (left) and its corresponding CEG representation (right).

ship between the problem variables. Here we choose `mod1` which was the best scoring model according to AIC and BIC.

The dataset in this simple example only includes four variables and its staged tree can be easily investigated by eye. For more complex applications the function `subtree` is useful as it enables the construction of a subtree having as root any vertex of the tree. This can be achieved specifying the path starting from the root and ending at that vertex. For instance, it is possible to construct the subtree relative to the crew of the Titanic.

```
R> subtree.crew <- subtree(mod1, c(Class = "Crew"))
R> subtree.crew

Staged event tree (fitted)
Sex[2] -> Age[2] -> Survived[2]

R> plot(subtree.crew)
```

`subtree.crew` is still formally a staged tree over three variables. The subtree is displayed in Figure 8 and its stage structure coincides with the one in the upper quarter of `mod1` reported on the left of Figure 4. The colors of the stages are different in the two plots since two different colors palettes have been used.

A detailed model summary of `mod1` can be obtained by the `summary` function.

```
R> summary(mod1)

Call:
stages_hc(m.indep)
lambda:  0
Stages:
  Variable:  Class
 stage npaths sample.size       1st       2nd       3rd      Crew
```

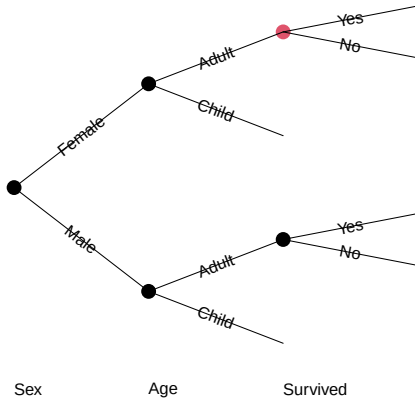

Figure 8: Subtree of the staged tree `mod1` representing `Sex`, `Age` and `Survived` of `Crew` passengers only.

```
    1      0        2201 0.1476602 0.1294866 0.3207633 0.40209
 ------------
 Variable:  Sex
stage npaths sample.size      Male    Female
    1      2         610 0.5885246 0.4114754
    2      1         706 0.7223796 0.2776204
    3      1         885 0.9740113 0.0259887
 ------------
 Variable:  Age
stage npaths sample.size        Child     Adult
    1      2         359 0.0445682451 0.9554318
    2      3        1030 0.0009708738 0.9990291
    3      3         812 0.1133004926 0.8866995
 ------------
 Variable:  Survived
stage npaths sample.size         No       Yes
    1      5         174 0.02298851 0.9770115
    2      3         371 0.60377358 0.3962264
    3      2         630 0.85873016 0.1412698
    4      2         116 0.13793103 0.8620690
    5      2         910 0.77472527 0.2252747
   na      2           0         NA        NA
 ------------
```

The output of `summary` together with the function `get_path` allows us to determine the estimated survival probabilities of the passengers of the Titanic. Stage 1 for `Survived` has the highest survival probability and it includes children from the first two classes and adult women from the first class as shown by the following code.

```
R> get_path(mod1, var = "Survived", stage = "1")
```

```
  Class    Sex   Age
1   1st   Male Child
3   1st Female Child
4   1st Female Adult
5   2nd   Male Child
7   2nd Female Child
```

Stage 3 has the lowest survival probability and includes adult males of second and third class.

```
R> get_path(mod1, var = "Survived", stage = "3")
```

```
   Class  Sex   Age
6    2nd Male Adult
10   3rd Male Adult
```

Package **stagedtrees** also includes the function `get_stage` to get the stage associated to a given path.

```
R> get_stage(mod1, path = c("Crew", "Female"))
```

```
[1] "2"
```

The function `prob` allows for the computation of the probability of any event of interest.

```
R> prob(mod1, c(Survived = "Yes"))
```

```
[1] 0.3236376
```

```
R> prob(mod1, c(Survived = "Yes"), conditional_on = c(Age = "Adult"))
```

```
[1] 0.3165252
```

```
R> prob(mod1, c(Survived = "Yes"), conditional_on = c(Age = "Child"))
```

```
[1] 0.4584954
```

For instance, the probability of survival of any passenger is 0.3236, but this decreases to 0.3165 or increases to 0.4585 given that the passenger was an adult or a child, respectively.

Similarly we can compute the conditional probability of survival of a male child traveling in first class.

```
R> cond <- c(Age = "Child", Class = "1st", Sex = "Male")
R> prob(mod1, c(Survived = "Yes"), conditional_on = cond)
```

```
[1] 0.9770115
```

This is exactly the same probability shown with the `summary` function for vertices in stage 1 of the variable `Survived`. This probability is also estimated to be the same for the conditioning events shown above with the `get_path` function.

All atomic probabilities related to the leaves of the staged tree can be obtained as follows:

```
R> obs <- expand.grid(mod1$tree[4:1])[, 4:1]
R> cbind(obs, p = round(prob(mod1, obs), 6))
```

```
   Class    Sex   Age Survived        p
1    1st   Male Child       No 0.000089
2    1st   Male Child      Yes 0.003784
3    1st   Male Adult       No 0.050130
4    1st   Male Adult      Yes 0.032898
5    1st Female Child       No 0.000001
6    1st Female Child      Yes 0.000058
7    1st Female Adult       No 0.001395
8    1st Female Adult      Yes 0.059304
9    2nd   Male Child       No 0.000078
10   2nd   Male Child      Yes 0.003318
11   2nd   Male Adult       No 0.062524
12   2nd   Male Adult      Yes 0.010286
13   2nd Female Child       No 0.000139
14   2nd Female Child      Yes 0.005898
15   2nd Female Adult       No 0.006516
16   2nd Female Adult      Yes 0.040727
17   3rd   Male Child       No 0.020339
18   3rd   Male Child      Yes 0.005914
19   3rd   Male Adult       No 0.176434
20   3rd   Male Adult      Yes 0.029025
21   3rd Female Child       No 0.006092
22   3rd Female Child      Yes 0.003998
23   3rd Female Adult       No 0.047675
24   3rd Female Adult      Yes 0.031286
25  Crew   Male Child       No 0.000000
26  Crew   Male Child      Yes 0.000000
27  Crew   Male Adult       No 0.303119
28  Crew   Male Adult      Yes 0.088141
29  Crew Female Child       No 0.000000
30  Crew Female Child      Yes 0.000000
31  Crew Female Adult       No 0.001440
32  Crew Female Adult      Yes 0.009000
```

It shows that around 30% of the observations follows the root-to-leaf path `Crew`, `Male`, `Adult`, `No`.

The function `confint` can be used to output confidence intervals for the staged tree parameters. For instance, confidence intervals for stages related to `Age` with the `goodman` method can be computed as:

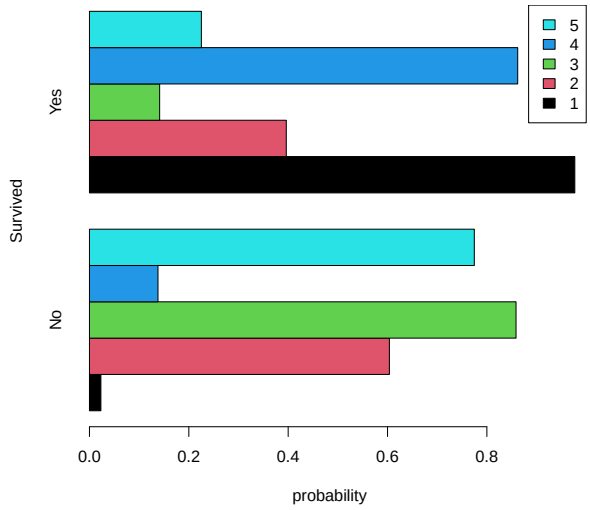

Figure 9: Output of the `barplot` function for the variable `Survived` according to the stage structure of `mod3` depicted in Figure 5 left.

```
R> confint(mod1, "Age", method = "goodman")

                    2.5 %      97.5 %
Age=Child|1 0.0258101180 0.075897181
Age=Adult|1 0.9241028192 0.974189882
Age=Child|2 0.0001411609 0.006645046
Age=Adult|2 0.9933549540 0.999858839
Age=Child|3 0.0907102352 0.140646387
Age=Adult|3 0.8593536135 0.909289765
```

There are two difficulties in estimating confidence intervals on staged trees: first, the presence of very small sample sizes for some vertices of the tree; second, the presence of highly unbalanced, almost degenerate, distributions, which may be caused by small sample sizes especially for vertices close to the leaves of the tree. Confidence intervals that have been estimated in one of these two scenarios may be therefore biased (for details, see Glaz and Sison 1999; Möstel *et al.* 2020).

Finally, barplots can be created to give a visual representation of the estimated probabilities associated to a stratum of the tree as reported in Figure 9.

```
R> barplot(mod3, "Survived", legend.text = TRUE, horiz = TRUE,
+    args.legend = list(x = 1), ylab = "Survived")
```

5. A comparison analysis

A comparison analysis of structural learning algorithms implemented in **stagedtrees** is performed on ten datasets, chosen mostly from the literature on CEGs and probabilistic graphical models for contingency tables. The main features of the datasets are summarized in Table 1, which for each dataset gives the number of observations, variables, root-to-leaf path, cells with

Dataset	# obs.	# variables	# root-to-leaf paths λ	# non-leaf nodes	# 0 cells	# edges
`Asym`	1000	4	16	15	1	30
`chestSim500`	500	8	256	255	182	510
`FallEld`	50000	4	64	27	0	90
`monks1`	432	7	864	603	243	1466
`PhDArticles`	915	6	144	136	0	279
`Pokemon`	999	5	32	31	0	62
`puffin`	69	6	768	343	284	1110
`reinis`	1841	6	64	63	0	126
`selfy`	2804	4	72	34	4	105
`Titanic`	2201	4	32	27	0	58

Table 1: Summary information about the ten datasets considered for the comparison analysis in Section 5.

Dataset	References	R package
`Asym`	Simulated dataset	**stagedtrees**
`chestSim500`	Højsgaard, Edwards, and Lauritzen (2012)	**gRbase**
`FallEld`	Shenvi, Smith, Walton, and Eldridge (2019)	
`monks1`	Michalski and Wnek (1993)	
`PhDArticles`	Long (1990)	**stagedtrees**
`Pokemon`	Gabbiadini, Sagioglou, and Greitemeyer (2018)	**stagedtrees**
`puffin`	Bouveyron, Celeux, Murphy, and Raftery (2019)	**MBCbook**
`reinis`	Højsgaard *et al.* (2012)	**gRbase**
`selfy`	Dalla Zuanna, Caltabiano, Minello, and Vignoli (2020)	
`Titanic`	Dawson (1995)	**datasets**

Table 2: Main references and R packages related to the analyzed datasets.

zero counts (either observed or structural), non-leaf nodes and edges in the staged tree. The datasets are available from the **stagedtrees**, **datasets** and **gRbase** (Dethlefsen and Højsgaard 2005) R packages. It is not the purpose of this section to show how to model these datasets. For this we refer to Section 6 and to the main references for each dataset reported in Table 2.

A short simulation study over these ten datasets is conducted. Twelve algorithms from the **stagedtrees** package are run on each dataset (all score-based algorithms use BIC as `score`). Seven additional models from the literature are estimated, namely BNs using hill-climbing and tabu search (in **bnlearn**), naive Bayes classifiers (in **e1071** Meyer, Dimitriadou, Hornik, Weingessel, and Leisch 2021), Logistic regression and neural networks with 10 units in the hidden layer and weight decay equal to 0.001 (in **nnet** Venables and Ripley 2002), classification trees and random forests with 200 trees and three variables randomly sampled as candidates at each split (in **rpart**, Therneau and Atkinson 2022, and **randomForest**, Liaw and Wiener 2002, respectively). See Table 3 for details.

For each dataset, each algorithm is run 10 times on 80% of the data randomly selected and the estimated model is tested on the remaining 20% of the dataset. The average of all the investigated quantities over the 10 runs is then computed. We compute the number of degrees of freedom, log-likelihood, AIC and BIC values, classification accuracy (the classification variable is the one in the first stratum of the staged tree) and computational cost of models estimated with 12 algorithms in **stagedtrees**.

Source	Name	Function (R package)
stagedtrees	Independent	`indep`
	Full	`full`
	HC – Independent	`stages_hc`
	HC – Full	`stages_hc`
	BHC	`stages_bhc`
	Fast BHC	`stages_fbhc`
	Random BHC	`stages_bhcr`
	Kullback-Leibler	`stages_bj`,
	Refined BN	`stages_bhc(as_sevt(bn.fit( )))`
	Hclust	`stages_hclust`
Literature	Bnlearn hill-climbing	`hc` (**bnlearn**)
	Bnlearn tabu	`tabu` (**bnlearn**)
	Naive Bayes classifier	`naiveBayes` (**e1071**)
	Logistic model	`multinom` (**nnet**)
	Neural network	`nnet` (**nnet**)
	Classification tree	`rpart` (**rpart**)
	Random forest	`randomForest` (**randomForest**)

Table 3: List of the algorithms from the R package **stagedtrees** and from the literature used for model estimation on the ten datasets in Table 2. In round brackets the corresponding R packages are presented.

For ease of exposition, we report here in Table 4 the results over the `selfy` dataset for the 12 algorithms from **stagedtree**, although similar conclusions could be drawn from any other dataset. For all datasets we report in Table 5 the mean accuracies of the algorithms from the literature as well as the mean accuracy of the staged tree learnt with `stages_bhc`, as a representative from the **stagedtrees** package. The following general conclusions can be made based on the results reported in these tables:

- Full and independent are the starting models in order to compare the performances of all the structural learning algorithms implemented. The first fits a full-dependence structure to the dataset, by providing one of the best results according to the log-likelihood, due to the over-fitting introduced. The independent model fits a full-independence structure to the dataset, estimating always the smallest log-likelihood, due to its under-fitting.

- The number of estimated parameters (df) is highly variable, according to the criterion and the starting stage structure (dependence or independence model). As expected, for backward algorithms with joining based on the Kullback-Leibler distance, the higher is the threshold below which the distance between the transition distributions of two stages are set to be equal, the lower will be the number of estimated parameters.

- Most often, the higher the number of degrees of freedom a model has, the higher will be the corresponding log-likelihood value.

- The minimum values of the AIC and BIC indices are attained with hill-climbing algorithms. This is intuitive, because the implemented score-based algorithms have as

Algorithm	df	logLik	AIC	BIC	Accuracy	Comp.time (sec.)
Independent	10.00	−7892.51	15805.03	15862.19	0.7554	0.2396
Full	64.60	−6251.36	12631.92	13001.18	0.8495	0.2459
HC – Indep.	31.00	−6277.94	12617.89	12795.08	0.8489	1.1130
HC – Full	35.20	−6264.32	12599.04	12800.25	0.8507	3.5747
BHC	32.60	−6271.70	12608.60	12794.94	0.8491	0.3916
Fast BHC	31.00	−6301.09	12664.18	12841.38	0.8495	0.2466
Random BHC	37.60	−6284.50	12644.21	12859.13	0.8480	0.2476
Kullback-Leibler – 0.01	60.40	−6250.38	12621.56	12966.80	0.8496	0.2331
Kullback-Leibler – 0.05	50.20	−6250.88	12602.17	12889.11	0.8502	0.2371
Kullback-Leibler – 0.20	38.00	−6262.82	12601.65	12818.86	0.8504	0.2612
Refined BN	28.60	−6286.85	12630.89	12794.37	0.8479	0.3627
Hclust $k = 2$	16.00	−6724.99	13481.97	13573.43	0.8041	0.2597

Table 4: Mean results for **stagedtrees** algorithms over 10 replications based on the random selection of 80% of the whole `selfy` dataset for the estimation of models and the remaining part for testing them. Experiments performed on a standard laptop with 8 GB of RAM and an i5 3.1 GHz CPU.

Algorithm	Dataset				
	`Asym`	`chestSim500`	`FallEld`	`monks1`	`PhDArticles`
stagedtrees BHC	0.8490	0.8460	0.7666	0.9744	0.4164
Bnlearn hill-climbing	0.6985	0.6610	0.6942	0.4500	0.4645
Bnlearn tabu	0.6985	0.8510	0.7596	0.4500	0.4754
Logistic model	0.6400	0.8480	0.7667	0.7372	0.4836
Naive Bayes classifier	0.6815	0.8480	0.7669	0.7372	0.4672
Neural network	0.8490	0.8360	0.7668	1.0000	0.4497
Classification tree	0.8490	0.8510	0.7668	0.7605	0.4530
Random forest	0.8490	0.8490	0.7668	1.0000	0.4639
Algorithm	Dataset				
	`Pokemon`	`puffin`	`reinis`	`selfy`	`Titanic`
stagedtrees BHC	0.7246	0.9000	0.8546	0.8491	0.7934
Bnlearn hill-climbing	0.7246	0.4385	0.8562	0.7554	0.6793
Bnlearn tabu	0.7246	0.4385	0.8562	0.7804	0.7102
Logistic model	0.7246	0.9385	0.8562	0.8486	0.7795
Naive Bayes classifier	0.7246	0.9692	0.8562	0.8282	0.7752
Neural network	0.7231	0.9538	0.8543	0.8502	0.7918
Classification tree	0.7231	0.8923	0.8552	0.8516	0.7902
Random forest	0.7231	0.9615	0.8562	0.8498	0.7925

Table 5: Mean accuracies for one of the best fitting **stagedtrees** algorithm (BHC) and algorithms from the literature over 10 replications based on the random selection of 80% of the whole dataset for model estimation and the remaining 20% for testing.

optimization default the minimization of the BIC index. However, even if the distance-based algorithms do not aim at minimizing these indices, their performances according to AIC and BIC values are satisfactory and comparable with the score-based methods.

- The hill-climbing algorithms are slower than others. In particular, the hill-climbing starting from the full-dependence model (HC - Full) is the slowest, because it both joins and splits stages. Conversely, distance-based methods, fast or random backward hill-climbing and Hclust are the fastest.
- The accuracy of all models is comparable, the lowest scoring models being independent and Hclust due to their simplicity.
- The accuracy of the **stagedtrees** BHC algorithm is higher in almost all datasets than the one of Bayesian network models, thus highlighting the need for context-specific conditional independence models in real-world applications.
- The simulated `Asym` dataset is characterized by context-specific conditional independences. As expected from the theory, all proposed algorithms in **stagedtrees** give better accuracies than ones obtained with Bayesian networks.
- Overall the algorithms implemented in **stagedtrees** have competitive accuracy, although these structural learning algorithms have the aim to estimate the joint probability distribution and not the conditional one of interest as for most of the literature algorithms. More precisely, all the literature's models in Table 3, except the ones from **bnlearn**, estimate directly the conditional probability of observing the response variable, given all the other explanatory variables.

6. A dataset analysis using stagedtrees

The `data.frame` `PhDArticles` includes information regarding the number of publications of 915 PhD biochemistry students during the 1950s and 1960s (Long 1990) and it is available in the **stagedtrees** package.

We use in the following the new native pipe operator to improve readability.

```
R> data("PhDArticles", package = "stagedtrees")
R> str(PhDArticles)

'data.frame':        915 obs. of  6 variables:
 $ Articles: Factor w/ 3 levels "0","1-2",">2": 1 1 1 1 1 1 1 1 1 1 ...
 $ Gender  : Factor w/ 2 levels "male","female": 1 2 2 1 2 2 2 1 1 2 ...
 $ Kids    : Factor w/ 2 levels "yes","no": 2 2 2 1 2 1 2 1 2 2 ...
 $ Married : Factor w/ 2 levels "no","yes": 2 1 1 2 1 2 1 2 1 2 ...
 $ Mentor  : Factor w/ 3 levels "low","medium",..: 2 2 2 1 3 1 1 2 2 1 ...
 $ Prestige: Factor w/ 2 levels "low","high": 1 1 2 1 2 2 2 1 2 1 ...

R> bn <- bnlearn::hc(PhDArticles)
R> plot(bn)
```

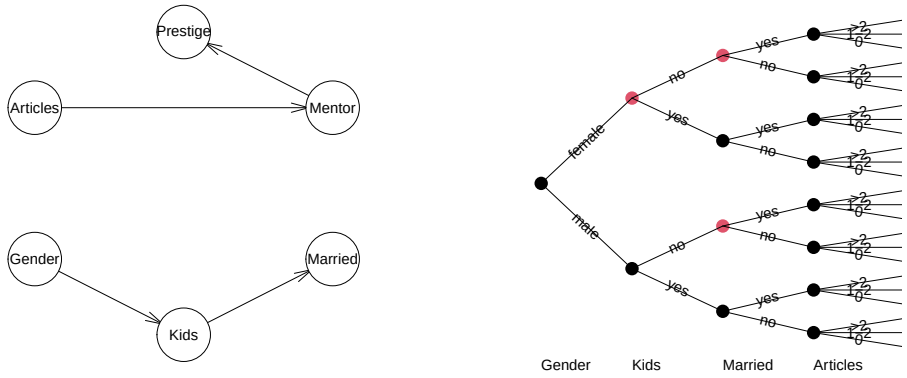

Figure 10: BN model learned over the `PhDArticles` dataset and equivalent staged tree over `Gender`, `Kids`, `Married` and `Articles`.

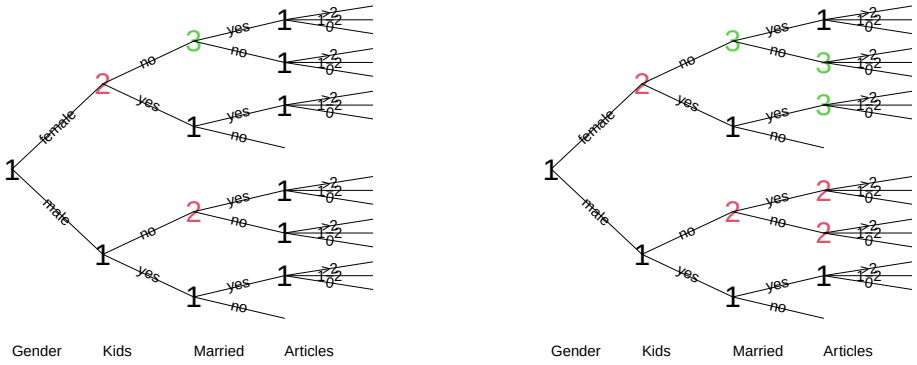

Figure 11: Staged tree models learned over the variables `Gender`, `Kids`, `Married` and `Articles` of `PhDArticles`. Left: Staged event tree `phd.mod1`. Right: Staged event tree `phd.mod2`.

```
R> order <- c("Gender", "Kids", "Married", "Articles")
R> bn.as.tree <- as_sevt(bn.fit(bn, data = PhDArticles), order = order)
R> plot(bn.as.tree)
```

The learned BN model in Figure 10 left states that the number of publications (`Articles`) is marginally independent of `Gender`, `Married` and `Kids` and states that the prestige of the University is conditionally independent of the number of publications of the student given the number of publications of the mentor. The strength of the marginal independence between `Articles` and (`Gender`, `Kids`, `Married`) is investigated. On these four variables, staged tree models starting from the independence tree (`phd.mod1`) and the full tree (`phd.mod2`) are learned using the hill-climbing algorithm and are reported in Figure 11.

```
R> phd.mod1 <- PhDArticles |> indep(order = order) |> stages_hc()
R> phd.mod2 <- PhDArticles |> full(order = order) |> stages_hc()
R> compare_stages(phd.mod1, phd.mod2, plot = TRUE, method = "stages")
```

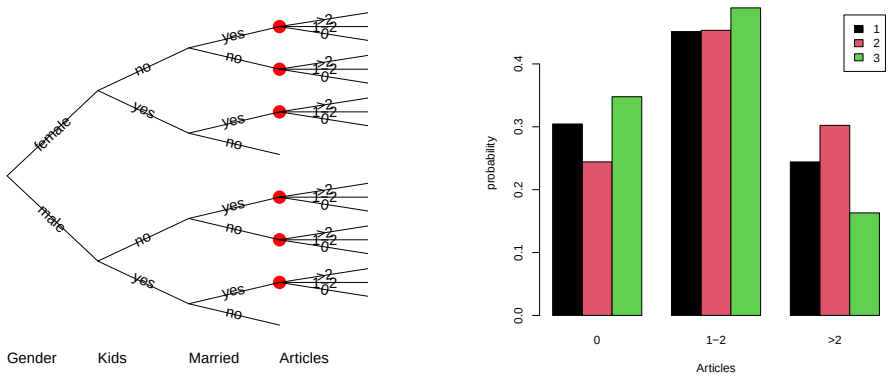

Figure 12: Left: Comparison between `phd.mod1` and `phd.mod2` over the variables `Gender`, `Kids`, `Married` and `Articles` of `PhDArticles`. Right: Conditional probability of `Articles` given `Gender`, `Kids` and `Married` for the stages in `phd.mod2`.

```
[1] FALSE
```

Investigating the estimated staging structures of the two staged trees, it is clear that for the first three variables they are exactly equal, according to the comparison depicted in Figure 12 left. Conversely, for the variable `Articles` in `phd.mod1` only one stage distribution is estimated and in `phd.mod2` three stages distributions are obtained. To further explore the different conditional probabilities associated to the stages for `Articles` in `phd.mod2`, the `barplot` function can be used (Figure 12 right).

```
R> barplot(phd.mod2, "Articles", legend.text = TRUE, xlab = "Articles")
```

From the output in Figure 12 right together with the staged tree in Figure 11 right, it can be noted that not married women without kids as well as married women with kids (stage 3) have the lowest estimated probability of a high number of articles. The population with the highest probability of a high number of publications consists of men with no kids (stage 2).

A likelihood-ratio test can be carried out with `lr_test` to test if the simpler `phd.mod1` model describes the data sufficiently well compared to the more complex `phd.mod2`. The function automatically checks if the two input models are nested.

```
R> lr_test(phd.mod1, phd.mod2)

Likelihood-ratio test

Gender[2] -> Kids[2] -> Married[2] -> Articles[3]
Model 1: phd.mod1
Model 2: phd.mod2
  #Df  LogLik Df  Chisq Pr(>Chisq)
1  10 -2555.9
2  14 -2547.4  4 16.955   0.001973 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

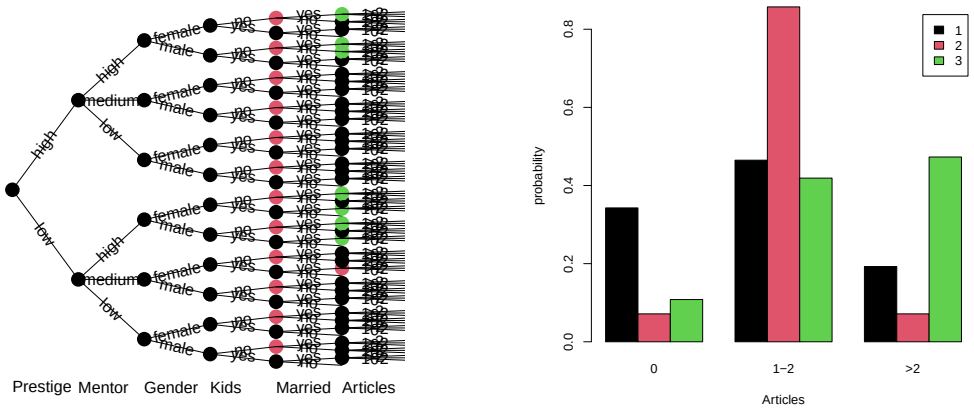

Figure 13: Staged tree `phd.all` (left) over all the variables of `PhDArticles` and corresponding estimated conditional probabilities for stages related to variable `Articles` (right).

The small p value obtained (< 0.01) confirms that the asymmetric structure described by `phd.mod2` is indeed supported by the data.

Finally, a staged tree over all the variables in `PhDArticles` is built by using the backward-joining algorithm implemented in `stages_bj`. In Figure 13 the plot of the resulting model is displayed together with the barplot associated to `Articles` conditional probabilities.

```
R> order <- c("Prestige", "Mentor", order)
R> phd.all <- PhDArticles |> full(order = order) |> stages_bj(thr = 0.5) |>
+    stndnaming()
```

The stage with highest probability of a large number of articles (stage 3) includes now the following paths:

```
R> get_path(phd.all, "Articles", "3")
```

```
   Prestige Mentor Gender Kids Married
18      low   high   male  yes     yes
20      low   high   male   no     yes
22      low   high female  yes     yes
24      low   high female   no     yes
43     high   high   male   no      no
44     high   high   male   no     yes
48     high   high female   no     yes
```

So PhD students with a high number of publications all have a mentor with a high number of publications and most of them are married and with no kids.

In all previous analyses we were interested in assessing how the number of articles were affected by the other factors. For this reason, the variable `Articles` was chosen to be the last in the order, whilst the others were arbitrarily fixed according to one of the topological orders of the learned BN. However, the function `search_best` implements the dynamic programming algorithm of Cowell and Smith (2014) to search an optimal order of the variables from data.

```
R> phd.order <- search_best(PhDArticles, alg = stages_bhc)
R> phd.order

Staged event tree (fitted)
Articles[3] -> Married[2] -> Mentor[3] -> Gender[2] -> Prestige[2] -> Kids[2]
'log Lik.' -4076.919 (df=19)
```

With respect to the BIC, this new staged tree provides a great improvement compared to `phd.all`. However, in the optimal order `Articles` is the root of the tree and therefore its staging cannot be studied to assess how it depends on the other variables.

```
R> BIC(phd.all, phd.order)

          df      BIC
phd.all   15 8504.529
phd.order 19 8283.398
```

7. Conclusions

stagedtrees is an R package which provides a freely-available implementation of staged trees and CEGs structure. Many functions are provided for the purpose of structural learning. **stagedtrees** is designed to support users in handling categorical experimental data and analyzing the learned models to untangle complex dependence structures. It provides a set of utility functions to perform exploratory data analysis and basic inference procedures.

Only structure learning algorithms for stratified staged trees are currently implemented. The difficulty with exploring the model space of non-stratified trees lies in the explosion of its size with the increase of the number of variables. Fast heuristic model search procedures are currently investigated, for instance using the maxsat approach or integer programming which have proven successful in structural learning of BNs (Bartlett and Cussens 2017; Berg, Järvisalo, and Malone 2014).

Graphical outputs from the functions' package are produced using the R package **graphics**. In addition, a simple function is provided to plot CEGs using the **igraph** package, since no theoretical studies have been carried out yet to establish how to "optimally" represent a CEG underlying graph. This line of research is currently been pursued by the authors who are planning to develop an additional R package which would provide the user with additional plotting and visualization functions.

Acknowledgments

Motivations for the implementation of the R package **stagedtrees** emerged at the *1st UK Workshop on Probabilistic Reasoning Using CEGs (Glasgow 2019)*. The participants are gratefully acknowledged. Professor Marco Scutari is thanked for helpful email exchanges. The members of the SELFY project and Professor Alessandra Minello are thanked for sharing the selfy dataset. GV was supported by a research grant (13358) from VILLUM FONDEN and by the European Research Council (ERC) Synergy Grant "Understanding and Modelling the Earth System with Machine Learning (USMILE)" under Grant Agreement No 855187.

References

Barclay L, Collazo R, Smith JQ, Thwaites PA, Nicholson A (2015). "The Dynamic Chain Event Graph." *Electronic Journal of Statistics*, **9**(2), 2130–2169. doi:10.1214/15-ejs1068.

Barclay L, Hutton J, Smith JQ (2014). "Chain Event Graphs for Informed Missingness." *Bayesian Analysis*, **9**(1), 53–76. doi:10.1214/13-ba843.

Barclay LM, Hutton JL, Smith JQ (2013). "Refining a Bayesian Network Using a Chain Event Graph." *International Journal of Approximate Reasoning*, **54**, 1300–1309. doi:10.1016/j.ijar.2013.05.006.

Bartlett M, Cussens J (2017). "Integer Linear Programming for the Bayesian Network Structure Learning Problem." *Artificial Intelligence*, **244**, 258–271. doi:10.1016/j.artint.2015.03.003.

Berg J, Järvisalo M, Malone B (2014). "Learning Optimal Bounded Treewidth Bayesian Networks via Maximum Satisfiability." In S Kaski, J Corander (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pp. 86–95. URL https://proceedings.mlr.press/v33/berg14.html.

Boutilier C, Friedman N, Goldszmidt M, Koller D (1996). "Context-Specific Independence in Bayesian Networks." In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pp. 115–123. URL https://dl.acm.org/doi/10.5555/2074284.2074298.

Bouveyron C, Celeux G, Murphy B, Raftery A (2019). *Model-Based Clustering and Classification for Data Science: With Applications in R*. Cambridge University Press. doi:10.1017/9781108644181.

Collazo R, Taranti P (2017). ***ceg**: Chain Event Graph*. R package version 0.1.0, URL https://CRAN.R-project.org/package=ceg.

Collazo RA, Görgen C, Smith JQ (2018). *Chain Event Graphs*. Chapmann & Hall. doi:10.1201/9781315120515.

Collazo RA, Smith JQ (2016). "A New Family of Non-Local Priors for Chain Event Graph Model Selection." *Bayesian Analysis*, **11**(4), 1165–1201. doi:10.1214/15-ba981.

Cowell RG, Smith JQ (2014). "Causal Discovery through MAP Selection of Stratified Chain Event Graphs." *Electronic Journal of Statistics*, **8**(1), 965–997. doi:10.1214/14-ejs917.

Csárdi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal*, **Complex Systems**(1695), 1–9.

Dalla Zuanna G, Caltabiano M, Minello A, Vignoli D (2020). "Catching Up! The Sexual Opinions and Behaviour of Italian Students (2000–2017)." *Genus*, **76**(16). doi:10.1186/s41118-020-00085-4.

Darwiche A (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press. doi:10.1017/cbo9780511811357.

Dawson RJM (1995). "The "Unusual Episode" Data Revisited." *Journal of Statistics Education*, **3**(3). doi:10.1080/10691898.1995.11910499.

Dethlefsen C, Højsgaard S (2005). "A Common Platform for Graphical Models in R: The **gRbase** Package." *Journal of Statistical Software*, **14**(17), 1–12. doi:10.18637/jss.v014.i17.

Fenton N, Neil M (2012). *Risk Assessment and Decision Analysis with Bayesian Networks*. CRC Press. doi:10.1201/b13102.

Freeman G, Smith JQ (2011a). "Bayesian MAP Model Selection of Chain Event Graphs." *Journal of Multivariate Analysis*, **102**(7), 1152–1165. doi:10.1016/j.jmva.2011.03.008.

Freeman G, Smith JQ (2011b). "Dynamic Staged Trees for Discrete Multivariate Time Series: Forecasting, Model Selection and Causal Analysis." *Bayesian Analysis*, **6**(2), 279–305. doi:10.1214/11-ba610.

Gabbiadini A, Sagioglou C, Greitemeyer T (2018). "Does Pokémon Go Lead to a More Physically Active Life Style?" *Computers in Human Behavior*, **84**, 258–263. doi:10.1016/j.chb.2018.03.005.

Glaz J, Sison C (1999). "Simultaneous Confidence Intervals for Multinomial Proportions." *Journal of Statistical Planning and Inference*, **82**(1-2), 251–262. doi:10.1016/s0378-3758(99)00047-6.

Görgen C, Bigatti A, Riccomagno E, Smith JQ (2018). "Discovery of Statistical Equivalence Classes Using Computer Algebra." *International Journal of Approximate Reasoning*, **95**, 167–184. doi:10.1016/j.ijar.2018.01.003.

Görgen C, Leonelli M, Smith JQ (2015). "A Differential Approach for Staged Trees." In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pp. 346–355. Springer-Verlag.

Görgen C, Smith JQ (2018). "Equivalence Classes of Staged Trees." *Bernoulli*, **24**(4A), 2676–2692. doi:10.3150/17-bej940.

Højsgaard S (2012). "Graphical Independence Networks with the **gRain** Package for R." *Journal of Statistical Software*, **46**(10), 1–26. doi:10.18637/jss.v046.i10.

Højsgaard S, Edwards D, Lauritzen S (2012). *Graphical Models with R*. Springer-Verlag. doi:10.1007/978-1-4614-2299-0.

Jaeger M, Nielsen JD, Silander T (2006). "Learning Probabilistic Decision Graphs." *International Journal of Approximate Reasoning*, **42**(1-2), 84–100. doi:10.1016/j.ijar.2005.10.006.

Keeble C, Thwaites PA, Barber S, Law G, Baxter P (2017a). "Adaptation of Chain Event Graphs for Use with Case-Control Studies in Epidemiology." *The International Journal of Biostatistics*, **13**(2). doi:10.1515/ijb-2016-0073.

Keeble C, Thwaites PA, Baxter P, Barber S, Parslow R, Law G (2017b). "Learning through Chain Event Graphs: The Role of Maternal Factors in Childhood Type 1 Diabetes." *American Journal of Epidemiology*, **186**(10), 1204–1208. doi:10.1093/aje/kwx171.

Leonelli M (2019). "Sensitivity Analysis Beyond Linearity." *International Journal of Approximate Reasoning*, **113**, 106–118. doi:10.1016/j.ijar.2019.06.007.

Leonelli M, Varando G (2021). "Context-Specific Causal Discovery for Categorical Data Using Staged Trees." arXiv:2106.04416 [stat.ME], URL https://arxiv.org/abs/2106.04416.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22.

Long JS (1990). "The Origins of Sex Differences in Science." *Social Forces*, **68**(4), 1297–1316. doi:10.2307/2579146.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2021). ***e1071**: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-9, URL https://CRAN.R-project.org/package=e1071.

Michalski RS, Wnek J (1993). "Comparing Symbolic and Subsymbolic Learning: Three Studies." In RS Michalski, T Tecuci (eds.), *Machine Learning: A Multistrategy Approach*. Morgan Kaufmann.

Möstel L, Pfeuffer M, Fischer M (2020). "Statistical Inference for Markov Chains with Applications to Credit Risk." *Computational Statistics*, **35**, 1659–1684. doi:10.1007/s00180-020-00978-0.

Pensar J, Nyman H, Koski T, Corander J (2015). "Labeled Directed Acyclic Graphs: A Generalization of Context-Specific Independence in Directed Graphical Models." *Data Mining and Knowledge Discovery*, **29**(2), 503–533. doi:10.1007/s10618-014-0355-0.

Pensar J, Nyman H, Lintusaari J, Corander J (2016). "The Role of Local Partial Independence in Learning of Bayesian Networks." *International Journal of Approximate Reasoning*, **69**, 91–105. doi:10.1016/j.ijar.2015.11.008.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL https://www.R-project.org/.

Riccomagno E, Smith JQ (2004). "Identifying a Cause in Models Which Are Not Simple Bayesian Networks." In *Proceedings of the 10th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pp. 1315–1322.

Riccomagno E, Smith JQ (2009). "The Geometry of Causal Probability Trees That Are Algebraically Constrained." In *Optimal Design and Related Areas in Optimization and Statistics*, pp. 133–154. Springer-Verlag.

Russell S, Norvig P (2016). *Artificial Intelligence: A Modern Approach*. Pearson.

Scutari M (2010). "Learning Bayesian Networks with the **bnlearn** R Package." *Journal of Statistical Software*, **35**(3), 1–22. doi:10.18637/jss.v035.i03.

Scutari M (2017). "Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimized Implementations in the **bnlearn** R Package." *Journal of Statistical Software*, **77**(2), 1–20. doi:10.18637/jss.v077.i02.

Shenvi A, Smith J, Walton R, Eldridge S (2019). "Modelling with Non-Stratified Chain Event Graphs." In R Argiento, D Durante, S Wade (eds.), *Bayesian Statistics and New Generations*, pp. 155–163. Springer-Verlag.

Silander T, Leong TY (2013). "A Dynamic Programming Algorithm for Learning Chain Event Graphs." In *Proceedings of the International Conference on Discovery Science*, pp. 201–216.

Smith JQ, Anderson PE (2008). "Conditional Independence and Chain Event Graphs." *Artificial Intelligence*, **172**(1), 42 – 68. doi:10.1016/j.artint.2007.05.004.

Therneau T, Atkinson B (2022). ***rpart**: Recursive Partitioning and Regression Trees*. R package version 4.1-16, URL https://CRAN.R-project.org/package=rpart.

Thwaites PA (2013). "Causal Identifiability via Chain Event Graphs." *Artificial Intelligence*, **195**, 291–315. doi:10.1016/j.artint.2012.09.003.

Thwaites PA, Smith JQ (2017). "A New Method for Tackling Asymmetric Decision Problems." *International Journal of Approximate Reasoning*, **88**, 624–639. doi:10.1016/j.ijar.2017.03.004.

Thwaites PA, Smith JQ (2018). "A Graphical Method for Simplifying Bayesian Games." *Reliability Engineering & System Safety*, **179**, 3–11. doi:10.1016/j.ress.2017.05.012.

Thwaites PA, Smith JQ, Cowell R (2008). "Propagation Using Chain Event Graphs." In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 546 – 553.

Thwaites PA, Smith JQ, Riccomagno E (2010). "Causal Analysis with Chain Event Graphs." *Artificial Intelligence*, **174**(12-13), 889–909. doi:10.1016/j.artint.2010.05.004.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York.

Wilkerson R, Smith JQ (2019). "Bayesian Diagnostics for Chain Event Graphs." arXiv:1910.04679 [stat.ME], URL https://arxiv.org/abs/1910.04679.

Zeileis A, Fisher JC, Hornik K, Ihaka R, McWhite CD, Murrell P, Stauffer R, Wilke CO (2020). "**colorspace**: A Toolbox for Manipulating and Assessing Colors and Palettes." *Journal of Statistical Software*, **96**(1), 1–49. doi:10.18637/jss.v096.i01.

Affiliation:

Gherardo Varando
Image Processing Laboratory (IPL)
Parc Científic Universitat de València
C/ Cat. Agustín Escardino Benlloch, 9
46980 Paterna (València). Spain
E-mail: gherardo.varando@uv.es

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
April 2022, Volume 102, Issue 6 *Submitted:* 2020-04-14
doi:10.18637/jss.v102.i06 *Accepted:* 2021-10-29