Package ‘jti’

July 5, 2021

Title Junction Tree Inference
Version 0.8.0
Date 2021-07-05
Description Minimal and memory efficient implementation of the junction tree algorithm using the Lauritzen-Spiegelhalter scheme;
Depends R (>= 3.5.0)
URL https://github.com/mlindsk/jti
License GPL-3
Encoding UTF-8
LazyData true
Imports Rcpp, igraph, sparta
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 7.1.1
SystemRequirements C++11
Suggests tinytest
NeedsCompilation yes
Author Mads Lindskou [aut, cre]
Maintainer Mads Lindskou <mads@math.aau.dk>
Repository CRAN
Date/Publication 2021-07-05 14:40:02 UTC

R topics documented:

  jti-package .................. ........................................ 2
  asia ................................................................. 3
  asia2 .............................................................. 3
  bnfit_to_cpts .......................... ......................... 4
jti-package

jti: Junction Tree Inference

Description


Author(s)

Maintainer: Mads Lindskou <mads@math.aau.dk>

See Also

Useful links:

- https://github.com/mlindsk/jti
Asia

Description
Small synthetic data set from Lauritzen and Spiegelhalter (1988) about lung diseases (tuberculosis, lung cancer or bronchitis) and visits to Asia. This copy of the data was taken from the R package "bnlearn" where all values "yes" have been converted to "y" and all values "no" have been converted to "n".

Usage
asia

Format
An object of class tbl_df (inherits from tbl, data.frame) with 5000 rows and 8 columns.

Details
D (dyspnea)
T (tuberculosis)
L (lung cancer)
B (bronchitis)
A (visit to Asia)
S (smoking)
X (chest C-ray)
E (tuberculosis vs cancer/bronchitis)

References
bnlearn-asia

Asia2

Description
See the asia data for information. This version, has class bn.fit.

Usage
asia2
Format

An object of class list of length 8.

References

bnlearn-asia

bnfit_to_cpts  bnfit to cpts

Description

Convert a bn.fit object (a list of cpts from the bnlearn package) into a list of ordinary array-like cpts

Usage

bnfit_to_cpts(x)

Arguments

x  A bn.fit object

compile  Compile information

Description

Compiled objects are used as building blocks for junction tree inference

Usage

compile(
  x,
  evidence = NULL,
  root_node = "",
  joint_vars = NULL,
  tri = "min_fill",
  pmf_evidence = NULL,
  alpha = NULL
)

## S3 method for class 'cpt_list'
compile(
  x,
Arguments

**x**
An object returned from `cpt_list` (Bayesian network) or `pot_list` (decomposable Markov random field).

**evidence**
A named vector. The names are the variables and the elements are the evidence.

**root_node**
A node for which we require it to live in the root clique (the first clique).

**joint_vars**
A vector of variables for which we require them to be in the same clique. Edges between all these variables are added to the moralized graph.

**tri**
The optimization strategy used for triangulation if `x` originates from a Bayesian network. One of

- 'min_fill'
- 'min_rfill'
- 'min_efill'
- 'min_sfill'
- 'min_sp'
- 'min_esp'
- 'minimal'
- 'alpha'

**pmf_evidence**
A named vector of frequencies. The names should correspond to the evidence that is expected to see over time. Relevant in connection to `min_efill` and `min_esp` triangulations.

**alpha**
Character vector. A permutation of the nodes in the graph. It specifies a user-supplied elimination ordering for triangulation of the moral graph.
Details

The Junction Tree Algorithm performs both a forward and inward message pass (collect and distribute). However, when the forward phase is finished, the root clique potential is guaranteed to be the joint pmf over the variables involved in the root clique. Thus, if it is known in advance that a specific variable is of interest, the algorithm can be terminated after the forward phase. Use the root_node to specify such a variable and specify propagate = "collect" in the juntion tree algorithm function jt.

Moreover, if interest is in some joint pmf for variables that end up being in different cliques these variables must be specified in advance using the joint_vars argument. The compilation step then adds edges between all of these variables to ensure that at least one clique contains all of them.

Evidence can be entered either at compile stage or after compilation. Hence, one can also combine evidence from before compilation with evidence after compilation. Before refers to entering evidence in the 'compile' function and after refers to entering evidence in the 'jt' function.

Finally, one can either use a Bayesian network or a decomposable Markov random field (use the ess package to fit these). Bayesian networks must be constructed with cpt_list and decomposable MRFs should be constructed with pot_list, but can also be constructed using cpt_list.

Examples

cpl <- cpt_list(asia2)
cpl <- compile(cptl, evidence = c(bronc = "yes"), joint_vars = c("bronc", "tub"))
print(cpl)
names(cpl)
dim_names(cpl)
plot(get_graph(cpl))

---

cpt_list

Conditional probability list

Description

A check and conversion of cpts to be used in the junction tree algorithm

Usage

cpt_list(x, g = NULL)

## S3 method for class 'list'
cpt_list(x, g = NULL)

## S3 method for class 'data.frame'
cpt_list(x, g)
Arguments

x Either a named list with cpts in form of array-like object(s) where names must be the child node or a data.frame

g Either a directed acyclic graph (DAG) as an igraph object or a decomposable graph as an igraph object. If x is a list, g must be NULL. The procedure then deduce the graph from the conditional probability tables.

Examples

```r
library(igraph)
el <- matrix(c(
  "A", "T",
  "T", "E",
  "S", "L",
  "S", "B",
  "L", "E",
  "E", "X",
  "E", "D",
  "B", "D"),
nc = 2,
byrow = TRUE)

g <- igraph::graph_from_edgelist(el)
cl <- cpt_list(asia, g)
print(cl)
dim_names(cl)
names(cl)
plot(get_graph(cl))
```

dim_names

Various getters

Description

Getter methods for `cpt_list`, `pot_list`, `charge` and `jt` objects

Usage

```r
dim_names(x)

has_inconsistencies(x)
```

## S3 method for class 'cpt_list'
dim_names(x)
Arguments

x  cpt_list, pot_list, charge or jt

Description

return the cliques of a junction tree

Usage

get_cliques(x)

## S3 method for class 'jt'
get_cliques(x)

## S3 method for class 'charge'
get_cliques(x)
## S3 method for class 'pot_list'
get_cliques(x)

get_clique_root_idx(x)

## S3 method for class 'jt'
get_clique_root_idx(x)

get_clique_root(x)

## S3 method for class 'jt'
get_clique_root(x)

Arguments

x A junction tree object, jt.

See Also

jt

Examples

# See Example 5 and 6 of the 'jt' function

get_graph

Description

Retrieve the graph from

Usage

get_graph(x)

## S3 method for class 'charge'
get_graph(x)

## S3 method for class 'cpt_list'
get_graph(x)

## S3 method for class 'pot_list'
get_graph(x)
Arguments

- `x`: A `cpt_list` or a compiled object

Value

A graph as an `igraph` object

---

### `jt` Junction Tree

Description

Construction of a junction tree and message passing

Usage

```r
jt(x, evidence = NULL, flow = "sum", propagate = "full")
```

Details

Evidence can be entered either at compile stage or after compilation. Hence, one can also combine evidence from before compilation with evidence after compilation. Before refers to entering evidence in the `compile` function and after refers to entering evidence in the `jt` function.

Value

A `jt` object

See Also

`query_belief`, `mpe`, `get_cliques`, `get_clique_root`, `propagate`
Examples

```r
# Setting up the network
# ----------------------

library(igraph)
el <- matrix(c(
  "A", "T",
  "T", "E",
  "S", "L",
  "S", "B",
  "L", "E",
  "E", "X",
  "E", "D",
  "B", "D"),
nc = 2,
byrow = TRUE)

library(igraph)
el <- matrix(c(
  "A", "T",
  "T", "E",
  "S", "L",
  "S", "B",
  "L", "E",
  "E", "X",
  "E", "D",
  "B", "D"),
nc = 2,
byrow = TRUE)
g <- igraph::graph_from_edgelist(el)
plot(g)

# -----------------------
# Data
# ----
# We use the asia data; see the man page (?asia)

# Compilation
# --------
cl <- cpt_list(asia, g) # Checking and conversion
cp <- compile(cl)

# After the network has been compiled, the graph has been triangulated and
# moralized. Furthermore, all conditional probability tables (CPTs) has been
# designated one of the cliques (in the triangulated and moralized graph).

# Example 1: sum-flow without evidence
# ------------------------------------
jt1 <- jt(cp)
plot(jt1)
print(jt1)
query_belief(jt1, c("E", "L", "T"))
query_belief(jt1, c("B", "D", "E"), type = "joint")

# Notice, that jt1 is equivalent to:
# jt1 <- jt(cp, propagate = "no")
# jt1 <- propagate(jt1, prop = "full")

# That is; it is possible to postpone the actual propagation
# In this setup, the junction tree is saved in the jt1 object,
# and one can repeatedly enter evidence for new observations
```
# using the set_evidence function on jt1 and then query
# several probabilities without repeatedly calculating the
# the junction tree over and over again. One just needs
# to use the propagate function on jt1.

# Example 2: sum-flow with evidence
# -----------------------------------
e2 <- c(A = "y", X = "n")
jt2 <- jt(cp, e2)
query_belief(jt2, c("B", "D", "E"), type = "joint")

# Notice that, the configuration (D,E,B) = (y,y,n) has changed
# dramatically as a consequence of the evidence
# We can get the probability of the evidence:
query_evidence(jt2)

# Example 3: max-flow without evidence
# ------------------------------------
jt3 <- jt(cp, flow = "max")
mpe(jt3)

# Example 4: max-flow with evidence
# ---------------------------------
e4 <- c(T = "y", X = "y", D = "y")
jt4 <- jt(cp, e4, flow = "max")
mpe(jt4)

# Notice, that T, E, S, B, X and D has changed from "n" to "y"
# as a consequence of the new evidence e4

# Example 5: specifying a root node and only collect to save run time
# ---------------------------------------------------------------------

cp5 <- compile(cpt_list(asia, g), root_node = "X")
jt5 <- jt(cp5, propagate = "collect")
query_belief(jt5, get_clique_root(jt5), "joint")

# We can only query from the variables in the root clique now
# but we have ensured that the node of interest, "X", does indeed live in
# this clique. The variables are found using 'get_clique_root'

# Example 6: Compiling from a list of conditional probabilities
# ---------------------------------------------------------------------

# * We need a list with CPTs which we extract from the asia2 object
#   - the list must be named with child nodes
#   - The elements need to be array-like objects
cl <- cpt_list(asia2)
cp6 <- compile(cl)

# Inspection; see if the graph correspond to the cpts
# g <- get_graph(cp6)
# plot(g)

# This time we specify that no propagation should be performed
jt6 <- jt(cp6, propagate = "no")

# We can now inspect the collecting junction tree and see which cliques
# are leaves and parents
plot(jt6)
get_cliques(jt6)
get_clique_root(jt6)

leaves(jt6)
unlist(parents(jt6))

# That is;
# - clique 2 is parent of clique 1
# - clique 3 is parent of clique 4 etc.

# Next, we send the messages from the leaves to the parents
jt6 <- send_messages(jt6)

# Inspect again
plot(jt6)

# Send the last message to the root and inspect
jt6 <- send_messages(jt6)
plot(jt6)

# The arrows are now reversed and the outwards (distribute) phase begins
leaves(jt6)
parents(jt6)

# Clique 2 (the root) is now a leave and it has 1, 3 and 6 as parents.

# Finishing the message passing
jt6 <- send_messages(jt6)
jt6 <- send_messages(jt6)

# Queries can now be performed as normal
query_belief(jt6, c("either", "tub"); "joint")
leaves

Description
Number of binary operations needed to propagate in a junction tree given evidence, using the Lauritzen-Spiegelhalter scheme

Usage
jt_nbinary_ops(x, evidence = character(0), root = NULL)

## S3 method for class 'jt'
jt_nbinary_ops(x, evidence = character(0), root = NULL)

## S3 method for class 'triangulation'
jt_nbinary_ops(x, evidence = character(0), root = NULL)

Arguments

x A junction tree object or an object returned from the triangulation function
evidence Character vector of evidence nodes
root Integer specifying the root node in the junction tree

leaves
Query Parents or Leaves in a Junction Tree

Description
Return the clique indices of current parents or leaves in a junction tree

Usage
leaves(jt)

## S3 method for class 'jt'
leaves(jt)

## S3 method for class 'jt'
parents(jt)

Arguments

jt A junction tree object, jt.

See Also

jt, get_cliques
Examples

# See example 6 in the help page for the jt function

### Description

Find the maximal prime decomposition and its associated junction tree

### Usage

```r
mpd(x, save_graph = TRUE)
```

#### S3 method for class 'matrix'

```r
mpd(x, save_graph = TRUE)
```

#### S3 method for class 'cpt_list'

```r
mpd(x, save_graph = TRUE)
```

### Arguments

- **x**: Either a neighbor matrix or a `cpt_list` object
- **save_graph**: Logical indicating if the moralized graph should be kept. Useful when `x` is a `cpt_list` object.

### Value

- **prime_ints**: a list with the prime components.
- **flawed**: indicating which prime components that are triangulated.
- **jt_collect**: the MPD junction tree prepared for collecting

### Examples

```r
library(igraph)
el <- matrix(c(
  "A", "T",
  "T", "E",
  "S", "L",
  "S", "B",
  "L", "E",
  "E", "X",
  "E", "D",
  "B", "D"),
  nc = 2,
  byrow = TRUE
)
```
```r
g <- igraph::graph_from_edgelist(el, directed = FALSE)
A <- igraph::as_adjacency_matrix(g, sparse = FALSE)
mpd(A)
```

---

**mpe**  
*Most Probable Explanation*

**Description**

Returns the most probable explanation given the evidence entered in the junction tree

**Usage**

```r
mpe(x)
```

```r
## S3 method for class 'jt'
mpe(x)
```

**Arguments**

- `x`: A junction tree object, `jt`, with max-flow.

**See Also**

- `jt`

**Examples**

```r
# See the 'jt' function
```

---

**plot.jt**  
*A plot method for junction trees*

**Description**

A plot method for junction trees

**Usage**

```r
## S3 method for class 'jt'
plot(x, ...)
```

**Arguments**

- `x`: A junction tree object, `jt`
- `...`: For S3 compatibility. Not used.
pot_list

See Also

jt

---

test

**Description**

A check and extraction of clique potentials from a Markov random field to be used in the junction tree algorithm

**Usage**

```r
pot_list(x, g)
## S3 method for class 'data.frame'
pot_list(x, g)
```

**Arguments**

- `x`: Character data.frame
- `g`: A decomposable Markov random field as an igraph object.

**Examples**

```r
# Typically one would use the ess package:
# library(ess)
# g <- ess::fit_graph(derma)
# pl <- pot_list(derma, ess::as_igraph(g))
# pl

# Another example
g <- igraph::sample_gnm(ncol(asia), 12)
while(!igraph::is.chordal(g)$chordal) g <- igraph::sample_gnm(ncol(asia), 12, FALSE)
igraph::V(g)$name <- colnames(asia)
plot(g)
pot_list(asia, g)
```
print.charge  \hspace{5cm} A print method for compiled objects

Description
A print method for compiled objects

Usage
## S3 method for class 'charge'
print(x, ...)

Arguments
x  \hspace{5cm} A compiled object
...  \hspace{5cm} For S3 compatibility. Not used.

See Also
jt

print.cpt_list  \hspace{5cm} A print method for cpt lists

Description
A print method for cpt lists

Usage
## S3 method for class 'cpt_list'
print(x, ...)

Arguments
x  \hspace{5cm} A cpt_list object
...  \hspace{5cm} For S3 compatibility. Not used.

See Also
compile
Description

A print method for junction trees

Usage

## S3 method for class 'jt'
print(x, ...)

Arguments

x A junction tree object, jt.
...
For S3 compatibility. Not used.

See Also

jt

Description

A print method for pot lists

Usage

## S3 method for class 'pot_list'
print(x, ...)

Arguments

x A pot_list object
...
For S3 compatibility. Not used.

See Also

compile
propagate

Propagation of junction trees

Description
Given a junction tree object, propagation is conducted

Usage
propagate(x, prop = "full")

## S3 method for class 'jt'
propagate(x, prop = "full")

Arguments
x A junction tree object, jt
prop Either "collect" or "full".

See Also
jt

Examples
# See Example 1 in the 'jt' function

query_belief

Query probabilities

Description
Get probabilities from a junction tree object

Usage
query_belief(x, nodes, type = "marginal")

## S3 method for class 'jt'
query_belief(x, nodes, type = "marginal")

Arguments
x A junction tree object, jt.
nodes The nodes for which the probability is desired
type Either 'marginal' or 'joint'
Description
Get the probability of the evidence entered in the junction tree object

Usage
query_evidence(x)

## S3 method for class 'jt'
query_evidence(x)

Arguments
x A junction tree object, jt.

See Also
jt, mpe

send_messages Send Messages in a Junction Tree

Description
Send messages from the current leaves to the current parents in a junction tree

Usage
send_messages(jt)

Arguments
jt A jt object return from the jt function

See Also
jt, get_cliques, leaves, parents
Examples

# See example 6 in the help page for the jt function

set_evidence Enter Evidence

Description

Enter evidence into a the junction tree object that has not been propagated

Usage

set_evidence(x, evidence)

## S3 method for class 'jt'
set_evidence(x, evidence)

Arguments

x

A junction tree object, jt.

evidence

A named vector. The names are the variabes and the elements are the evidence.

See Also

jt, mpe

Examples

# See the 'jt' function

---

sim_data_from_bn Simulate data from a Bayesian network

Description

Simulate data from a Bayesian network

Usage

sim_data_from_bn(
  net,
  lvls,
  nsims = 1000,
  increasing_prob = FALSE,
  p1 = 0.8,
  p2 = 1
)
**Arguments**

- **`net`**: A Bayesian network as an igraph object
- **`lvls`**: Named integer vector where each element is the size of the statespace of the corresponding variable
- **`nsims`**: Number of simulations distributions from which the simulations are drawn.
- **`increasing_prob`**: Logical. If true, probabilities in the underlying CPTs increases with as the number of levels increases.
- **`p1`**: Probability
- **`p2`**: Probability

**Examples**

```r
net <- igraph::graph(as.character(c(1,2,1,3,3,4,3,5,5,4,2,6,6,7,5,7)), directed = TRUE)
nodes_net <- igraph::V(net)$name
lvls_net <- structure(sample(3:9, length(nodes_net)), names = nodes_net)
lvls_net <- structure(rep(3, length(nodes_net)), names = nodes_net)
sim_data_from_bn(net, lvls_net, 10)
```

**Description**

Simulate data from a decomposable discrete Markov random field

**Usage**

```r
sim_data_from_dmrf(
  graph,
  lvls,
  nsims = 1000,
  increasing_prob = FALSE,
  p1 = 0.8,
  p2 = 1
)
```

**Arguments**

- **`graph`**: A decomposable discrete Markov random field as an igraph object
- **`lvls`**: Named integer vector where each element is the size of the statespace of the corresponding variable
- **`nsims`**: Number of simulations distributions from which the simulations are drawn.
- **`increasing_prob`**: Logical. If true, probabilities in the underlying CPTs increases with as the number of levels increases.
Description

Given a list of CPTs, this function finds a triangulation.

Usage

```r
triangulate(
  x,
  root_node = "",
  joint_vars = NULL,
  tri = "min_fill",
  pmf_evidence = NULL,
  alpha = NULL,
  perm = NULL
)
```

## S3 method for class 'cpt_list'

```r
triangulate(
  x,
  root_node = "",
  joint_vars = NULL,
  tri = "min_fill",
  pmf_evidence = NULL,
  alpha = NULL,
  perm = NULL
)
```

Arguments

- **x**: An object returned from `cpt_list` (Bayesian network) or `pot_list` (decomposable Markov random field).
- **root_node**: A node for which we require it to live in the root clique (the first clique).
- **joint_vars**: A vector of variables for which we require them to be in the same clique. Edges between all these variables are added to the moralized graph.
- **tri**: The optimization strategy used for triangulation if `x` originates from a Bayesian network. One of:
  - 'min_fill'
  - 'min_rfill'
  - 'min_efill'

`triangulate` *Triangulate a Bayesian network*
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmf_evidence</td>
<td>A named vector of frequencies. The names should correspond to the evidence that is expected to see over time. Relevant in connection to <code>min_efill</code> and <code>min_esp</code> triangulations.</td>
</tr>
<tr>
<td>alpha</td>
<td>Character vector. A permutation of the nodes in the graph. It specifies a user-supplied elimination ordering for triangulation of the moral graph.</td>
</tr>
<tr>
<td>perm</td>
<td>Experimental</td>
</tr>
</tbody>
</table>
Index

* datasets
  asia, 3
  asia2, 3
  asia, 3
  asia2, 3
  bnfit_to_cpts, 4
compile, 4, 18, 19
cpt_list, 6
dim_names, 7
  get_clique_root, 10
  get_clique_root (get_cliques), 8
  get_clique_root_idx (get_cliques), 8
  get_cliques, 8, 10, 14, 21
  get_graph, 9
  has_inconsistencies (dim_names), 7
jt, 9, 10, 14, 16–22
jt_nbinary_ops, 13
jti (jti-package), 2
jti-package, 2
leaves, 14, 21
mpd, 15
mpe, 10, 16, 21, 22
  names.charge (dim_names), 7
  names.cpt_list (dim_names), 7
  names.jt (dim_names), 7
  names.pot_list (dim_names), 7
parents, 21
parents (leaves), 14
plot.jt, 16
pot_list, 17
print.charge, 18
print.cpt_list, 18
print.jt, 19
print.pot_list, 19
propagate, 10, 20
query_belief, 10, 20
query_evidence, 21
send_messages, 21
set_evidence, 22
sim_data_from_bn, 22
sim_data_from_dmrf, 23
triangulate, 24

26