CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingenería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Simulación de redes Bayesianas

Autor: Alejandro Patricio Castelló Pascual

Director: Juan Antonio Fernández del Pozo

MADRID, JULIO 2018

Resumen

En este proyecto, partiendo de una colección de software existente, he desarrollado un paquete R estándar con el nombre de *bnidr* (*Bayesian Netoworks and Influence Diagrams for R*). El propósito de esta libreria es proporcionar multiples algoritmos de simulación de redes bayesianas ausente en la libreria de origen, y crear una implementación concorde con la orientación a objetos S4. A diferencia del software existente para la simulación de redes bayesianas, el diseño de este paquete se este centrado en la simplicidad y extensibilidad: primeramente con el objetivo de extender unas librerias que ya estan en uso, y segundo para poder ser usado comodamente por la comunidad de R.

Abstract

In this project, based on a previous software suite, I've developed a standard R package by the name of *bnidr* (*Bayesian Netoworks and Influence Diagrams for R*). The purpose of this library is to provide several bayesian network simulation algorithms which were not present in the original software, and with an implementation compliant with S4 object orientation. Unlike the current software for bayesian network simulation, this package has been designed with simplicity and extensibility in mind: firstly to extend the previous, already in use libraries, and to make it easily used by the R community.

Tabla de Contenidos

Resumen **iii**

Abstract **iv**

1 INTRODUCCIÓN Y OBJETIVOS **1**
1.1 Introducción 1
1.2 Objetivos 1

2 TRABAJOS PREVIOS **3**
2.1 Avances en redes bayesianas 3

3 DESARROLLO **6**
3.1 Estructuras de datos 6
3.1.1 `node` 7
3.1.2 `network` 8
3.1.3 Uso de tipos básicos 9
3.1.4 Validación de redes 10
3.2 Ampliación del software existente 11
3.3 Funciones y uso básico 12
3.4 Métodos exactos 14
3.5 Métodos de simulación 15
3.6 Comparación de distribuciones 16
3.7 Generación alteatoria 16
3.7.1 Muestreo Lógico Probabilístico 17
3.7.2 Muestreo Estratificado o Sistemático 18
3.7.3 Muestreo de Gibbs 19
3.8 Material gráfico 19
3.9 Documentación dentro de R 21

4 RESULTADOS Y EFICIENCIA. CASOS **23**

5 CONCLUSIONES **32**

Índice de Ilustraciones **33**

Índice de Tablas **34**

Capítulo 1

INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

Aunque el concepto de inferencia bayesiana tiene siglos de antiguedad. Las redes bayesianas no llegaron a ver la luz hasta finales de los años ochenta, gracias a décadas de investigación acerca de grafos, y como producto de lo que se empezó a llamar entonces "sistemas expertos". También conocidas como *redes probabilisticas*, estas empezaron a merecer la antención de varios grupos de investigación por su capacidad para representar conociemiento en un sistema experto: de forma cualitativa gracias a una estructura gráfica, y de forma cuantitativa con tablas de probabilidades condicionales.

No se tardó mucho en demostrar que la propagación de la evidencia en redes bayesianas de topología general es un problema NP-duro [2]. De ahí viene la división en acercamientos que da lugar a este proyecto: Existen los métodos exactos de caĺculo de probabilidades. Y los métodos aproximados, que usan simulaciones estocásticas de una red bayesiana para obtener muestras que permitan estimarlas.

1.2 Objetivos

Este trabajo se centra en algunos de estos métodos de simulación, en implementarlos en un entorno más moderno como R, sus limitaciones y rendimiento. Medidos con varios ejemplos.

Aunque ya existen varios entornos que permiten hacer amplio uso de redes bayesianas, uno de los objetivos principales de este proyecto es crear un paquete que dependa únicamente del entorno R, y con una implementación sencilla que permita su uso junto a otros programas de R, concretamente con la libreria "idr"

que implementa entre otras cosas, aprendizaje en redes bayesianas.

Este páquete no está orientado a dar una solución a un único problema, sino a poder distribuirse como cualquier otro paquete de R y usarse con cierta flexibilidad, es por ello que se ha enfocado en la concordancia con ciertos estándares en cuanto a R se refiere:

1. Los nuevos tipos de datos definidos son clases S4 bien definidas, con conversión entre tipos, implementación y creación de nuevas funciones genéricas.
   (a) Las funciones dan una cierta robustez, evitando la creación de redes cíclicas, inconsistentes o incompletas
   (b) Otros paquetes o programas pueden implementar funciones genéricas definidas por *bnidr* como `as.network`, `as.node`
2. Los resultados de las funciones de muestreo son `data.frame`, lo cual facilita el guardado de datos en ficheros externos, la acumulación de varias muestras.
3. Las funciones gráficas hacen uso y extienden el paquete *grid*. Son extensibles con el restro del paquete.
4. Las funciones están completamente documentadas en ficheros `.Rd`, los cuales proveen pagina de instrucciones de uso similares a las páginas de manual (`man`).
5. Se proporcionan varios ejemplos con el paquete, redes generadas aleatoriamente, redes correspondientes a ejemplos clásicos, llamadas a métodos de simulación con distintos parámetros.

Capítulo 2

TRABAJOS PREVIOS

2.1 Avances en redes bayesianas

Como se ha mencionado en la introducción, en la década de los ochenta se estudió en profundidad lo que llamaban "sistemas expertos", modelos computacionales capaces de asistir en la toma de decisiones basandose en conocimiento previo, uno de ellos, de tipo probabilistico, son las redes bayesianas. Originalmente el concepto fue formulado para "representar razonamiento inferencial humano" (Pearl, 1985) [7]. Fue en esta publicación donde se planteó la estructura general de las redes bayesianas: Un grafo dirigido acíclico y un mecanismo para indicar el peso de cada variable en las probabilidad condicionales.
Más adelante, en esfuerzos conjuntos se detallaron las características y limitaciones básicas de las redes bayesianas, así como su lugar entre otros sistemas expertos [1] [3]. A continuación se muestra la definición e ilustración típica de una red bayesiana:
De *Castillo et al, 1997* [1]:

Definición 1. Red Bayesiana. Una red Bayesiana es un par (D,P), donde D es un grafo dirigido acíclico, $P = p(x_1|\pi_1, ..., p(x_n|\pi_n)$ es un conjunto de n funciones de probabilidad condicionada, una para cada variable, y Π_i es el conjunto de padres del nodo X_i en D. El conjunto P define una función de probabilidad asociada mediante la factorización

$$p(x) = \prod_{i=1}^{n} p(x_i|\pi_i). \tag{2.1}$$

El grafo dirigido acíclico D es un I-mapa minimal de $p(x)$.

Un ejemplo típico, original de *Cowell et al, 1999* ilustra una de las aplicaciones más características de las redes bayesianas: diagnóstico médico. En Fig. 2.1

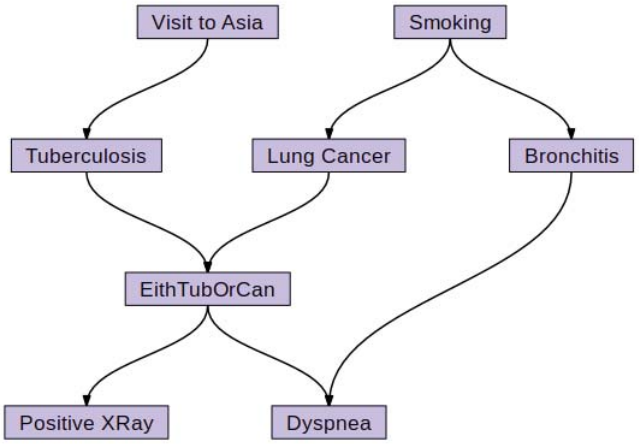


Fig. 2.1: La red Asia.

se muestra la relación entre comportamientos, enfermedades y síntomas, y en Tabla 2.1 sus distribuciones de probabilidad.
La clase de inferencia que las redes bayesianas permiten realizar es: *dada la evidencia de ciertos síntomas, cual es la probabilidad marginal de que el paciente padezca una enfermedad concreta.*

Ejemplo 1. En la red Asia, con la evidencia de que $Dyspnea = T$, $XRay = T$ y $Smoking = F$, la probabilidad de que el paciente padezca tuberculosis es $p(Tuberculosis = T) = 0{,}26$. Si no huberia evidencia, $p(Tuberculosis = T) = 0{,}0109$, si por otra parte la evidencia fuera $Dyspnea = T$, $XRay = T$ y $Smoking = F$, entonces $p(Tuberculosis = T) = 0{,}08$

Este ejemplo muestra cómo la evidencia de nodos que no son ni descendientes ni ascendientes de una variable objetivo sigue teniendo un peso relevante en la probabilidad marginal. Propagar la evidencia en una red bayesiana general es un problema NP-duro [2]. Aunque hay algoritmos para cálculos exactos en redes bayesianas de estructuras concretas que dan soluciones en tiempo lineal, Más interesantes para redes grandes son los algoritmos aproximados. Estos se basan en la toma de muestras de realizaciones obtenidas en una simulación de la red bayesiana a observar, el proceso suele ser similar para todos los algoritmos:

1. Mediante una simulación, obtener el conjunto de muestras $s(x^j)$. Donde cada x^j es una de las posible realizaciones del conjunto total de variables X.

(a) Visita Asia

F	T
0.99	0.01

(b) Fuma

F	T
0.5	0.5

(c) Tuberculosis

visita=	F	T
F	0.95	0.05
T	0.9	0.1

(d) Cáncer Pulmonar

fuma=	F	T
F	0.99	0.01
T	0.9	0.1

(e) Bronquitis

fuma=	F	T
F	0.7	0.3
T	0.4	0.6

(f) Tuberculosis o Cáncer

tub,cancer=	F	T
F,F	1	0
F,T	0	1
T,F	0	1
T,T	0	1

(g) XRay

tubocancer=	F	T
F	0.95	0.05
T	0.02	0.98

(h) Dispnea

tubocancer,bronquitis=	F	T
F,F	0.9	0.1
F,T	0.2	0.8
T,F	0.3	0.7
T,T	0.1	0.9

Tabla 2.1: Distribuciones de probabilidad condicional de la red Asia

2. Dado este espacio de muestras s, la probabilidad de un suceso $Y = y$ se estima como la proporción de las muestras en las cuales se cumple $Y = y$:

$$p(Y = y) \approx \frac{\sum_{y \in x^j} s(x^j)}{\sum_{j=1}^{N} s(x^j)} \tag{2.2}$$

Capítulo 3

DESARROLLO

El resultado de este proyecto es un paquete de R, completamente documentado en ficheros `.Rd` de acuerdo a los estándares de R [11]. El uso de este paquete abarca:

1. Creación de redes bayesianas y sus nodos, ambos implementados como objetos S4 [**?**].
   (a) Gracias a los métodos S4, se pueden obtener redes modeladas en otros programas.
2. Generación de redes aleatorias
3. Obtención de muestras mediante simulación, las cuales se guardan como objetos tipo `data.frame`.
4. Aproximación de las probabilidades marginales partiendo de dichas muestras.
5. Comparación de la calidad de las aproximaciones.
6. Creación de gráficos para visualizar las estructuras creadas.
7. La documentación del funcionamiento de modo que sean accesibles mediante las funciones de ayuda de R como `?`, `help`

3.1 Estructuras de datos

Este paquete solo implementa dos clases: `node` y `network`, por lo demás hace uso de los tipos de datos básicos de R. Ambas clases han sido creadas como objetos S4, que ha diferencia de otras formas de operar con objetos en R, son mucho más inflexibles.

3.1.1 `node`

Aunque bastante simple, tiene los componentes básicos para moverte por el grafo. Algunos de sus atributos son opcionales, muchos se inicializan o pueden inicializarse automáticamente al crease la red que contiene al nodo, como `preds`, `preds.id`, `succs`, `succs.id`. Sus atributos son:

1. `name`: El nombre del objeto, R tiene una sintaxis que permite el uso de palabras reservadas como identificadores, esto permite que el nombre del nodo sirva también para indexarlo y encontrarlo dentro de la red. Por ejemplo:

```
a.network <- network(node(name = "Nood", pots =
   c(0.3, 0.7)))  # se crea una red simple
cat(a.network[["Nood"]]@name)  # imprime el nombre
   del nodo tras encontrarlo en la red

b.network <- network(node(name = "!% _", pots =
   c(0.4, 0.6)))
cat(b.network$`!% _`)  # no da problemas en
   encontrarlo
```

2. `values`: Los posibles valores de la variable aleatoria asociada al nodo, debe ser un vector de tipo `character`, y por defecto adopta el valor `c("F", "T")`, aunque puede ser de mayor longitud.

3. `pots`: Las probabilidades condicionales/marginales del nodo. Al inicializarse la red que contenga al nodo, y hacerse conocidos todos sus predecesores, y la longitud de los valores de estos; se validará la longitud del dato. Es de tipo `matrix` de `double`, pero puede pasarse al constructor como una matriz o como un vector. Las columnas están ordenadas para que los `double` correspondan al atributo `values`, el orden de las filas tiene un significado muy concreto cuando el nodo no es marginal: son las probabilidades de que la variable tome sus valores dado los valores de los predecesores en orden lexicográfico. Por ejemplo:

```
a.network <- network(
    node("A", pots = c(0.1, 0.9)),
    node("B", pots = c(0.2, 0.8,
                       0.3, 0.7),
         preds = "A"),
    node("C", pots = c(0.4, 0.6,  # F,F
                       0.5, 0.5,  # F,T
                       0.6, 0.4,  # T,F
```

```
                    0.7, 0.3), # T,T
        preds = c("A","B"))
```

La primera fila del nodo `C` corresponde a $P(C|A = F, B = F)$, la segunda a $P(C|A = F, B = T)$, y asi sucesivamente en orden lexicográfico. Es importante entonces prestar atención a la hora de escribir las probabilidades y el orden de los predecesores.

4. `preds`: Predecesores, nombres de los nodos padres/predecesores de este dentro de la red, es la forma más clara de especificar la ascendencia del nodo. Al crearse la red que contiene al nodo este campo puede ser actualizado si otro nodo tiene a este en su campo `succs`.

5. `succs`: Sucesores, nombres de los nodos hijos/sucesores. En el funcionamiento estándar este atributo se inicializa al construirse la red que contiene al nodo.

6. `preds.id`, `succs.id`: Similarmente a `preds` y `succs`, pero con el índice numérico que corresponde a los nodos dentro de la red.

7. `type`: Presente por compatibilidad con *idr-prj* y para futuras extensiones. Su valor puede ser uno de `"DECISION"`, `"UTILITY"`, `"CHANCE"`, `"DETERMINISTIC"`, `"xDECISION"`, `"xUTILITY"`, `"xCHANCE"`, `"xDETERMINISTIC"`. Pero si no se espefica el valor por defecto es `"CHANCE"`.

Se inicializa con la función `node` que acepta todos los comportamientos descritos. También se puede convertir una `list` cuyos campos tengan nombres similares a los de los atributos de `node` via la función `as.node`. Esta función es `generic`, lo cual permite a cualquier otra clase crear una implementación eficiente para convertirse en `node`.

3.1.2 `network`

La clase clave del paquete, aunque es incluso más simple que `node`, prácticamente todas las funciones de la librería requieren una `network` como primer parámetro. A efectos prácticos esta clase es un envoltorio alrededor de una `list` de `node`s, pero `network` es mucho mas inflexible y verifica en varios puntos que se cumplan las propiedades necesarias para funcionar. Solo tiene 3 atributos:

1. `nodes`: una `list` de `node`s, generalmente no se accede directamente a este campo; el constructor de `network` retoca los atributos de los nodos para tener una estructura bien conexa, y generalmente se accede a los nodos a través de las funciones genéricas que `network` implementa.

2. `autoupdated`: Si esta variable es `FALSE`, no se verificará que la red sigue siendo acíclica tras insertar nuevos nodos. `TRUE` por defecto.

3. `adjm`: la matriz de adyacencia del grafo asociado. Esta se crea al crear la red, y sirve para comprobar si hay ciclos en la red, si la red tiene forma de poliárbol, entre otras operaciones.

Los objetos de tipo `network` no están pensados para ser modificados después de su creación, sino para servir de input, junto a otros parámetros que uno podría pensar que deberían ir dentro de `network`, para distintas funciones. Por ejemplo

```
## se inicializa una red
a.network <- network(node("A", c(0.3, 0.7)),
                     node("B", c(0.4, 0.6,
                                 0.5, 0.5), preds = "A"))
## calcula e imprime por pantalla la inferencia bayesiana
## simple dado B = T, la evidencia se ofrece como una
## variable aparte de la red
print(posterior.dist(a.network, "A",
                     evidence = c(B = "T")))
```

3.1.3 Uso de tipos básicos

Una de las directrices de este paquete es usar clases S4 para las estructuras complejas, y los tipos básicos solo para objetivos simples relacionados con el tipo en sí.

- `data.frame`: Dentro de los tipos básicos de R, `data.frame` es de los más potentes y usados. En esta librería `data.frame` se utiliza principalmente para guardar resultados de las funciones de muestreo, la primera columna del `data.frame` tiene nombre `NA`, y contiene el recuento de ocurrencias de la realización descrita en el resto de columnas. El siguiente ejemplo es una muestra en la que la realización de los nodos A, B, C, D con valor T, T, F, T tiene un recuento 1, la realización T, F, F, F tiene un total de 0.57.

  ```
    NA    A B C D
  1  1    T T F T
  2 12    T T T F
  3 0.57  T F F F
  ```

  Más adelante se muestra como algunos algoritmos permite en efecto recuentos decimales. `data.frame`s como esta se pueden combinar y extender y usarse para estimar probabilidades marginales y posteriores.

- `vector`: *bnidr* hace uso extensivo de arrays nombrados con un significado muy concreto: representar evidencia. Más adelante se verán todas las funciones que aceptan evidencia como parte de sus parámetros, pero siempre tienen el mismo aspecto:

```
>>> c(A = "F", B = "T", E = "Z")
 A B E
 F T Z
```

  `vector`es como este se usan en una gran cantidad de funciones, aunque en todos los casos, `list` tambien se acepta, pero la construcción con `c()` debería ser más cómoda.

- `list`: Cualquier tipo de dato medianamente anidado se representa o se exige que sea representado como `list`. Por ejemplo, el resultado de calcular probabilidades marginales/posteriores de más de un nodo es una `list`, con cada elemento teniendo el nombre del nodo al que corresponde y su valor siendo un `vector` de probabilidades.

- `grob`: Más adelante en la sección 3.8 se ilustran las funciones que permiten generar elementos gráficos.

3.1.4 Validación de redes

Una característica de los objetos S4 es que tienen *funciones de validación*, las cuales permiten verificar la consistencia de los objetos mientras están siendo inicializados, y devolver un error si es necesario. Dentro de `node`: se comprueba que las probabilidades sumen 1.0, que el nombre no sea `NULL`, que no se tenga a sí mismo de predecesor y que los tipos de las atributos sean compatibles. En `network` se busca que no halla ciclos en el grafo, se da un `warning` si un nodo todavía no tiene sus predecesores o sucesores en la red, y se verifica que el número de valores en las probabilidades de cada nodo concuerde con la cardinalidad del espacio de condiciones y valores. Por ejemplo:

```
a.network <- network(node("A", c(0.5, 0.5),
                     node("B", c(0.3, 0.4), preds = "A"));
## Esta red dara un error porque el nodo B
## deberia tener 4 valores en sus probabilidades
```

3.2 Ampliación del software existente

Este paquete se ha desarrollado con la idea de ampliar y formalizar una librería de redes bayesianas y diagramas de influencia preexistente (*idr-prj*). Aunque no se han implementado todas las prestaciones del software original, se ha priorizado la extensibilidad de la nueva librería y parte de la orientación a objetos S4 permite la conversión entre tipos, en concreto, listas, las cuales son la base del funcionamiento de la librería preexistente.

Ejemplo 2. Parte de la definición de la red Asia en *idr-prj* es

```
asia = list(
  Smoking = node( Type = "CHANCE", Name = "Smoking",
                  Values = c("NonSmoker","Smoker"),
                  Preds = c(),
                  Pots=matrix( data = c(
                                      0.50, 0.50),
                               nrow=1, ncol=2,
                               byrow=TRUE, dimnames=NULL)),

  LungCancer = node( Type = "CHANCE", Name = "LungCancer",
                     Values=c("ABSENT","PRESENT"),
                     Preds=c("Smoking"),
                     Pots=matrix( data = c(
                                        0.99, 0.01,
                                        0.90, 0.10),
                                  nrow = 2, ncol = 2,
                                  byrow = TRUE,
                                  dimnames = NULL)),

  Bronchitis = node( Type = "CHANCE", Name = "Bronchitis",
                     Values=c("ABSENT","PRESENT"),
                     Preds=c("Smoking"),
                     Pots=matrix( data = c(
                                        0.70, 0.30,
                                        0.40, 0.60),
                                  nrow = 2, ncol = 2,
                                  byrow = TRUE,
                                  dimnames = NULL))
)
```

Esto devuelve una `list` en R, con cada elemento siendo un nodo, los cuales a su vez también son listas con campos. Facilmente se puede convertir en una

`network` del paquete *bnidr* con

```
bn.asia <- bndir::as.network(asia)
```

La cual nos permite obtener resultados y dibujar un gráfico de la red.

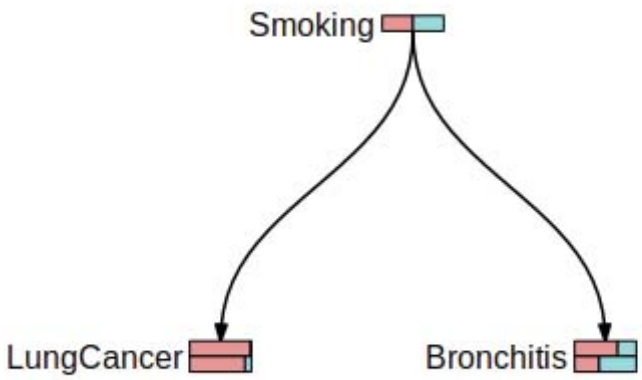


Fig. 3.1: Subred de Asia, importada desde un programa en *idr-prj*

3.3 Funciones y uso básico

Aunque tanto `network` como `node` tienen represenatciones internas de bajo nivel, hay un cierto número de funciones básicas pensadas para ahorrar frustración y facilitar tareas básicas, como por ejemplo bbtener subconjuntos de nodos y/o sus valores/realizaciones:

Nombre	Parámetros	Descripción
`network.marginals`	`net`: una `network`	Devuelve un vector de los nombres de todos los nodos marginales de la red.
`network.terminals`	`net`: una `network`	Devuelve un vector de los nombres de todos los nodos terminales de la red.

`well.ordered.nodes`	`net`: una `network`	Devuelve un vector con los nombres de todos los nodos de la red en un orden ancestral.
`markov.blanket`	`net`: una `network` `nodes`: un nodo o su nombre	Devuelve la manta de markov de un nodo: sus predecesores, sucesores y los predecesores de sus sucesores.
`nodes.values`	`net`: una `network` `nodes`: un vector de nombres de nodos o una lista de nodos `evidence`: un vector o lista de evidencias	Devuelve una lista de vectores `character`, la lista está indexada por los nombres de los nodos provistos. Los nodos evidenciales solo tienen su valor evidencial.
`realizations.count`	`net`: una `network` `nodes`: un vector de nombres de nodos o una lista de nodos	Devuelve la cantidad de realizaciones distintas de los nodos.
`nth.realization`	`net`: una `network` `nodes`: un vector de nombres de nodos o una lista de nodos `i`: `integer` índice deseado Formato alternativo: `values`: lista de todos los valores posibles de cada nodo `i`: `integer` índice deseado	Devuelve la í-esima iteración lexicográfica de los posibles valores de los nodos

`nodes.realizations`	`net`: una `network` `nodes`: un vector de nombres de nodos o una lista de nodos	Devuelve un `data.frame` conteniendo todas las realizaciones posibles de esos nodos con esa evidencia. Versión de consumo intensivo de `nth.realization`
`nodes.values.vector`	`node`: un nodo `valor a repetir`	Devuelve un vector de la misma longitud que la de los valores del nodo
`cond.probability`	`net`: una `network` `node`: un nodo o su nombre `evidence`: un vector o lista de evidencias	Devuelve un vector de probabilidades.

Tabla 3.1: Funciones Básicas del paquete

3.4 Métodos exactos

Aunque esta no sea la fuerte del paquete, se han implementado un par de métodos básicos para trabajar con resultados exactos.

`eval.polytree`

Este algoritmo permite hallar las probabilidades marginales/posteriores de una red bayesiana en tiempo lineal, siempre y cuando el gráfo tenga estructura de poliárbol (Kim y Pearl (1983)) [5]. La función `is.polytree` permite comprobarlo. Sus argumentos son:

- `net`: una `network` poliárbol.
- `evidence`: evidencia en la forma de una `list` o `vector`.

`posterior.dist`

El algoritmo de fuerza bruta básico, permite hallar la distribución de probabilidad posterior de un nodo de forma exacta en tiempo exponencial. Sus parámetros son:

- `net`: una `network`.

- `node`: uno o varios `node`s, o sus nombres.
- `evidence`: evidencia en la forma de una `list` o `vector`.
- `marginalized`: lista de probabilidades marginales halladas de nodos no marginales.

`marginal.dist`

Un alias de `posterior.dist` que no admite evidencia. Con argumentos:

- `net`: una `network`.
- `node`: uno o varios `node`s, o sus nombres.

`joint.prob`

Calcula la probabilidad conjunta de una realización (posiblemente parcial).

- `net`: una `network`.
- `targets`: Realización objetivo en la forma de una `list` o `vector`.

3.5 Métodos de simulación

El objetivo central de paquete. Todas estas funciones devuelven el mismo tipo de datos: `data.frame`, como se explica en la sección 3.1.3. Estos `data.frame` son capaces de contener, guardar en disco, restaurar, concatenar y comprimir cantidades ingentes de datos, la forma en la que se hace uso de ellos es mediante obtención de muestras; con los algorítmos de las secciones siguientes; y mediante el recuento de ocurrencias. El concepto origina del siguiente teorema [10][9][4][8]. De Castillo et al (1997)[1]

Teorema 3.5.1 (El método de aceptación-rechazo.)**.** *Sea X una variable aleatoria con función de probabilidad p(x). Supóngase que p(x) puede ser expresada como*

$$p(x) = cg(x)h(x) \tag{3.1}$$

donde c 1, 0 g(x) 1 y h(x) es una función de probabilidad. Sea U una variable aleatoria uniforme U(0, 1) y sea Y una variable aleatoria con función de probabilidad h(y) independiente de U. Entonces, la función de probabilidad condicional de Y dado que $u \leq g(y)$ coincide con la función de probabilidad de X. Por otra parte, la probabilidad de aceptar la muestra (eficiencia) es 1/c.

Este teorema da las bases para los métodos estocsticos de rechazo, estos siendo aquellos en los cuales la muestra uniforme u se rechaza cuando $u < g(y)$. Pero más importante todavía es la obtención de otra formula para llegar a $p(x)$: [1]

$$p(x) = \frac{p(x)}{h(x)} h(x) = s(x)h(x) \tag{3.2}$$

donde

$$s(x) = \frac{p(x)}{h(x)} \tag{3.3}$$

Y $s(x)$ será conocido como la muestra de x. Es de aquí de donde sale la aproximación de $p(x)$ mediante muestreo. $s(x)$ se calculará como una frequencia de ocurrencias:[1]

$$p(y) \approx \frac{\sum\limits_{y \in x^j} s(x^j}{\sum\limits_{j=1}^{N} s(x^j)} \tag{3.4}$$

3.6 Comparación de distribuciones

La necesidad de medir la calidad de las aproximaciones no es única al campo de la probabilidad, por ello en la libreria se incluye una implementación de la divergencia Kullback-Leibler [6]. La cual permite cuantificar cuanto distan nuestras aproximaciones de una distribución ideal; ya sea calculada exactamente o estimada con un N muy grande. Se tiene entonces en *bnidr* la función `KL.div` que toma dos `list`/`vectors` P y Q, siendo P la de mayor calidad y devuelve la divergencia Kullback-Leibler $D_{KL}(P||Q)$

3.7 Generación alteatoria

El paquete también implementa una función de generación aleatoria de redes bayesianas. `random.network` con parámetros N y M indicando el número de nodos y el número máximo de predecesores respectivamente, también permite proporcionar intervalos para las probabilidades de los nodos.

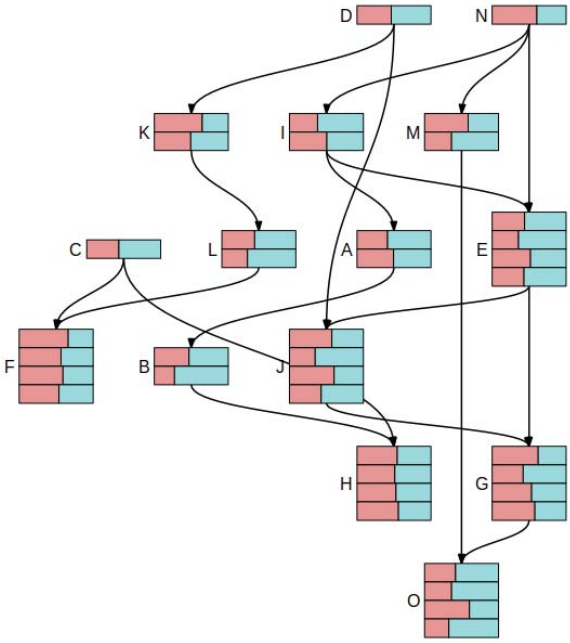


Fig. 3.2: Red aleatoria con N = 15, M = 2

3.7.1 Muestreo Lógico Probabilístico

Este primer algoritmo es relativamente simple y efectivo. Consiste en, dada una ordenación ancestral X de todos los nodos de la red, se van generando valores de X_i usando su probabilidad condicional a los nodos anteriores $X_1, ..., X_{j-1}$ como vector de pesos para ponderar una función de toma de muestras aleatorias e ir así generando una realización completa para agregar a la muestra. Si en cualquier punto de este paso se obtiene un valor que contradiga a la evidencia proporcionada, se deshechan los valores $X_1, ..., X_j$ y se vuelve a empezar. El resultado final es una muestra de N realizaciones. La forma en la que *bnidr* permite hacer uso de estas muestras es mediante la función `posterior.by.sample`, que cuenta el número de ocurrencias de los valores de una variable dentor de la muestra, y lo normaliza por el total de la muestra, obteniendo la aproximación de $P(X)$. Este algoritmo tiene una convergencia considerablemente buena para lo simple y paralelizable que es. Como se verá en el Capítulo 4.

3.7.2 Muestreo Estratificado o Sistemático

Aunque se trata de un muestreo, este algoritmo es determinista, se basa en repartir el intervalo $(0, 1)$ en N puntos de control f_j y mapear realizaciones al intervalo $(0, 1)$ mediante su función de probabilidad conjunta.

1. Al igual que en el algoritmo anterior, se necesita una ordenación ancestral de los nodos X También se requiere adoptar un orden para permutar la realización según avanzan los valores de f_j, el orden lexicográfico es trivial y el software lo facilita.

2. Ahora, tomando la realización inicial (en el caso booleano: F para todos las variables), se inicializan los límites superiores (el inferior es 0 para todos), se va reduciendo, dado que el límite superior u_i tiene el valor $P(x_1, ..., x_i)$, es decir, se va calculando la probabilidad del valor en x_i condicionado a sus predecesores y multiplicado por u_{i-1}, el último u_j tendrá un valor igual al de la probabilidad conjunta de toda la realización.

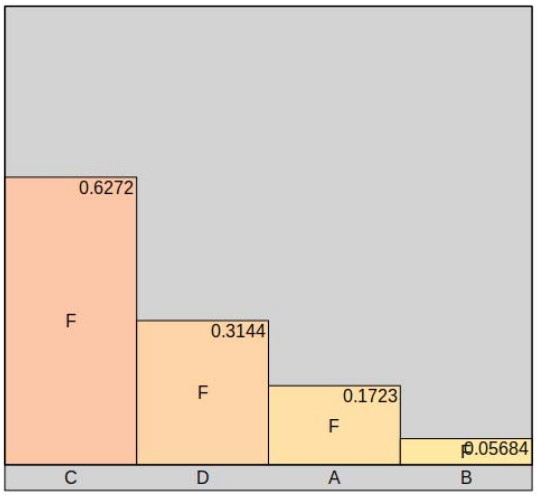


Fig. 3.3: Intervalo de la primera realización

3. $j < -1$. Mientras $j < N$ se lleva a cabo la búsqueda de la evidencia correspondiente

   3.1. Dado que conocemos el límite superior u_j calculado, podemos determinar a partir de qué i habrá que iterar lexicográficamente para llegar al u_i que queda por encima de f_j: iteramos hasta lograr un $u_i > f_j$. Se alcanzaría algo como lo siguiente:

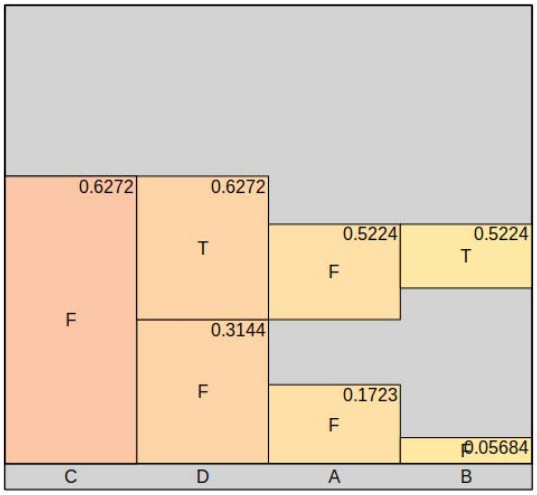


Fig. 3.4: Realización encontrada para $f_j = 0{,}44$

3.2. Puesto que conocemos f_j y u_i, siendo los f_j equidistantes podemos saber para cuantos j se cumple u_i ¿f_j, a ese valor δ, y nos permite no aumentar la j y tomar la misma realización de nuevo. Este δ es el contador de cuantos f_j aparecen en la realización actual, solo queda ajustar δ para que refleje la probabilidad real de esa realización, dada la evidencia. El resultado será una fila de un `data.frame`

3.3. Haciendo $j < -j + \delta$ y partiendo de la realización obtenida, repetir el paso 3.1 hasta que $j > N$

El resultado es un `data.frame` con frecuencias de realizaciones, a ser usado como el el algoritmo anterior.

3.7.3 Muestreo de Gibbs

Este tipo de muestreo pertenece a la familia de los métodos de Monte Carlo en cadenas de Markov (MCMC) y se basa en iterar repetidamente los valores una realización, y guardando cada iteración. Eventualmente un subconjunto de estas realizaciones estarán en torno a la probabilidad real.

3.8 Material gráfico

Gran parte de la ayuda gráfica expuesta en este documento se ha generado con funciones incluidas en *bnidr*, concretamente las funciones `networkGrob` y

`nodeGrob` hacen uso y extensión de la librería estándar de R *grid*, y permiten una cierta variedad a la hora de renderizar las redes generadas.

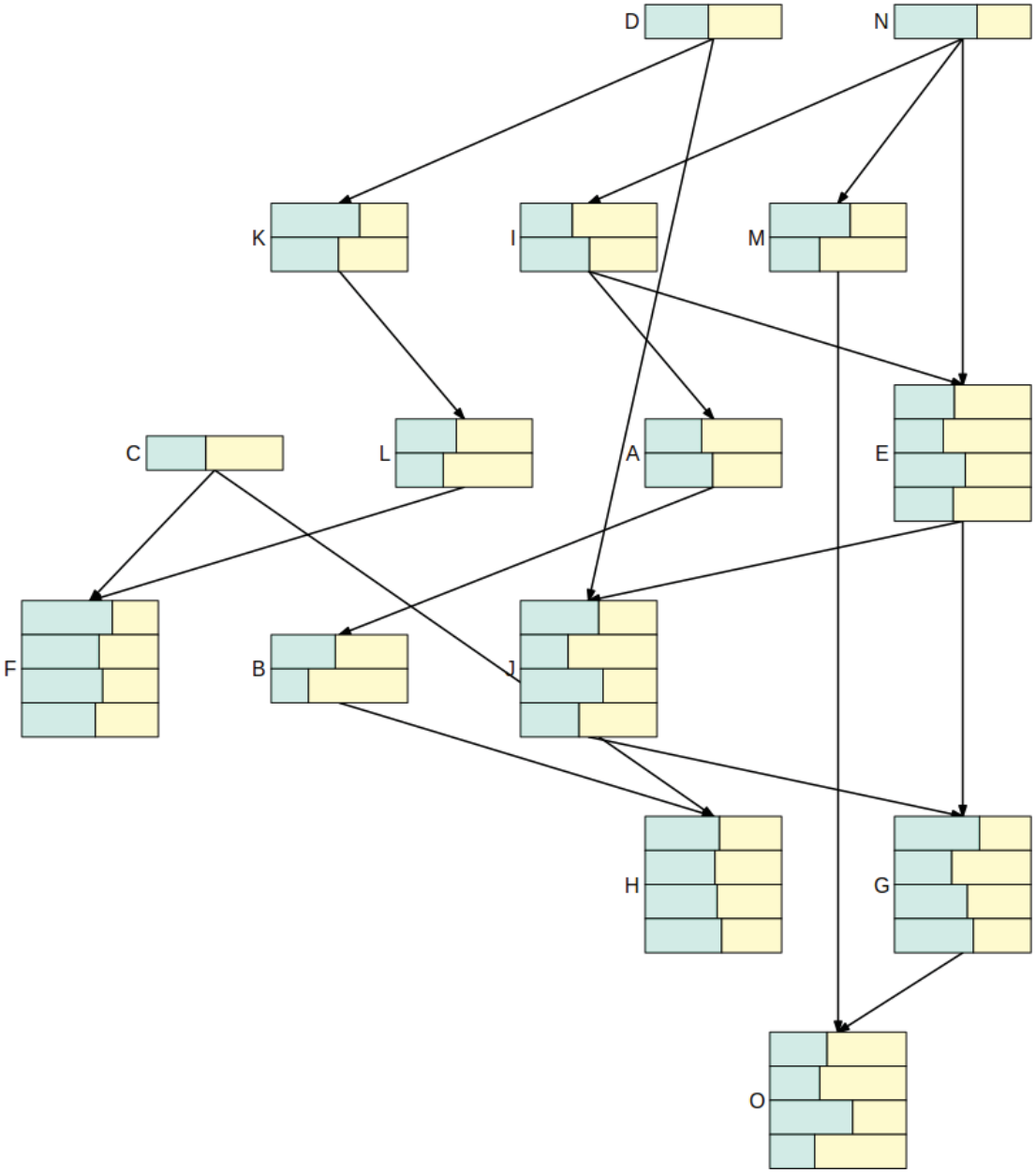
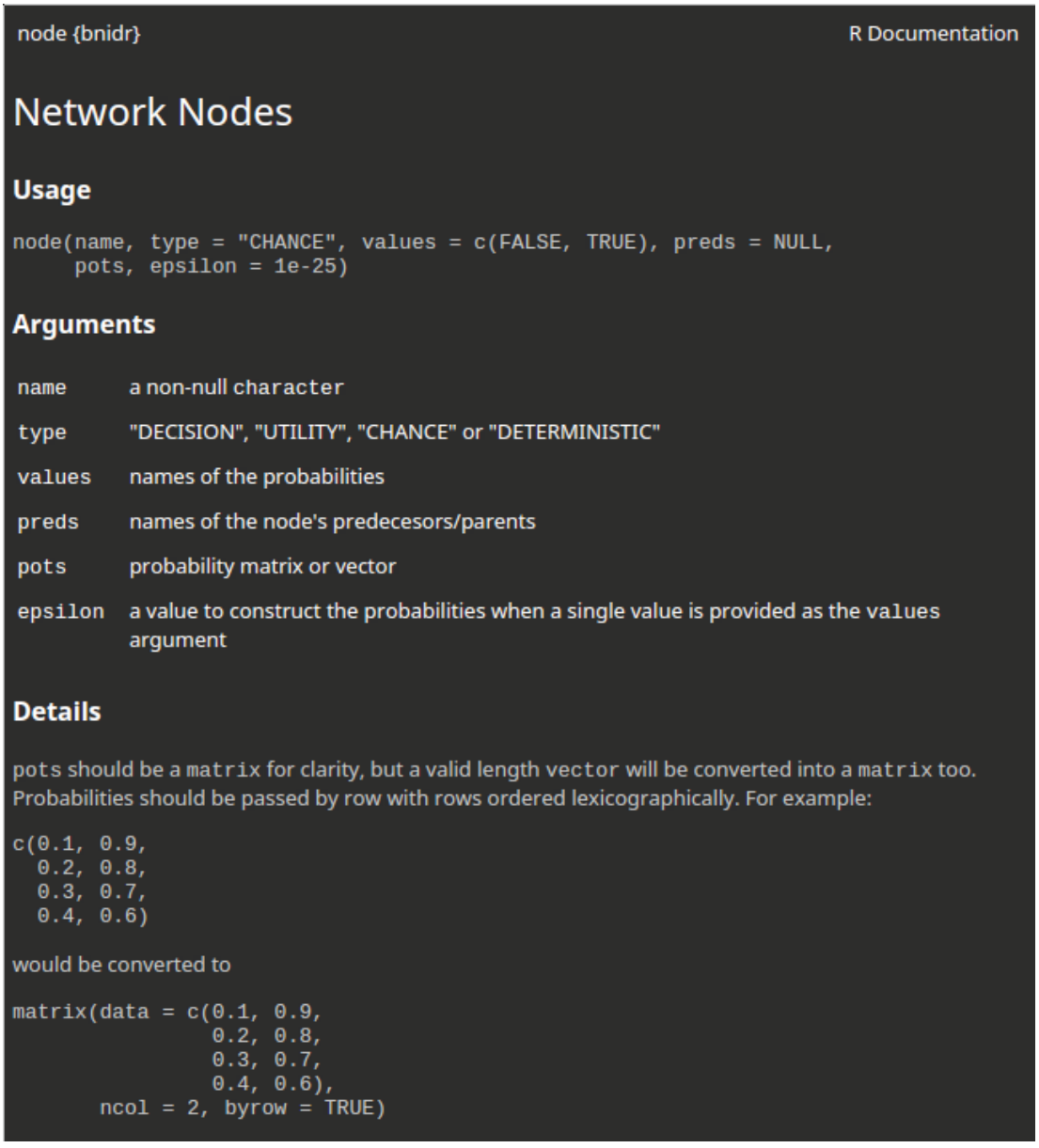


Fig. 3.5: Una ilustración de una red bayesiana aleatoria

3.9 Documentación dentro de R

R proporciona un mecanismo para documentar las funciones que se van implementando similar a las *man pages*, uno de los objetivos esenciales de este proyecto es generar documentación extensiva de todas las funciones.



node {bnidr} R Documentation

Network Nodes

Usage

```
node(name, type = "CHANCE", values = c(FALSE, TRUE), preds = NULL,
     pots, epsilon = 1e-25)
```

Arguments

name	a non-null character
type	"DECISION", "UTILITY", "CHANCE" or "DETERMINISTIC"
values	names of the probabilities
preds	names of the node's predecesors/parents
pots	probability matrix or vector
epsilon	a value to construct the probabilities when a single value is provided as the values argument

Details

pots should be a matrix for clarity, but a valid length vector will be converted into a matrix too. Probabilities should be passed by row with rows ordered lexicographically. For example:

```
c(0.1, 0.9,
  0.2, 0.8,
  0.3, 0.7,
  0.4, 0.6)
```

would be converted to

```
matrix(data = c(0.1, 0.9,
                0.2, 0.8,
                0.3, 0.7,
                0.4, 0.6),
       ncol = 2, byrow = TRUE)
```

Fig. 3.6: Página de ayuda de R de la función de *bnidr* `node`

Capítulo 4

RESULTADOS Y EFICIENCIA. CASOS

En la sección 2.1 se cita la demostración de que hallar la probabilidad marginal exacta de un nodo en una red Bayesiana es un problema NP-duro, esto se hace evidente en nuestra librería: en Fig. 4.1 se puede apreciar el tiempo exponencial requerido para computar la probabilidad marginal de toda una red Bayesiana con el método de fuerza bruta.

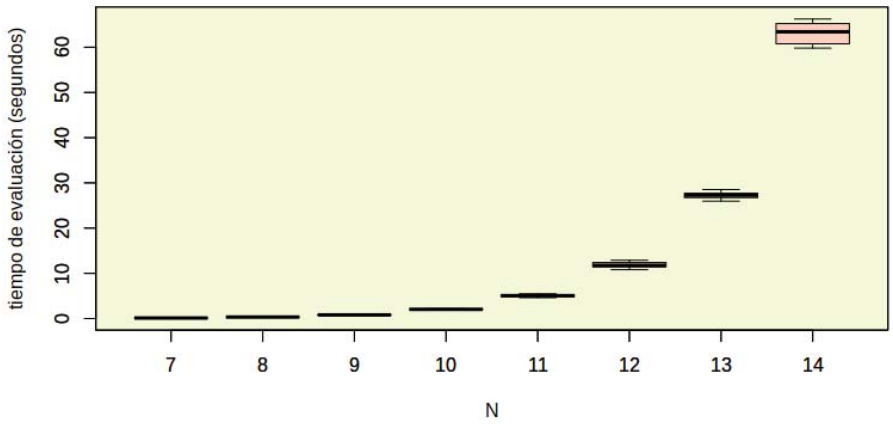


Fig. 4.1: Tiempo requerido para redes aleatorias de N nodos y hasta 2 predecesores por nodo. 8 Redes para cada N

A continuación se muestra el proceso de simular una red típica con varios algoritmos:

Las métricas que vamos a tomar son tiempo de computación y divergencia Kullback-Leibler, compararemos la evolución de la segunda respecto a la primera en cada algorítmo para dar una visualización de las limitaciones que tienen.

La red Asia

Sin Evidencia

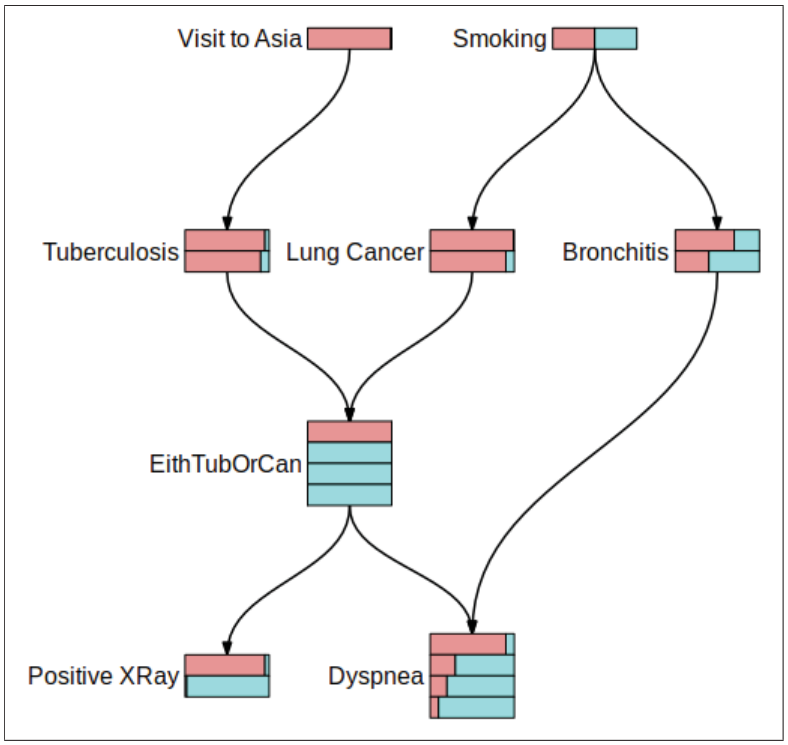


Fig. 2.1 La red Asia.

La red Asia se evalúa en `1.876s`, para solo tener 8 nodos no es un tiempo despreciable, pero nos da las medidas con las cuales comparar las simulaciones.

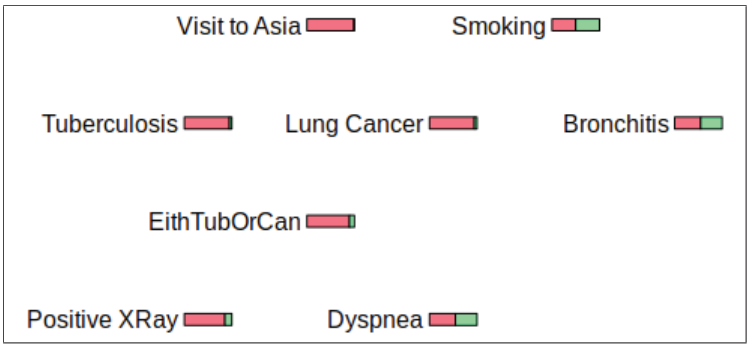


Fig. 4.2: La red Asia, marginalizada.

Muestreo Lógico Probabilístico

1. Se comienza por obtener una ordenación ancetral de los nodos de 2.1, esta no es necesariamente única, una de ellas es:

$$V \to S \to T \to C \to E \to B \to X \to D$$

2. A continuación se van generando posibles valores de cada nodo, como no tenemos evidencia, no vamos a tener ningún caso en el que paremos en este paso:

   V $P(V = F) = 0{,}99, P(V = T) = 0{,}01$, de modo que se aleatoriamente se elige: F con probabilidad 0.99, T con 0.01. Supongamos que hemos obtenido $V = F$

   S S es independiente de V de modo que se procede de forma similar: $P(S = F) = 0{,}5$, de modo que es F con un 0.5 de probabilidad y T con un 0.5. Supongamos que se obtiene $S = T$

   T T es condicional a V, tenemos $P(T = F|V = F) = 0{,}95$, seleccionamos estocásticamente con pesos 0,95 y 0,05, tenemos $T = F$

   C, E, B, X De forma similar, digamos que hemos obtenido $C = F, E = F, B = T, X = F$

   D $P(D = F|B = T, E = F) = 0{,}2$, aleatoriamente F con 0.2 de probabilidad y T con 0.8 obtenemos $D = T$

   Tenemos de este modo la realización $V = F, S = T, T = F, C = F, E = F, B = T, X = F, D = T$. En nuestro programa, esto se traduce en una fila de un `data.frame` y tiene este aspecto:

	NA	Visit	Smok	Tub	Cancer	Eith	Dyspnea	Bronchitis	XRay
1	1	F	T	F	F	F	T	T	F

   Donde `NA` es la columna donde se indica, el recuento de esta realización, a pesar de que esta misma realización puede darse en otra fila más adelante en vez de sumarse a esta. Esto es así para poder usar las muestras de distintos algorítmos de forma similar, y siempre se puede iterar manualmente luego para eliminar filas repetidas.

3. Se repite el paso 2 `N` veces, obteniendo un `data.frame` con `N` filas.

Con el `data.frame` resultante podemos estimar las probabilidades. Por ejemplo: en sintaxis de R, si quisiéramos hallar $P(Dyspnea = T)$, lo estimaríamos mediante la frecuencia con la cual la columna `Dyspnea` tiene valor `T` en la muestra. Si solo tuviésemos la primera muestra, $P(Dyspnea = T)$ sería 1, si tras 100 muestras, 48 tienen valor T, la probabilidad marginal $P(Dyspnea = T)$ quedaría aproximada a 0.55 (el método exacto devolvió 0.454). Veamos ahora como se comporta en el caso real, primero viendo como aproxima $Dyspnea$, y luego observamos las demás métricas: Vease Tab. 4.1

N	$P(D = T)$ approx	$P(D = T)$ real	ϵ	$D_{KV}(D_{approx}\|D_{real})$
5	0.8	0.45352	0.253	0.253
15	0.6667	0.45352	0.2131	0.092
25	0.36	0.45352	0.0935	0.018
70	0.3714	0.45352	0.0821	0.0138
202	0.4059	0.45352	0.0476	0.0046
583	0.4322	0.45352	0.0213	0.00092
991	0.441	0.45352	0.0126	0.00032
1684	0.4483	0.45352	0.0052	0.00005
2863	0.469	0.45352	0.0156	0.00049
4867	0.4522	0.45352	0.0013	0.000005

Tabla 4.1: Estimaciones de $P(Dyspnea = T)$

Como se puede observar, este algorítmo no parece converger ni rápida ni uniformemente, pero incluso con un bajo número de muestras (25-70) hemos obtenido estimaciones aceptables de la probabilidad que estabamos buscando. Veamos ahora que resultados obtiene en la divergencia Kullback-Leibler de todas las probabilidades marginales:

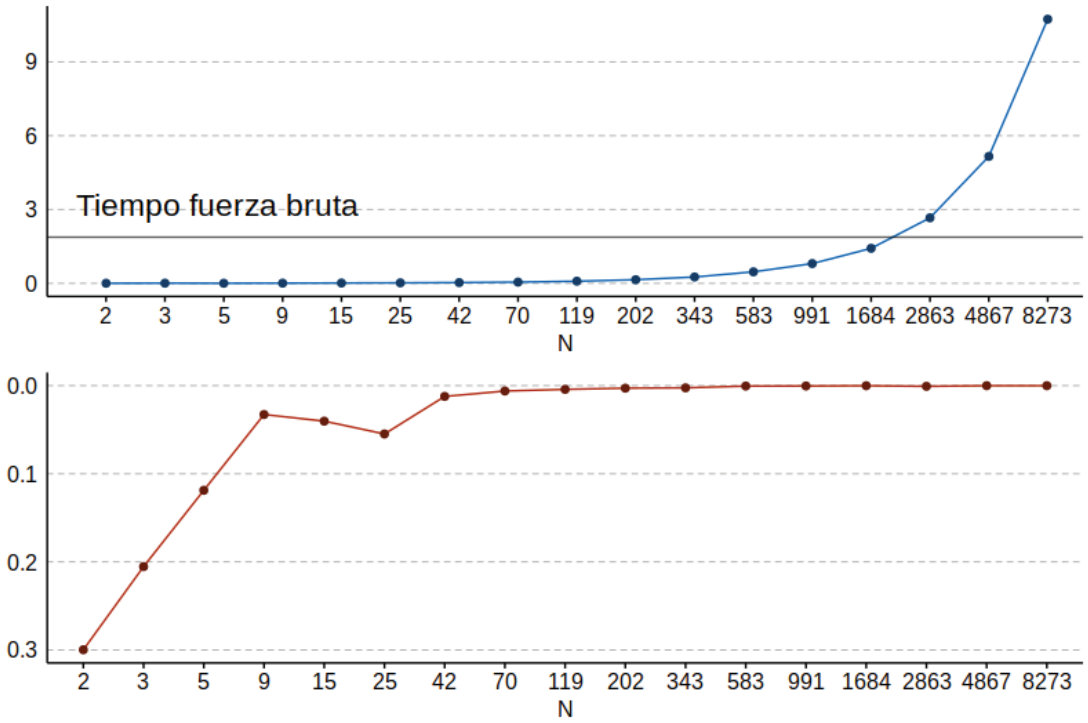


Fig. 4.3: Tiempo (s) y divergencia Kullback-Leibler para N tomadas cada vez

En este caso el algoritmo alcanza un valor razonable al menos un orden de magnitud más rapido que el método de fuerza bruta.

Muestreo Estratificado Partiendo de un tamaño n, se tienen los N intervalos f_j a lo largo del intervalo $(0, 1)$,

1. Al igual que en el algoritmo anterior, se necesita una ordenación ancestral de los nodos, la misma de antes sirve:

$$V \to S \to T \to C \to E \to B \to X \to D$$

   También se requiere adoptar un orden para permutar la realización según avanzan los valores de f_j, el orden lexicográfico es trivial y el software lo facilita.

2. Ahora, tomando la realización inicial (en este caso F para todos las variables), se inicializan los límites superiores (el inferior es 0 para todos), se va reduciendo, dado que el límite superior u_i tiene el valor $P(x_1, ..., x_i)$, es decir, se va calculando la probabilidad conjunta de los valores según se avanza por la realización.

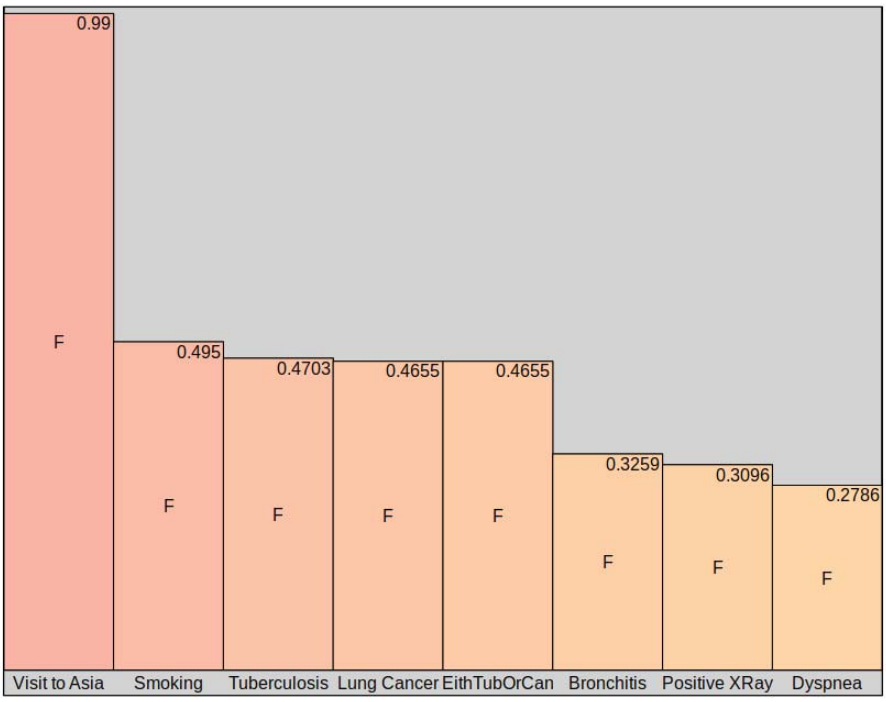


Fig. 4.4: Intervalo de la primera realización

3. $j < -1$. Mientras $j < N$ se lleva a cabo la búsqueda de la evidencia correspondiente

   3.1. Dado que conocemos el límite superior de cada i calculado, podemos determinar a partir de qué i habrá que iterar lexicográficamente para llegar al u_i que queda por encima de f_j. Por ejemplo, supongamos que $f_j = 0{,}44$, podemos determinar que no hay que iterar antes de $Bronchitis$, dado que u_5 ($EithTubOrCan$) es mayor que 0,44, iteramos hasta lograr un $u_i > f_j$. En este caso, se llega a la realización de la Figura 4.5

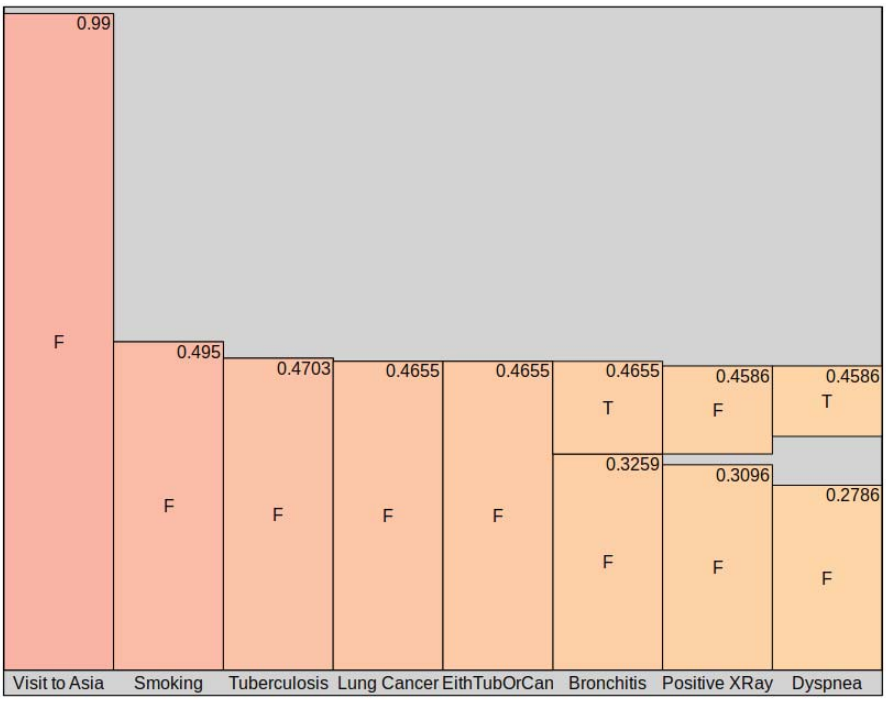


Fig. 4.5: Realización encontrada para $f_j = 0{,}44$

3.2. Puesto que conocemos f_j y u_i, siendo los f_j equidistantes podemos saber para cuantos j se cumple u_i ¿f_j, a ese valor δ, y nos permite no aumentar la j y tomar la misma realización de nuevo. Este δ es el contador de cuantos f_j aparecen en la realización actual, solo queda ajustar δ para que refleje la probabilidad real de esa realización, dada la evidencia. El resultado será una fila de un `data.frame`

3.3. Haciendo $j < -j + \delta$ y partiendo de la realización obtenida, repetir el paso 3.1 hasta que $j > N$

El resultado es un `data.frame` con frecuencias de realizaciones, a ser usado como el el algoritmo anterior. Veamos entonces los resultados de simular Asia con el muestreo estratificado:

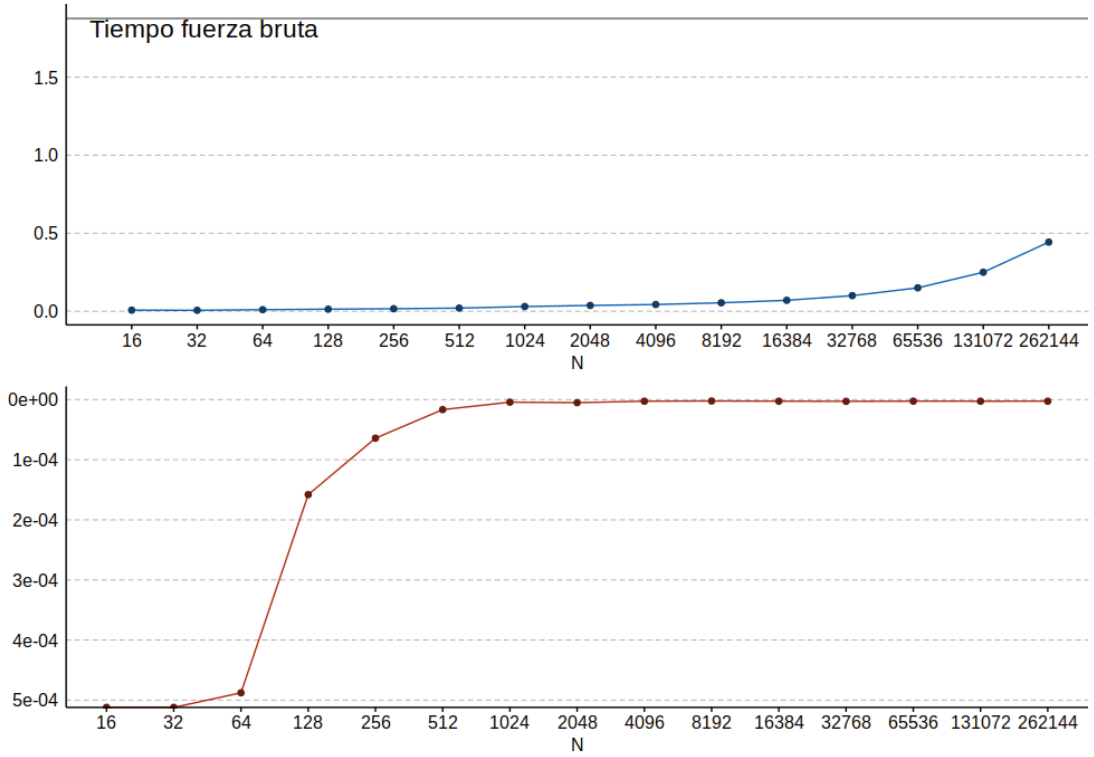


Fig. 4.6: Tiempo (s) y divergencia Kullback-Leibler para N tomadas cada vez

Como se puede observar, el tiempo de ejecución del algoritmo stratificado es considerablemente menor, pero esto también se debe al bajo número de nodos de Asia y a la baja eficiencia del método de fuerza bruta.

Capítulo 5

CONCLUSIONES

Índice de Ilustraciones

2.1 La red Asia. 4

3.1 Subred de Asia, importada desde un programa en *idr-prj* 12
3.2 Red aleatoria con N = 15, M = 2 17
3.3 Intervalo de la primera realización 18
3.4 Realización encontrada para $f_j = 0{,}44$ 19
3.5 Una ilustración de una red bayesiana aleatoria 20
3.6 Página de ayuda de R de la función de *bnidr* `node` 22

4.1 Tiempo requerido para redes aleatorias de N nodos y hasta 2 predecesores por nodo. 8 Redes para cada N 23
4.2 La red Asia, marginalizada. 25
4.3 Tiempo (s) y divergencia Kullback-Leibler para N tomadas cada vez 28
4.4 Intervalo de la primera realización 29
4.5 Realización encontrada para $f_j = 0{,}44$ 30
4.6 Tiempo (s) y divergencia Kullback-Leibler para N tomadas cada vez 31

Índice de Tablas

2.1 Distribuciones de probabilidad condicional de la red Asia 5

3.1 Funciones Básicas del paquete 14

4.1 Estimaciones de $P(Dyspnea = T)$ 27

Bibliografía

[1] E. Castillo et al (1997). *Sistemas Expertos y Modelos de Redes Probabilísticas*. Springer-Verlag, New York.

[2] G. F. Cooper (1990). The computational complexity of probabilistic inference using belief networks. *Artificial Intelligence*, **42**, 393405.

[3] R. G. Cowell et al (1999). *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York.

[4] L. Devroye (1986). *Non-Uniform Random Variate Generations*. Springer Verlag, New York.

[5] J. H. Kim and J. Pearl (1983). A Computational Model for Combined Causal and Diagnostic Reasoning in Inference Systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*. Morgan Kaufmann Publishers, San Mateo, CA, 190193.

[6] S. Kullback and R. A. Leibler (1951). On information and sufficiency. *Annals of Mathematical Statistics*. 22 (1): 7986. 10.1214/aoms/1177729694.

[7] J. Pearl (1985). Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning (http://ftp.cs.ucla.edu/tech-report/198_-reports/850017.pdf) (UCLA Technical Report CSD-850017). Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA. pp. 329334. Retrieved 2009-05-01.

[8] B. D. Ripley (1987). *Stochastic Simulation*. John Wiley and Sons, New York.

[9] R. Y. Rubinstein (1981). *Simulation and the Monte Carlo Method*. John Wiley and Sons, New York.

[10] Von Neumann (1951). Various Techniques Used in Connection with Random Numbers. *U.S. Nat. Bur. Stand. Appl. Math. Ser.*, 12:3638.

[11] https://www.r-project.org/

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Tue Jul 10 12:12:17 CEST 2018
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)