

---

# Nuevos algoritmos de inferencia en redes de creencia

Andrés Cano, Manuel Gómez, Serafín Moral

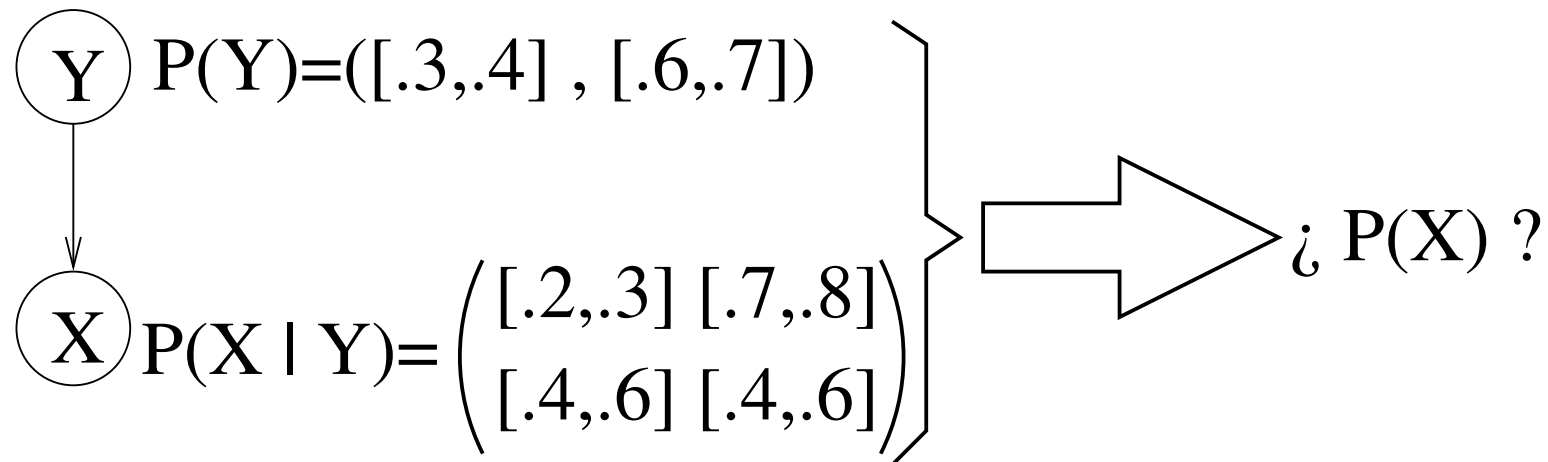
Departamento de Ciencias de la Computación

Universidad de Granada

# El problema

## Objetivo principal

- Usar **intervalos de probabilidad** o bien conjuntos de creencia (*credal sets*) en las redes de dependencia (**credal networks**).
- Desarrollar **algoritmos de propagación** para obtener información a posteriori en credal networks.

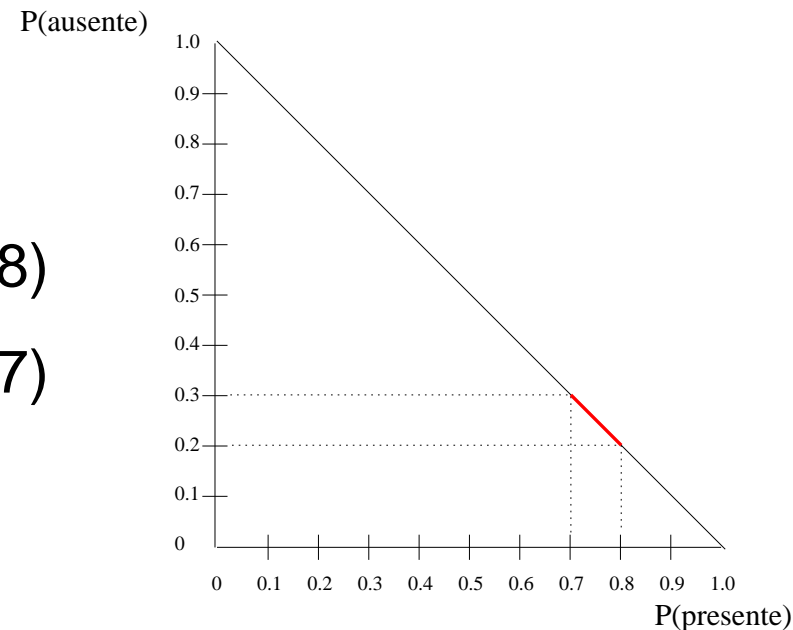


# Conjunto de creencia - Intervalos

Ejemplo de conjunto de creencia asociado a una distribución dada por intervalos

$$P(Cancer) = \begin{cases} [0.2, 0.3] & \text{si Cancer = ausente} \\ [0.7, 0.8] & \text{si Cancer = presente} \end{cases}$$

$$Vertices(P(Cancer)) = \begin{cases} (0.2, 0.8) \\ (0.3, 0.7) \end{cases}$$

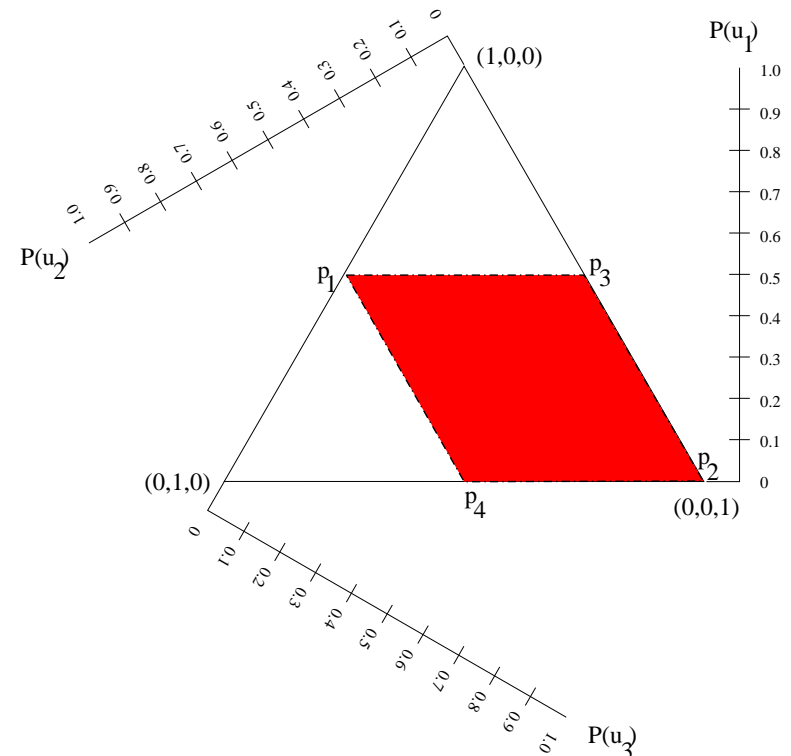


# Conjuntos de creencia - Intervalos

## Otro ejemplo

$$P(X) = \begin{cases} [0.0,0.5] & \text{si } X = x_1 \\ [0.0,0.5] & \text{si } X = x_2 \\ [0.0,1.0] & \text{si } X = x_3 \end{cases}$$

$$\text{Vertices}(P(X)) = \begin{cases} p_1 = (0.5, 0.5, 0.0) \\ p_2 = (0.0, 0.0, 1.0) \\ p_3 = (0.5, 0.0, 0.5) \\ p_4 = (0.0, 0.5, 0.5) \end{cases}$$



# Conjuntos de creencia condicionales

- Un *conjunto de creencia condicional*  $H^{X|Y}$  sobre  $X$  dado un conjunto de variables  $Y$  es un conjunto convexo y cerrado de distribuciones

$$p : X \times Y \longrightarrow [0, 1]$$

que verifican

$$\sum_{x_i \in \Omega_X} p(x_i, y_j) = 1, \forall y_j \in \Omega_Y$$

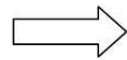
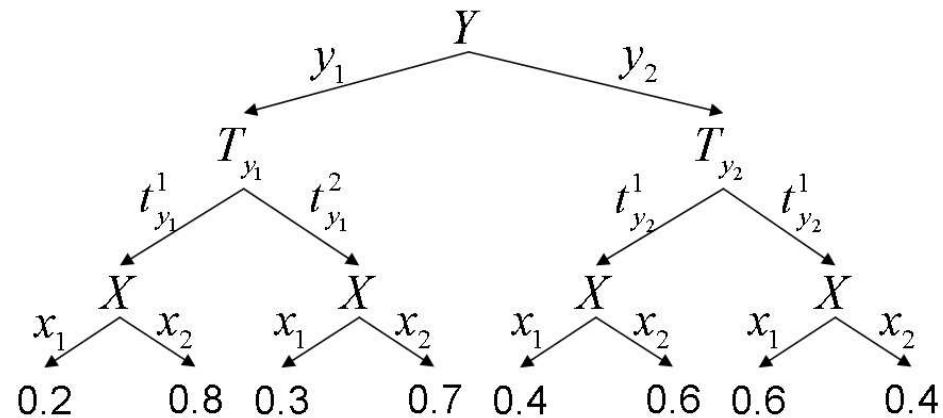
- Aquí trabajaremos con un conjunto de creencia  $H^{X|Y=y_j}$  para cada configuración  $y_j$  de  $Y$  (*separately specified credal sets*).

# Conjuntos de creencia condicionales

$$P(X|Y) = \begin{cases} & \begin{array}{c|cc} & y_1 & y_2 \\ \hline x_1 & [0.2,0.3] & [0.4,0.6] \\ x_2 & [0.7,0.8] & [0.4,0.6] \end{array} \end{cases}$$

$$H^{X|Y=y_1} = \begin{array}{c|cc} & x_1 & x_2 \\ \hline p_1 & 0.2 & 0.8 \\ p_2 & 0.3 & 0.7 \end{array}$$

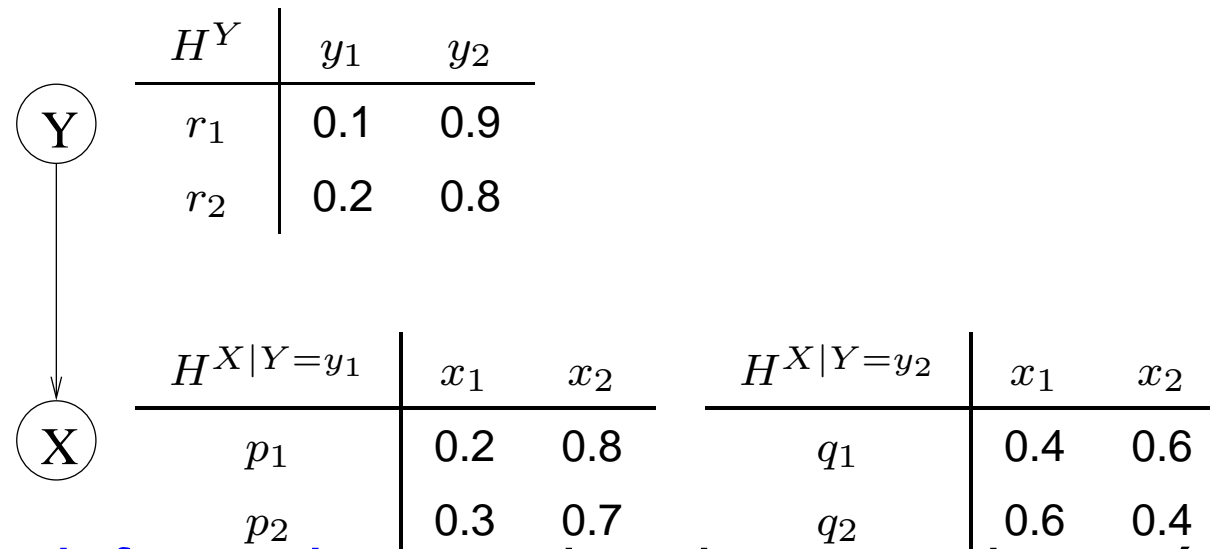
$$H^{X|Y=y_2} = \begin{array}{c|cc} & x_1 & x_2 \\ \hline q_1 & 0.4 & 0.6 \\ q_2 & 0.6 & 0.4 \end{array}$$



$$H^{X|Y} = \begin{array}{c|cccc} & (x_1, y_1) & (x_2, y_1) & (x_1, y_2) & (x_2, y_2) \\ \hline r_1 & 0.2 & 0.8 & 0.4 & 0.6 \\ r_2 & 0.2 & 0.8 & 0.6 & 0.4 \\ r_3 & 0.3 & 0.7 & 0.4 & 0.6 \\ r_4 & 0.3 & 0.7 & 0.6 & 0.4 \end{array}$$

# Inferencia en Credal networks

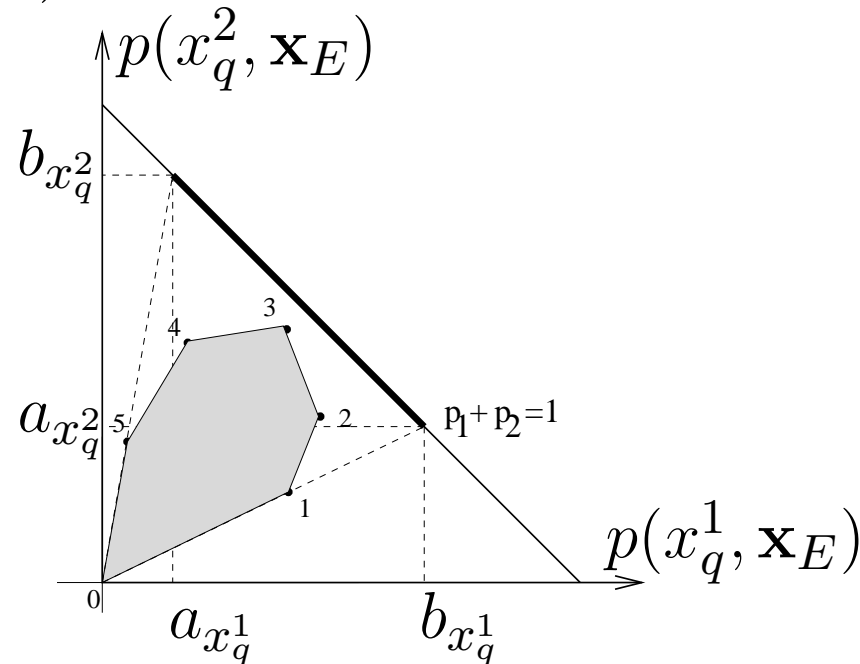
- Una red de creencia (*credal network*) es un DAG en el que se asocia un conjunto de creencia condicional  $H^{X|Y}$  (definido mediante un conjunto de conjuntos convexos especificados separadamente).



- El objetivo de la *inferencia* en redes de creencia será obtener los intervalos a posteriori de los valores de una variable  $X_q$  ( $\forall x_q^i \in \Omega_{X_q}$ ) dado un conjunto de variables observadas  $X_E$ .

# Inferencia en Credal networks

- Aplicando un **algoritmo de propagación exacto** obtendríamos un conjunto convexo de distribuciones de  $\Omega_{X_q}$  en  $[0, 1]$ , que llamaremos  $R_q$ .
- Por ejemplo, si  $X_q$  puede tomar dos posibles valores  $x_q^1$  y  $x_q^2$ , entonces  $R_q$  estaría formado por una serie de vértices  $(p_1, p_2)$  ( $p_i = p(x_q^i, \mathbf{x}_E)$ , siendo  $\mathbf{x}_E$  la evidencia dada).

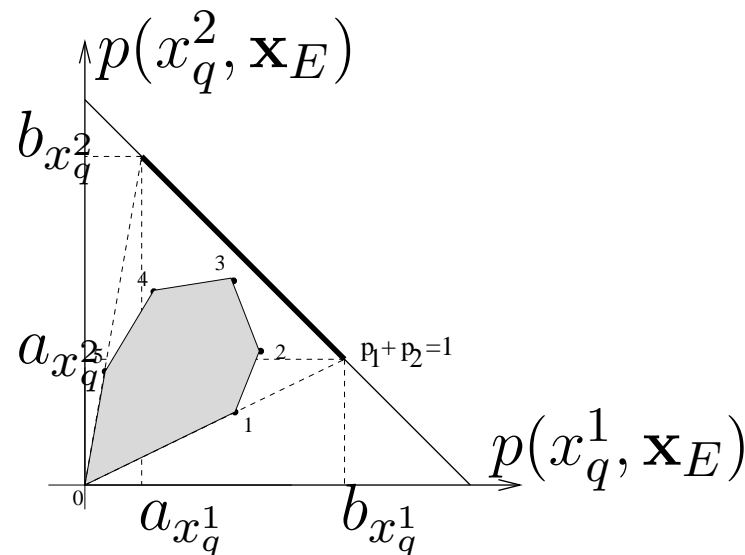


# Inferencia en Credal networks

- Normalizando cada uno de los vértices de  $R_q$  podremos calcular el mínimo y máximo de la probabilidad a posteriori para cada  $x_q^i \in \Omega_{X_q}$ :

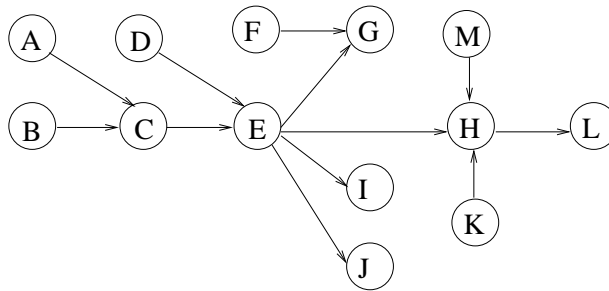
$$a = \min_i \{p_i / (p_1 + p_2) \mid (p_1, p_2) \in R_q\}$$

$$(1) \quad b = \max_i \{p_i / (p_1 + p_2) \mid (p_1, p_2) \in R_q\}$$



# Inferencia en Credal networks

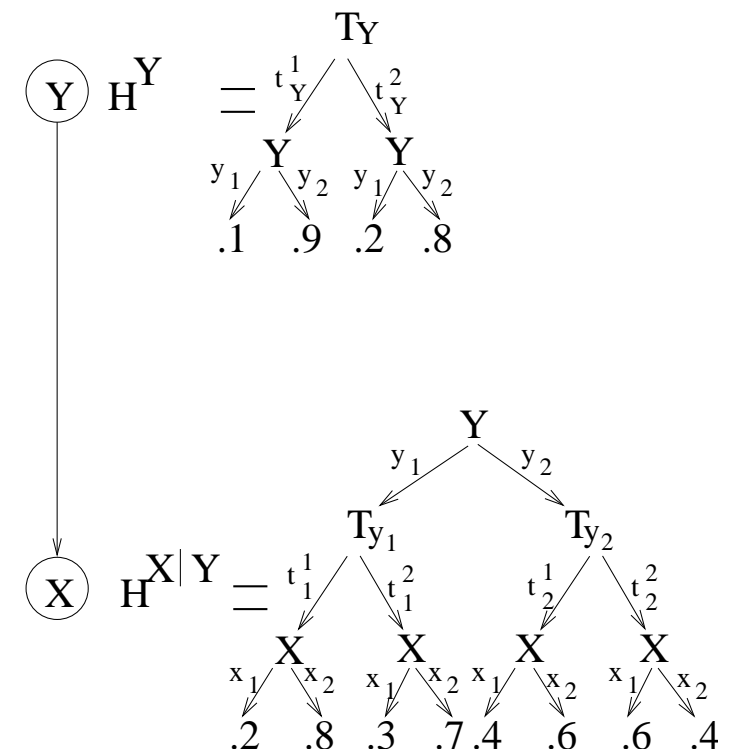
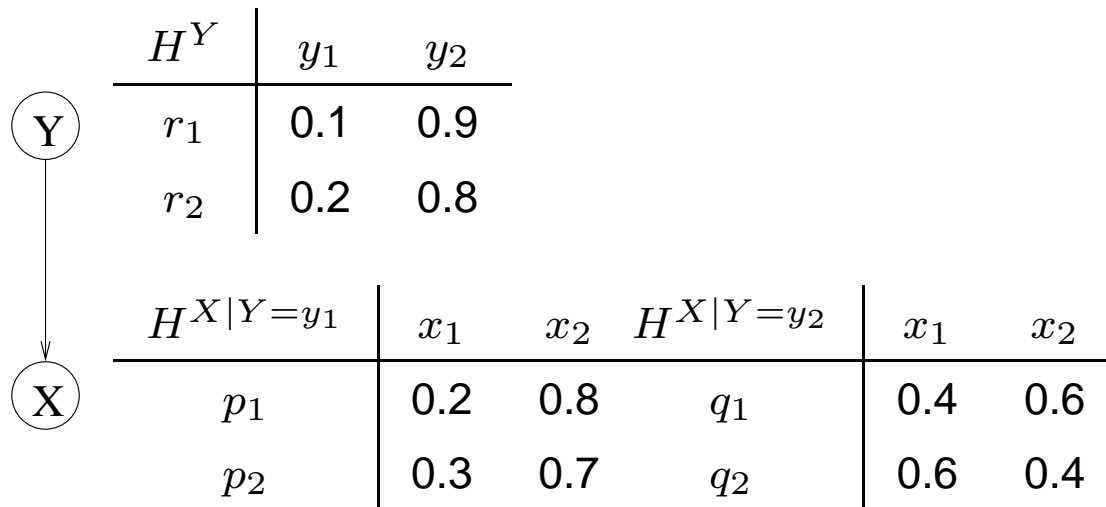
- El tiempo requerido para la inferencia exacta en redes de creencia crece exponencialmente con el número de estados de las variables ( $ns$ ) y con el número de vértices ( $nv$ ) de cada conjunto convexo especificado separadamente.
- Ejemplo: calcular  $P(E)$  en



$ns$	$nv$	$nprop$	Red
2	2	$2^{11}$	BN2s2ext
2	3	$3^{11}$	BN2s3ext
3, 2 en C	2	$2^{18}$	BNMixed2ext
3	2	$2^{21}$	
3	3	$3^{21}$	

# Algoritmos de inferencia en Elvira

- En los algoritmos implementados en Elvira, se trabaja con un árbol de probabilidad para cada variable de la red de creencia



# Algoritmo de ascensión de colinas

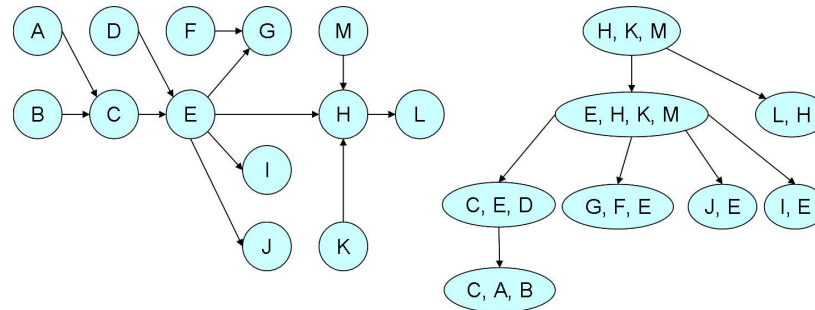
---

- Es un algoritmo aproximado para el problema de obtener el máximo (o bien mínimo) de la probabilidad a posteriori para un determinado  $x_q^i$  de la variable  $X_q$ .
- Este algoritmo puede aplicarse a redes con cualquier estructura (no sólo a poliárboles).
- En nuestro caso se traduce en encontrar la configuración de variables transparentes que produce la máxima probabilidad a posteriori para el caso  $x_q^i$ :

$$(2) \quad \max_{\mathbf{t}_s} p_{\mathbf{t}_s} (X_q = x_q^i | \mathbf{x}_E)$$

# Algoritmo de ascensión de colinas

- El algoritmo se basa en el de propagación en un *join tree* de Shenoy-Shafer para redes bayesianas.



- **Paso 1:** Los **pasos iniciales** de este algoritmo son:
  - Construir un *join tree* a partir de la red de creencia.
  - Asociar una valuación  $\Psi_{C_i}$  a cada clique  $C_i$ .
  - Asociar las valuaciones de las distribuciones condicionales a uno de los cliques que contengan todas sus variables.
  - Calcular  $\Psi_{C_i}$  como la combinación de todas la valuaciones asociadas a  $C_i$

# Algoritmo de ascensión de colinas

- Denotemos como  $\mathbf{x}_E$  al conjunto de evidencia.
- Una diferencia es que se utiliza un *doble sistema de mensajes*:

Para cada par de cliques  $C_i$  y  $C_j$ , hay dos mensajes desde  $C_i$  hacia  $C_j$ ,  $M_{C_i \rightarrow C_j}^1$  y  $M_{C_i \rightarrow C_j}^2$ , y dos mensajes desde  $C_j$  hacia  $C_i$ ,  $M_{C_j \rightarrow C_i}^1$  y  $M_{C_j \rightarrow C_i}^2$ .

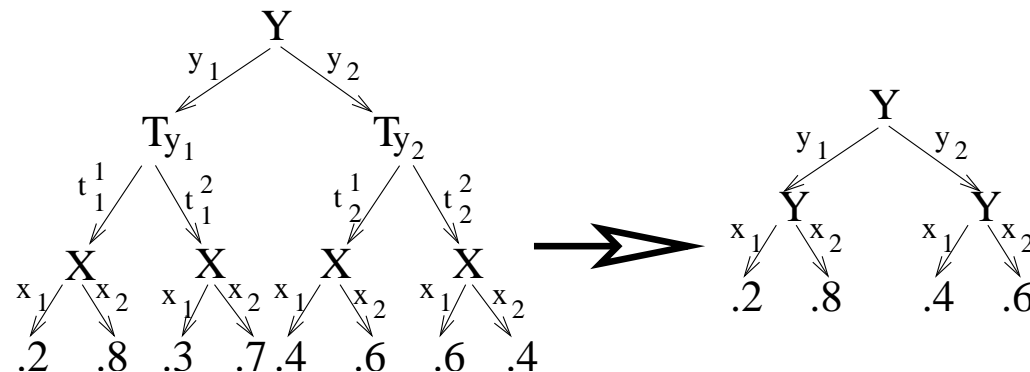
- Los mensajes  $M_{C_i \rightarrow C_j}^1$  se calculan de la forma usual:

$$(3) \quad M_{C_i \rightarrow C_j} = \left( \Psi_{C_i} \otimes \left( \bigotimes_{C_k \in \text{Ady}(C_i, C_j)} M_{C_k \rightarrow C_i} \right) \right) \downarrow_{C_i \cap C_j}$$

- Los mensajes  $M_{C_i \rightarrow C_j}^2$  se calculan también con la fórmula anterior pero suponemos que la observación  $X_q = x_q^i$  se añade al conjunto evidencia  $\mathbf{x}_E$ .

# Algoritmo de ascensión de colinas

- En cada momento el algoritmo tiene asociado una configuración  $t$  para todas las variables transparentes.
  - **Paso 2:** Se elige una configuración aleatoria inicial  $t_0$ .
  - **Paso 3:**  $t_0$  se añade al conjunto de evidencia  $x_E$ .
  - **Paso 4:** La evidencia  $x_E$  es introducida en el *join tree* mediante la operación de *restricción* en los árboles de probabilidad de cada una de la valuaciones  $\Psi_{C_i}$ .
- Ejemplo  $t = \{T_{y_1} = t_1^1, T_{y_2} = t_2^1\}$



# Algoritmo de ascensión de colinas

---

- Tras el último paso ningún  $\Psi_{C_i}$  contiene variables transparentes: el join tree define entonces una única distribución de probabilidad conjunta.
- **Paso 5: Propagación inicial** desde los cliques hoja hacia el raíz, calculando los dos sistemas de mensajes.
- **Paso 6: Fase de optimización**: Se recorre el join tree desde el clique raíz hacia las hojas y viceversa  $m$  veces o bien hasta que el algoritmo no mejore la solución.
- Cada vez que se visita un clique  $C_i$  intentamos mejorar (maximizar)  $p(X_q = x_q^i | \mathbf{x}_E)$  tomando una a una las transparentes del clique y eligiendo el estado de la transparente que mayor valor dé para  $p(X_q = x_q^i | \mathbf{x}_E)$ .

# Algoritmo de ascensión de colinas

- Sea  $t$  la configuración actual de variables transparentes .
- Sea  $T_{y_j}$  la transparente que vamos a optimizar.
- Calculamos  $p(\mathbf{x}_E)$  y  $p(x_q^i, \mathbf{x}_E)$  para cada transparente  $T_{y_j}$ , en la distribución de probabilidad determinada por  $t$ .
  - Descartar de  $\mathbf{x}_E$  la observación actual para  $T_{y_j}$ .
  - Los valores de  $p(\mathbf{x}_E)$  para cada  $t_{y_j}^i \in \Omega_{T_{y_j}}$  con:

$$(4) \quad p(T_{y_j}, \mathbf{x}_E) = (\Psi_{C_i} \otimes (\bigotimes_{C_j \in \text{Ady}(C_i)} M_{C_j \rightarrow C_i}^1))^{\downarrow T_{y_j}}$$

- Los valores de  $p(x_q^i, \mathbf{x}_E)$  para cada  $t_{y_j}^i \in \Omega_{T_{y_j}}$  con:

$$(5) \quad p(T_{y_j}, x_q^i, \mathbf{x}_E) = (\Psi_{C_i} \otimes (\bigotimes_{C_j \in \text{Ady}(C_i)} M_{C_j \rightarrow C_i}^2))^{\downarrow T_{y_j}}$$

# Algoritmo de ascensión de colinas

---

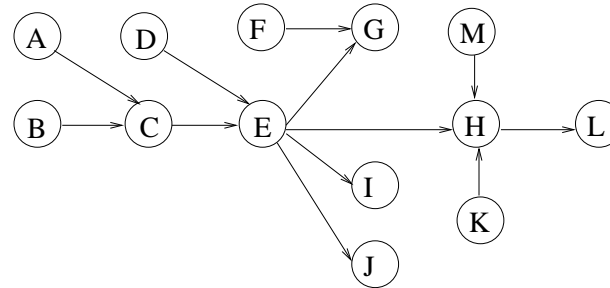
- El paso anterior obtiene un valor de  $p(\mathbf{x}_E)$  y  $p(x_q^i, \mathbf{x}_E)$  para cada  $t_{\mathbf{y}_j}^i \in \Omega_{T_{\mathbf{y}_j}}$ .
- Estos valores nos permiten seleccionar el estado de  $T_{\mathbf{y}_j}$  que maximiza  $p(x_q^i | \mathbf{x}_E)$

$$(6) \quad p(x_q^i | \mathbf{x}_E) = \frac{p(x_q^i, \mathbf{x}_E)}{p(\mathbf{x}_E)}$$

- El estado  $t_{\mathbf{y}_j}^i \in \Omega_{T_{\mathbf{y}_j}}$  que maximiza  $p(x_q^i | \mathbf{x}_E)$  se añade de nuevo a  $\mathbf{x}_E$

# Algoritmo de ascensión de colinas: Experimento

- Hemos generado aleatoriamente 30 redes de cada tipo



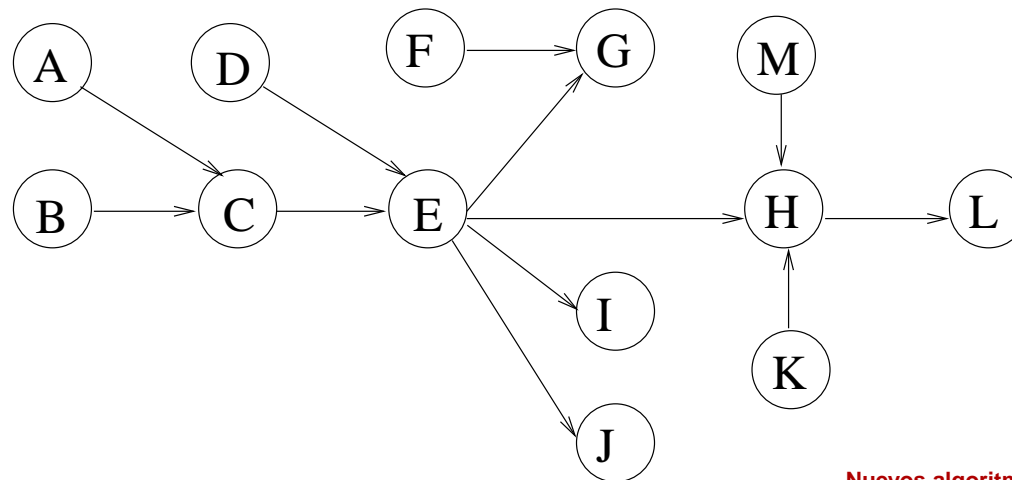
- *Error cuadrático medio* con sólo 3 recorridos en el join tree

	BN2s2ext	BN2s3ext	BNMixed2ext
mín $p(x_q^1   \mathbf{x}_E)$	0.0031	0.043	0.0094
máx $p(x_q^1   \mathbf{x}_E)$	0.027	0.097	0.019
mín $p(x_q^2   \mathbf{x}_E)$	0.027	0.096	0.0092
máx $p(x_q^2   \mathbf{x}_E)$	0.065	0.043	0.0011
mín $p(x_q^3   \mathbf{x}_E)$	-	-	0.0046
máx $p(x_q^3   \mathbf{x}_E)$	-	-	0.029

- El tiempo medio de propagación ha sido 0.015, 0.017 y 0.16 segundos

# Algoritmo Branch-and-bound

- Este algoritmo es exacto y también puede aplicarse a redes con cualquier estructura.
- El algoritmo también está orientado a obtener el máximo o bien mínimo de  $p(x_q^i | \mathbf{x}_E)$ .
- Supongamos de nuevo que buscamos  $\max p(x_q^i | \mathbf{x}_E)$
- El algoritmo tendrá como entrada la red  $\mathcal{N}_0$  obtenida de  $\mathcal{N}$  (red inicial) descartando las variables que no son necesarias por d-separación.



# Algoritmo Branch-and-bound

---

- **Bound:** La técnica branch-and-bound requiere de un procedimiento  $r$  para obtener una *aproximación*  $f$  por arriba (sobreestimación) de la solución del problema  $P$ .
- **Branch:** El algoritmo es como una **búsqueda en un árbol**:
  - El nodo raíz contiene  $P$ .
  - Los nodos descendientes contienen sub-instancias de  $P$ .
  - Los nodos hoja contienen problemas que se pueden resolver de forma exacta.
  - Si el valor  $f$  calculado en un **nodo hoja**  $l$  es mayor que el más grande  $\hat{p}$  por ahora entonces  $\hat{p} = f$
  - En los **nodos internos** se aplica el procedimiento  $r$  para obtener una sobreestimación de la solución: si  $f < \hat{p}$  se detiene la búsqueda por esta rama.

# Particularización del Branch-and-bound

**Input** : A credal network  $\mathcal{N}_0$

**Output**: The value of  $\bar{p} = \max p(x_q^i | \mathbf{x}_E)$

Initialize  $\hat{p}$  with  $-\infty$

**if** the credal net  $\mathcal{N}_0$  contains a single vertex **then**

    | Update  $\hat{p}$  with  $p(x_q^i | \mathbf{x}_E)$  if  $p(x_q^i | \mathbf{x}_E) > \hat{p}$

**end**

**else**

    Using the  $k$  possibilities of one of the credal sets in  $\mathcal{N}_0$ , obtain a list of  $k$  credal networks  $\{\mathcal{N}_{01}, \dots, \mathcal{N}_{0k}\}$  from  $\mathcal{N}_0$

**foreach**  $\mathcal{N}_{0h}$  **do**

        | **if**  $r(\mathcal{N}_{0h}) > \hat{p}$  **then**

            | Call recursively depth-first branch-and-bound over  $\mathcal{N}_{0h}$

        | **end**

**end**

**end**

Take the last  $\hat{p}$  as  $\bar{p}$

# Particularización del Branch-and-bound

- En nuestro caso el nodo raíz del árbol de búsqueda contiene la red  $\mathcal{N}_0$ .
- El nodo raíz se divide en redes más simples, seleccionando una transparente  $T_{y_j}$  de  $\mathcal{N}_0$  y obteniendo tantas redes  $\{\mathcal{N}_{01}, \dots, \mathcal{N}_{0k}\}$  como casos tenga  $T_{y_j}$ , aplicando la operación de **restricción** donde aparezca  $T_{y_j}$ .
- Las redes  $\{\mathcal{N}_{01}, \dots, \mathcal{N}_{0k}\}$  se insertan como hijas del raíz en el árbol de búsqueda.
- El procedimiento de descomposición se aplica recursivamente.
- En los nodos hoja se tiene una **red bayesiana** en la que aplicamos *algoritmo de eliminación de variables* para obtener  $p(x_q^i | \mathbf{x}_E)$  (si  $p(x_q^i | \mathbf{x}_E) > \hat{p}$ )

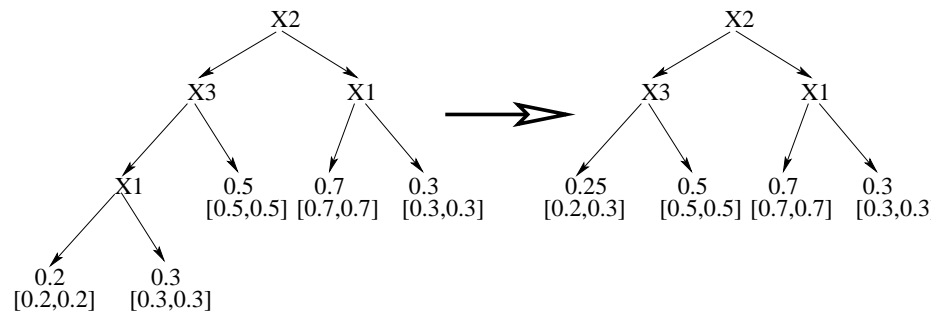
# Particularización del Branch-and-bound

---

- En los nodos internos las redes contienen variables transparentes y por tanto algunas distribuciones están definidas con *conjuntos de creencia*:
  - Aplicamos una **versión especial** del algoritmo de eliminación de variables (VE) para obtener una cota superior  $r$  de  $\max p(x_q^i | \mathbf{x}_E)$ .
  - Si  $r \leq \hat{p}$  detenemos la búsqueda por esta rama.

# Particularización del Branch-and-bound

- La **version especial del VE** usa árboles de probabilidad con dos valores adicionales en los nodos hoja,  $r_{\text{mín}}$  y  $r_{\text{máx}}$ .



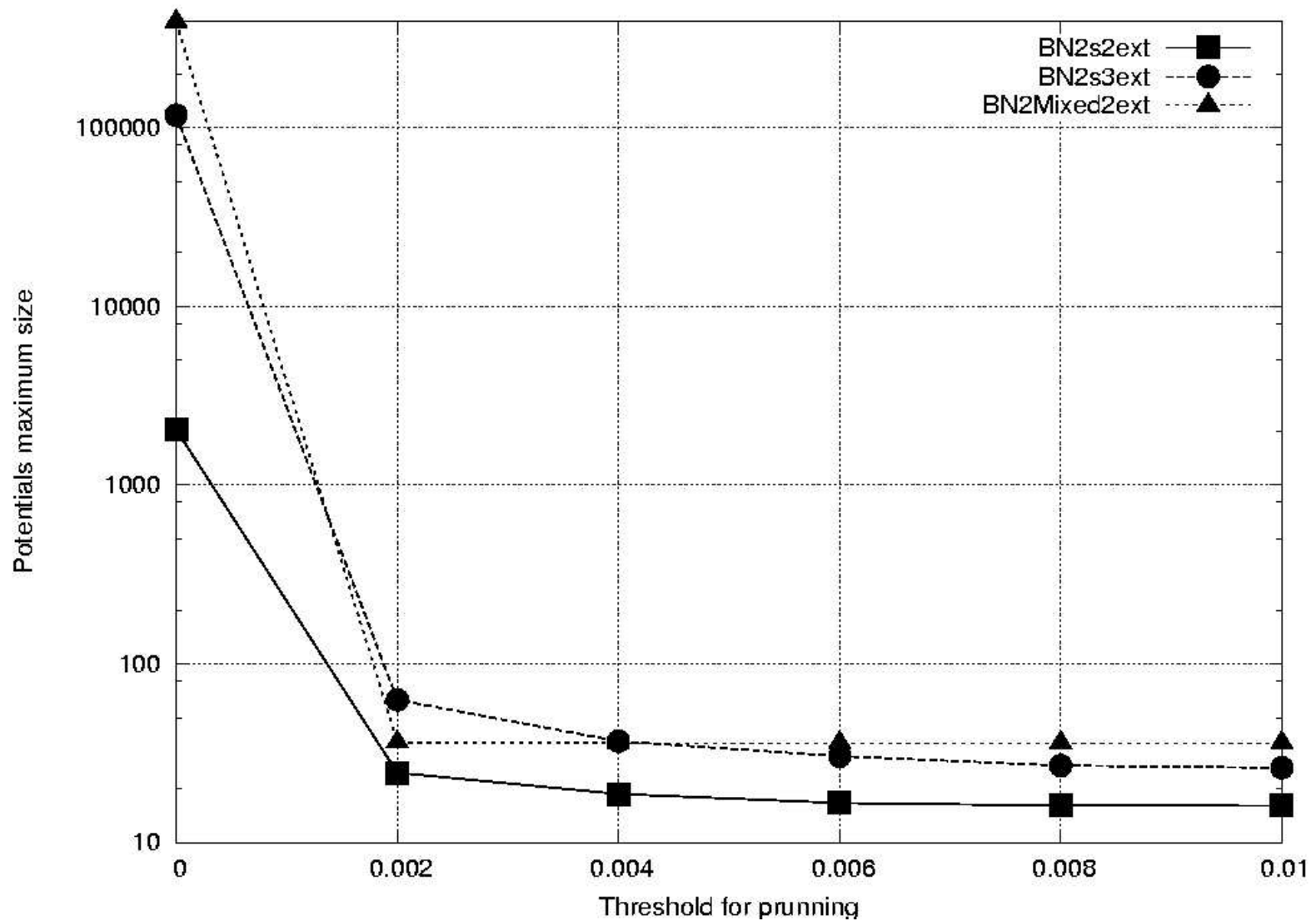
- En la propagación VE, los árboles son podados usando métodos basados en **distancia de Kullback-Leibler** después de combinación y marginalización para que los potenciales no crezcan demasiado haciendo imposible la propagación.
- El resultado del VE es un árbol de probabilidad con la variable objetivo y valores  $r_{\text{mín}}$  y  $r_{\text{máx}}$  en los nodos hoja.
- Los  $r_{\text{mín}}$  y  $r_{\text{máx}}$  permiten obtener cotas superiores para  $\max p(x_q^i | \mathbf{x}_E)$ .

# Branch-and-bound: Experimentos

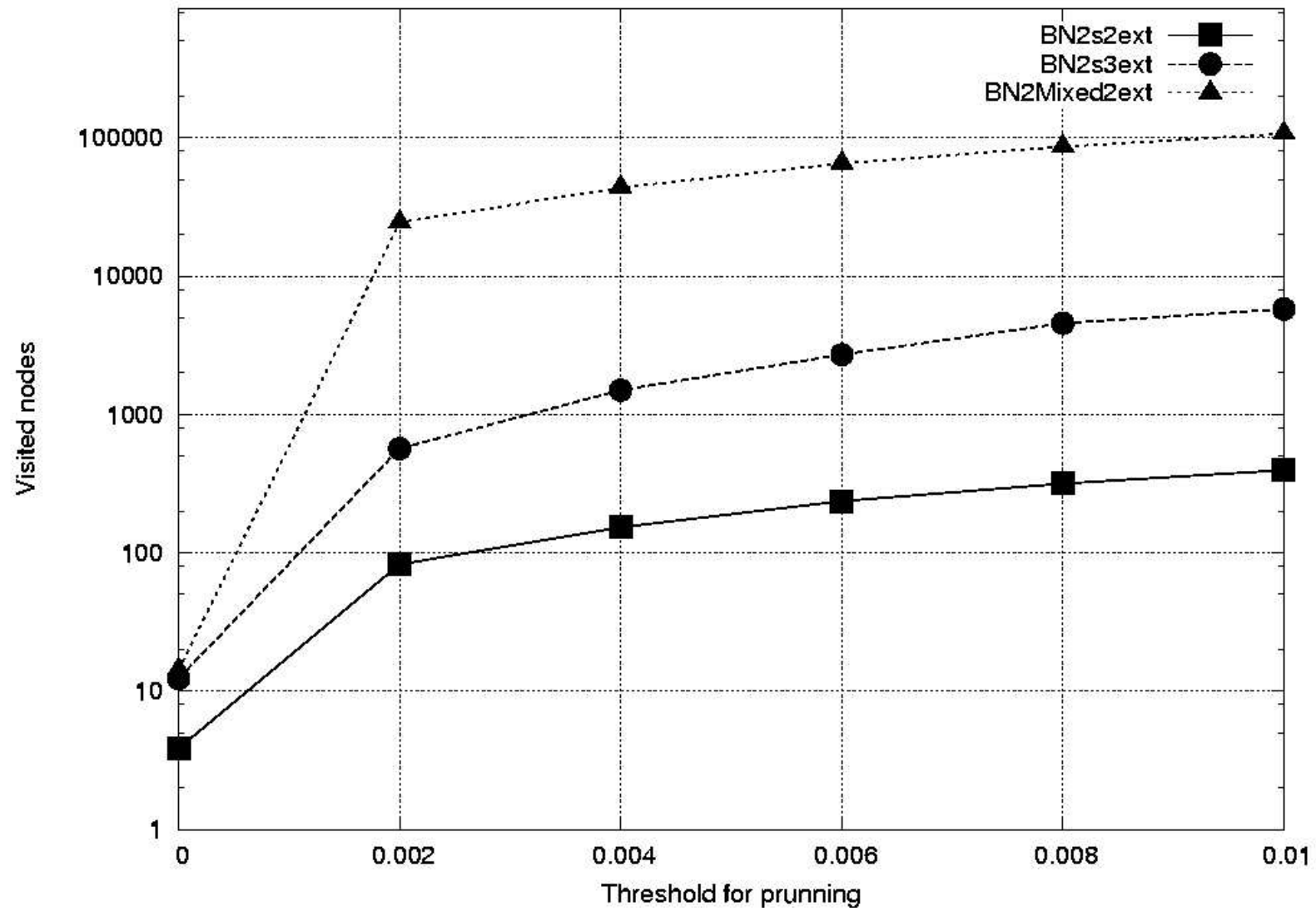
---

- Usamos de nuevo las 30 redes aleatorias de cada tipo que usamos en la ascensión de colinas.
- En primer lugar hemos ejecutado las pruebas usando los resultados del algoritmo de ascensión de colinas como **solución inicial** en el proceso de búsqueda.
- En el algoritmo VE con valores  $r_{\text{mín}}$  y  $r_{\text{máx}}$  el parámetro que controla la poda es  $\sigma$ . Hemos probado con valores desde 0.0 (ninguna poda) hasta 0.004 (poda grande).
  - Un valor pequeño para  $\sigma$  requiere de un número pequeño de nodos visitados en el árbol de búsqueda, pero necesita de árboles de probabilidad muy grandes.
  - Un valor alto de  $\sigma$  requiere un gran número de nodos visitados y árboles de probabilidad pequeños.

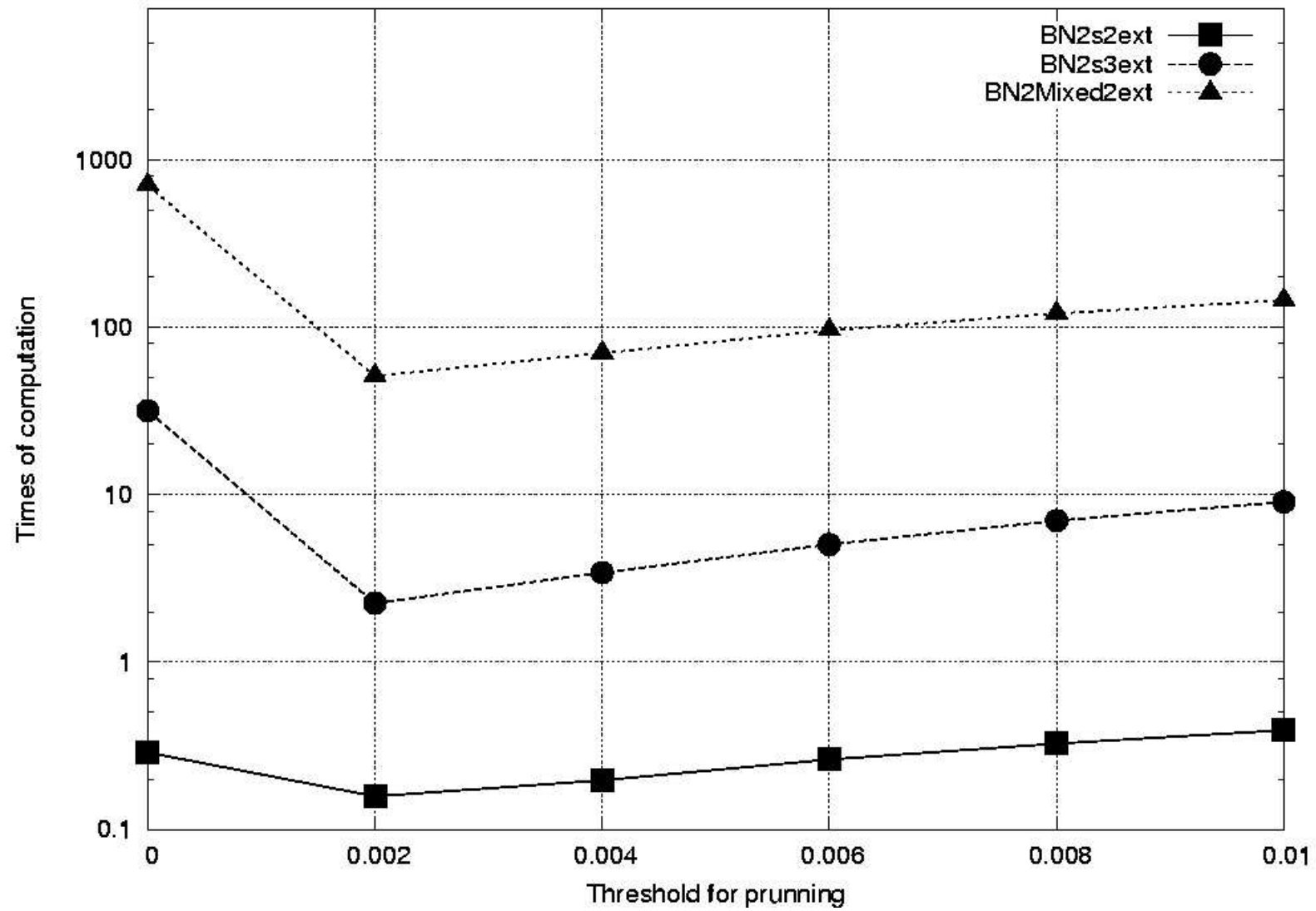
# Branch-and-bound: Experimentos



# Branch-and-bound: Experimentos



# Branch-and-bound: Experimentos



# Branch-and-bound: Experimentos

- Cuando no usamos el algoritmo de ascensión de colinas para dar la solución en el branch-and-bound se obtienen peores resultados:

	BN2s2ext	BN2s3ext	BNMixed2ext
max. pot. size	2.87 %	0.0063 %	0.001 %
visited nodes	38.91 %	51.22 %	41.34 %
comput. time	235.76 %	161.38 %	80.59 %

# Conclusiones

---

- Hemos dado dos nuevos algoritmos para inferir en redes de creencia: uno aproximado (de ascensión de colinas) y otro exacto (branch-and-bound).
- Ambos se pueden aplicar a cualquier estructura de red.
- El de ascensión de colinas da buenos resultados en poco tiempo.
- El de ascensión de colinas puede usarse con branch-and-bound para encontrar la solución inicial consiguiendo así mejores resultados (tiempo, tamaño de potenciales, número de nodos visitados).
- En el branch-and-bound el parámetro  $\sigma$  nos permite controlar el número de nodos visitados y el tamaño de los árboles de probabilidad.