

# Optimización de ASV para árboles de decisión

Director:  
Santiago Cifuentes

Co-director:  
Sergio Abriola

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

23 de junio de 2025



UNIVERSIDAD  
DE BUENOS AIRES

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

## Dificultad tesis

Vamos a utilizar pelotas de voley para medir la dificultad de las diapositivas.

- 1 pelota de voley: Family friendly 
- 2 pelotas de voley: Prestando atención se llega 
- 3 pelotas de voley: Con grafos y ganas alcanza 
- <4 pelotas de voley: Esta es para los jurados 

## 1 Introducción

### Introducción a Shapley y ASV

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

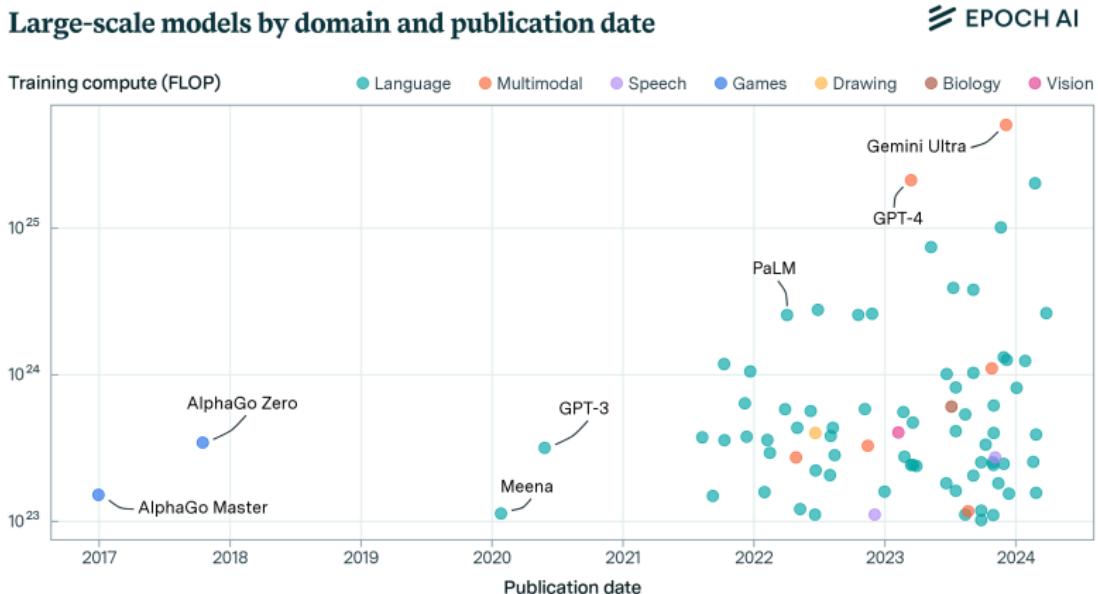
## 7 Conclusión

## 8 Extra

## Introducción a la XAI



- La capacidad de los modelos de IA ha crecido exponencialmente.



Introducción a la XAI

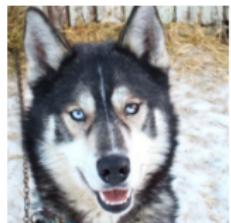


- La capacidad de los modelos de IA ha crecido exponencialmente.
  - Esto aumenta su complejidad y dificulta su comprensión.

## Introducción a la XAI



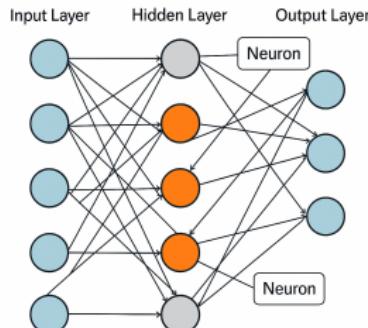
- La capacidad de los modelos de IA ha crecido exponencialmente.
  - Esto aumenta su complejidad y dificulta su comprensión.
  - La XAI (Explainable Artificial Intelligence) busca explicar decisiones y predicciones de los modelos.



(a) Husky classified as wolf



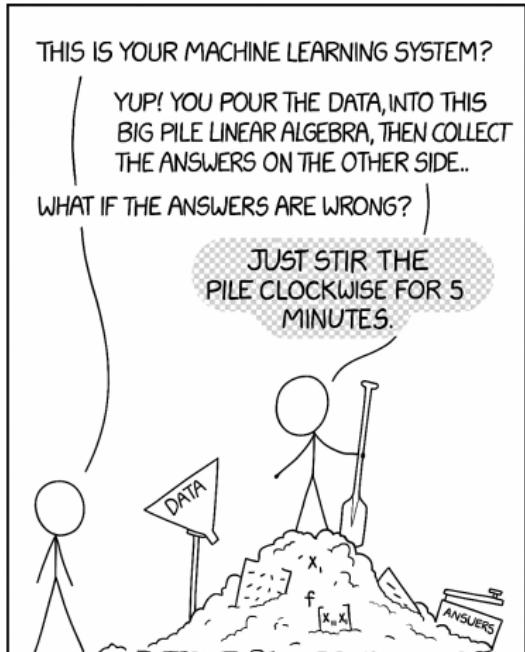
(b) Explanation



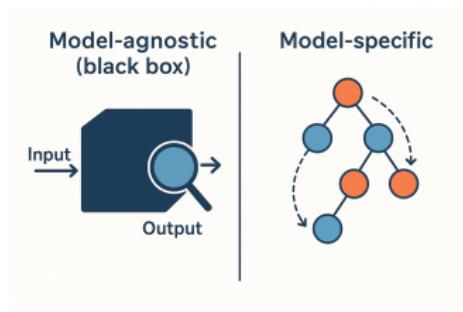
## Before and after XAI



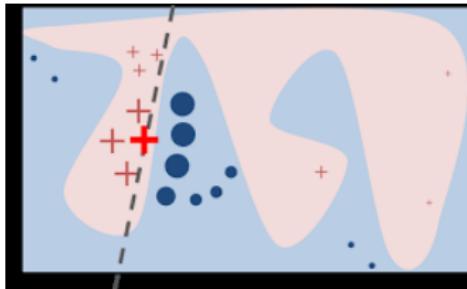
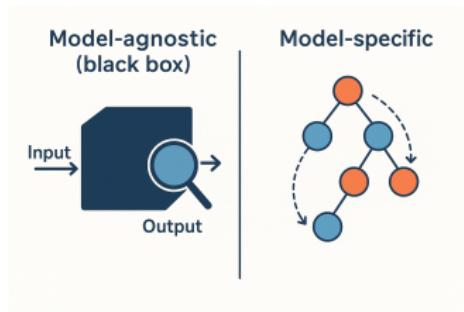
## Before and after XAI



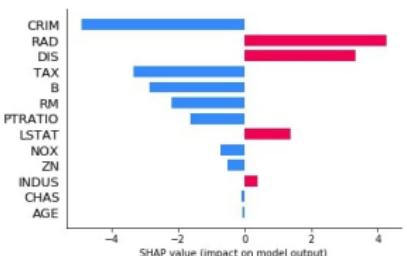
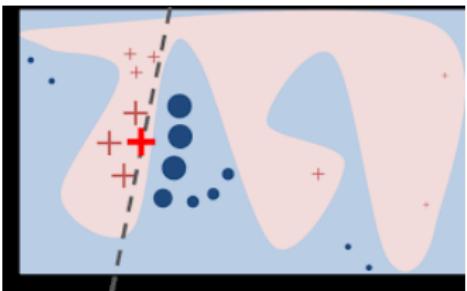
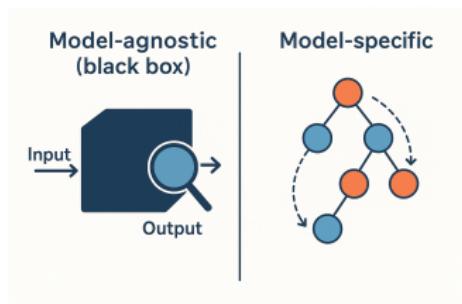
## Tipos de métodos en XAI



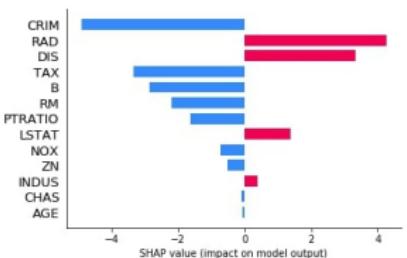
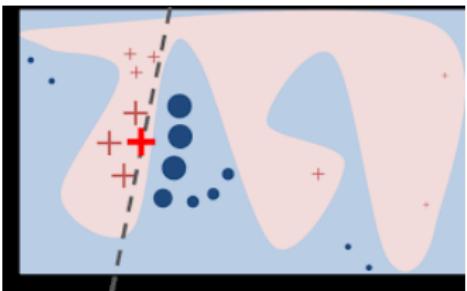
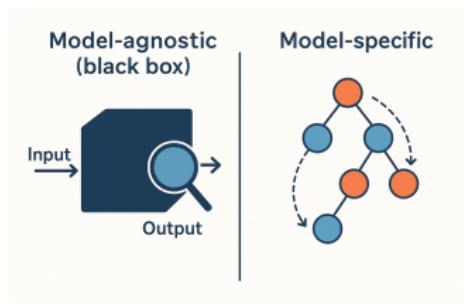
## Tipos de métodos en XAI



## Tipos de métodos en XAI



## Tipos de métodos en XAI



*No existe una definición universal de “explicación”. Las técnicas actuales son aproximaciones pragmáticas.*

# SHAP: Un método de feature attribution



- SHAP se basa en los valores de Shapley de teoría de juegos, los cuales veremos más adelante.

# SHAP: Un método de feature attribution



- SHAP se basa en los valores de Shapley de teoría de juegos, los cuales veremos más adelante.
- **Características:**

# SHAP: Un método de feature attribution



- SHAP se basa en los valores de Shapley de teoría de juegos, los cuales veremos más adelante.
- Características:
  - Agnóstico al modelo.

# SHAP: Un método de feature attribution



- SHAP se basa en los valores de Shapley de teoría de juegos, los cuáles veremos más adelante.
- Características:
  - Agnóstico al modelo.
  - Alcance local (aunque puede ser global)

# SHAP: Un método de feature attribution



- SHAP se basa en los valores de Shapley de teoría de juegos, los cuales veremos más adelante.
- Características:
  - Agnóstico al modelo.
  - Alcance local (aunque puede ser global)
  - Feature attribution.

# ASV vs SHAP: Incorporando Causalidad



- **ASV (Asymmetric Shapley Values)** incorpora la estructura causal de los datos al calcular la relevancia de los features.

# ASV vs SHAP: Incorporando Causalidad



- **ASV (Asymmetric Shapley Values)** incorpora la estructura causal de los datos al calcular la relevancia de los features.
- Veamos un ejemplo de cómo ASV incorpora esta información.

# ASV vs SHAP: Incorporando Causalidad



- **ASV (Asymmetric Shapley Values)** incorpora la estructura causal de los datos al calcular la relevancia de los features.
- Veamos un ejemplo de cómo ASV incorpora esta información.
  - Modelo: Red neuronal que predice si el sueldo anual de una persona es mayor o menor a 50.000 dolares.

# ASV vs SHAP: Incorporando Causalidad



- **ASV (Asymmetric Shapley Values)** incorpora la estructura causal de los datos al calcular la relevancia de los features.
- Veamos un ejemplo de cómo ASV incorpora esta información.
  - Modelo: Red neuronal que predice si el sueldo anual de una persona es mayor o menor a 50.000 dolares.
  - Datos de entrenamiento: *Census Income* dataset del UC Irvine.

# ASV vs SHAP: Incorporando Causalidad



- **ASV (Asymmetric Shapley Values)** incorpora la estructura causal de los datos al calcular la relevancia de los features.
- Veamos un ejemplo de cómo ASV incorpora esta información.
  - Modelo: Red neuronal que predice si el sueldo anual de una persona es mayor o menor a 50.000 dolares.
  - Datos de entrenamiento: *Census Income* dataset del UC Irvine.
  - Métricas a evaluar: **ASV y SHAP globales**.

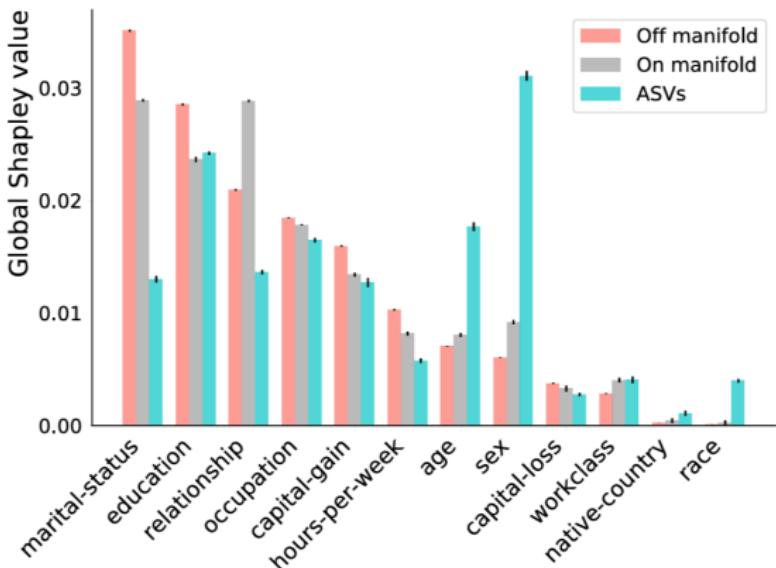
# Dígrafo causal del Census Income Dataset



Dígrafo causal: aristas de  $A = \{age, sex, native\ country, race\}$  hacia  $D = \{marital, education, relation, occupation, capital - gain, hours, capital - loss, workclass\}$



## Resultados ASV vs SHAP



## Comparación de los valores globales de SHAP y ASV para el Census Income Dataset.

# Objetivo de la tesis



- Problema de ASV: Es costoso calcularlo, pero a diferencia de SHAP, para algunos modelos que SHAP es difícil, ASV es tratable.

# Objetivo de la tesis



- Problema de ASV: Es costoso calcularlo, pero a diferencia de SHAP, para algunos modelos que SHAP es difícil, ASV es tratable.
- Objetivo inicial de la tesis: Calcular ASV en *tiempo polinomial* para una familia de *digrafos causales*

# Objetivo de la tesis



- Problema de ASV: Es costoso calcularlo, pero a diferencia de SHAP, para algunos modelos que SHAP es difícil, ASV es tratable.
- Objetivo inicial de la tesis: Calcular ASV en *tiempo polinomial* para una familia de *digrafos causales*
- Todos los algoritmos mencionados los pueden encontrar en:

---

Repo <https://github.com/EchuCompa/pasantia-BICC>

---

## 1 Introducción

### Introducción a Shapley y ASV

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

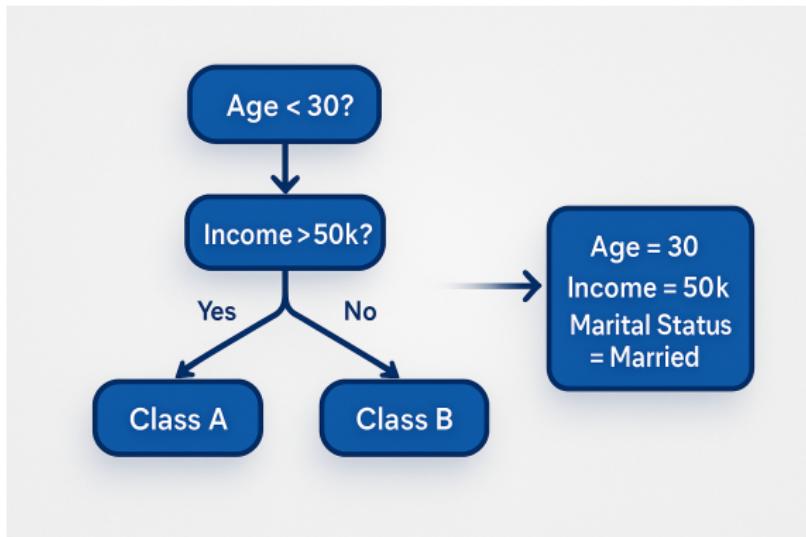
## 7 Conclusión

## 8 Extra

## Definiciones



- Sean  $X$  los features y  $ent(x)$  el conjunto de entidades.
  - Luego  $M$  es el modelo y  $\phi$  un *feature attribution score*



# Shapley Values



**SHAP** se basa en los *Shapley Values* de teoría de juegos cooperativos. Podemos interpretar a este valor  $\phi_i(\nu)$  como el aporte del jugador  $i$  al juego definido por la función  $\nu$ .

# Asado Familiar : Subconjunto original



# Asado Familiar : Subconjunto óptimo



## Propiedades Shapley Values

Las propiedades de los *Shapley Values* son:

- **Eficiencia:** Toda la ganancia es distribuida
- **Simetría:** Jugadores equivalentes reciben el mismo valor.
- **Linealidad:** El valor de Shapley es lineal respecto a la función característica.
- **Jugador Nulo:** Si no aporta, su valor es 0.

# Fórmula Shapley



**Fórmula general:**

Definición (Shapley Value)

$$\phi_i(\nu) = \text{Shapley Value del jugador } i \text{ para la función } \nu$$

# Fórmula Shapley



**Fórmula general:**

Definición (Shapley Value)

$$\phi_i(\nu) = \text{Shapley Value del jugador } i \text{ para la función } \nu$$

Definición

*La función característica se define como:*

$$\nu : \mathcal{P}(X) \rightarrow \mathbb{R}$$

*Asigna un real a cada posible coalición de jugadores, es decir, a cada subconjunto de  $X$ .*

# Fórmula Shapley



## Fórmula general:

Definición (Shapley Value)

$$\phi_i(\nu) = \nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i})$$

- Se calcula cuánto colabora  $i$  a  $\pi$ :  $\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i})$ .

# Fórmula Shapley



## Fórmula general:

### Definición (Shapley Value)

$$\phi_i(\nu) = \sum_{\pi \in perm(X)} \nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i})$$

- Se calcula cuánto colabora  $i$  a  $\pi$ :  $\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i})$ .
- Se suman todas las permutaciones de  $X$ , para ver cuánto colabora  $i$  a cada una.

# Fórmula Shapley



## Fórmula general:

### Definición (Shapley Value)

$$\phi_i(\nu) = \frac{1}{|X|!} \sum_{\pi \in perm(X)} (\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i}))$$

- Se calcula cuánto colabora  $i$  a  $\pi$ :  $\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i})$ .
- Se suman todas las permutaciones de  $X$ , para ver cuánto colabora  $i$  a cada una.
- Se divide todo por  $|X|!$ , porque se está promediando sobre todas las permutaciones posibles.

# SHAP en Aprendizaje Automático



- Features  $X$  son los jugadores, y  $\nu_{M,e,Pr}(S)$  es la predicción promedio.
- Así la fórmula de SHAP nos queda:

# SHAP en Aprendizaje Automático



- Features  $X$  son los jugadores, y  $\nu_{M,e,Pr}(S)$  es la predicción promedio.
- Así la fórmula de SHAP nos queda:

## Definición (SHAP)

Teniendo un modelo  $M$ , una entidad  $e$ , un feature  $x_i$  y una función de probabilidad  $Pr$  definimos Shap como:

$$Shap_{M,e,Pr}(x_i) = \sum_{S \subseteq X \setminus \{x_i\}} c_{|S|} (\nu(S \cup \{x_i\}) - \nu(S))$$

# Asymmetric Shapley Values (ASV)



- La definición clásica de Shapley values asigna el **mismo peso** a todas las permutaciones.

# Asymmetric Shapley Values (ASV)



- La definición clásica de Shapley values asigna el **mismo peso a todas las permutaciones**.
- En ASV se busca **asignarle más peso a ciertas permutaciones**.

# Asymmetric Shapley Values (ASV)



- La definición clásica de Shapley values asigna el **mismo peso a todas las permutaciones**.
- En ASV se busca **asignarle más peso a ciertas permutaciones**.
- Para capturar esta idea **relajamos la propiedad de la simetría**, definiendo una función de peso  $w$  sobre las permutaciones

# Definición formal de ASV

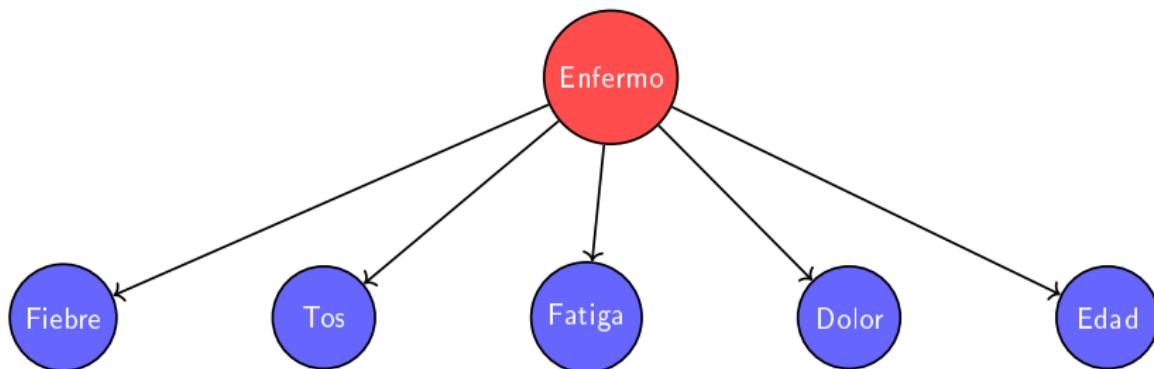


## Definición (Asymmetric Shapley Values)

Dada una función característica  $\nu : \mathcal{P}(\mathcal{I}) \rightarrow \mathbb{R}$  y un peso  $w$  sobre permutaciones, los ASV son

$$\phi_i^{\text{assym}}(\nu) = \sum_{\pi \in \text{perm}(\mathcal{I})} w(\pi) (\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i})).$$

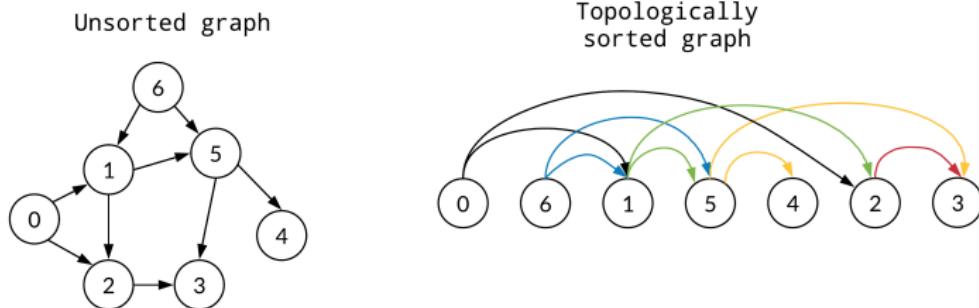
## Ejemplo : Naive Bayes como DAG causal



# Función de peso desde un DAG causal



Dado un DAG causal  $G = (X, E)$ . El conjunto de **órdenes topológicos** son las permutaciones que respetan al grafo causal.



## Función de peso desde un DAG causal



Así es como **cada orden recibe el mismo peso**, y cada permutación que **no es un orden tópologico no debe ser contabilizada**. De esta forma  $w$  nos queda:

$$w(\pi) = \begin{cases} \frac{1}{|\text{topos}(G)|}, & \pi \in \text{topos}(G), \\ 0, & \text{en otro caso.} \end{cases}$$

# ASV en aprendizaje automático



## Definición (ASV con DAG Causal)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}))$$

# ASV en aprendizaje automático



## Definición (ASV con DAG Causal)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}))$$

- Es la misma fórmula que SHAP, solo que tomamos las permutaciones que sean órdenes topológicos

# ASV en aprendizaje automático



## Definición (ASV con DAG Causal)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}))$$

- Es la misma fórmula que SHAP, solo que tomamos las permutaciones que sean órdenes topológicos
- Este enfoque prioriza las permutaciones causalmente válidas y *asigna más importancia a las causas que a las consecuencias.*

# Complejidad de ASV



Nuestro objetivo es estudiar la **tratabilidad** de ASV para los casos que SHAP no es tratable. Luego teniendo en cuenta que:

# Complejidad de ASV



Nuestro objetivo es estudiar la **tratabilidad** de ASV para los casos que SHAP no es tratable. Luego teniendo en cuenta que:

- El conjunto sobre el cual se itera en ASV es más reducido que el de SHAP

# Complejidad de ASV



Nuestro objetivo es estudiar la **tratabilidad** de ASV para los casos que SHAP no es tratable. Luego teniendo en cuenta que:

- El conjunto sobre el cual se itera en ASV es más reducido que el de SHAP
- La distribución Naive Bayes se asemeja a una producto

# Complejidad de ASV



Nuestro objetivo es estudiar la **tratabilidad** de ASV para los casos que SHAP no es tratable. Luego teniendo en cuenta que:

- El conjunto sobre el cual se itera en ASV es más reducido que el de SHAP
- La distribución Naive Bayes se asemeja a una producto

# Complejidad de ASV



Nuestro objetivo es estudiar la **tratabilidad** de ASV para los casos que SHAP no es tratable. Luego teniendo en cuenta que:

- El conjunto sobre el cual se itera en ASV es más reducido que el de SHAP
- La distribución Naive Bayes se asemeja a una producto

Llegamos a nuestro primer resultado:

## Teorema (Tratabilidad de ASV)

*ASV puede calcularse en tiempo polinomial bajo una distribución Naive Bayes para una familia de modelos  $\mathcal{F}$  si y solo si SHAP puede calcularse para  $\mathcal{F}$  bajo distribución producto.*

# Objetivo de la tesis



- A raíz del teorema anterior surge el objetivo inicial de la tesis, el cuál era realizar *el cálculo exacto de ASV en tiempo polinomial para una red bayesiana*.

# Objetivo de la tesis



- A raíz del teorema anterior surge el objetivo inicial de la tesis, el cuál era realizar *el cálculo exacto de ASV en tiempo polinomial para una red bayesiana*.
- Como vamos a necesitar evaluar a  $\nu$ , necesitamos un modelo el cual pueda calcular el promedio en tiempo polinomial. Por eso nos vamos a centrar en **árboles de decisión**.

# Objetivo de la tesis



- A raíz del teorema anterior surge el objetivo inicial de la tesis, el cuál era realizar *el cálculo exacto de ASV en tiempo polinomial para una red bayesiana*.
- Como vamos a necesitar evaluar a  $\nu$ , necesitamos un modelo el cual pueda calcular el promedio en tiempo polinomial. Por eso nos vamos a centrar en **árboles de decisión**.
- El cálculo del promedio va a depender de su distribución  $Pr$ , que en este caso es una **Red Bayesiana**.

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

# Redes Bayesianas: Definición y Semántica

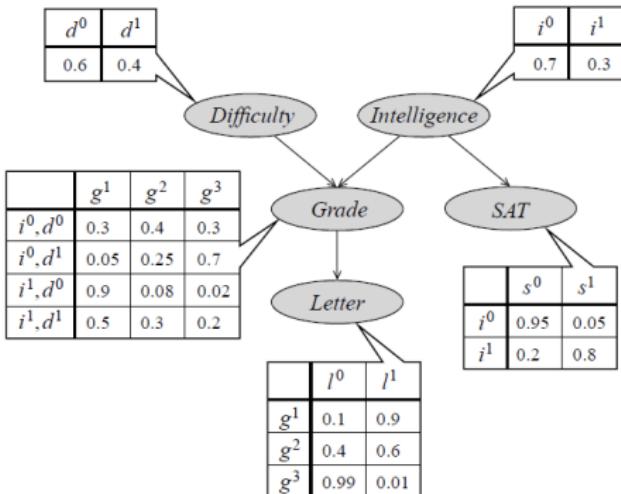


## Definición

Una red bayesiana  $N = (X, E, \Pr)$  consta de:

- Un DAG  $(X, E)$ : nodos son features, aristas son dependencias condicionales.
- Una función  $\Pr$  que codifica la probabilidad en base a esa red.

# Ejemplo: Red Bayesiana



Red bayesiana sobre datos de un estudiante con sus correspondientes Conditional Probability Tables (CPTs).

# Inferencia en Redes Bayesianas



- **Inferencia:** calcular  $Pr(C | E)$ , la probabilidad del evento  $C$  dada la evidencia  $E$ .

# Inferencia en Redes Bayesianas



- **Inferencia:** calcular  $Pr(C | E)$ , la probabilidad del evento  $C$  dada la evidencia  $E$ .
- El algoritmo que calcula la inferencia es *Variable Elimination*.

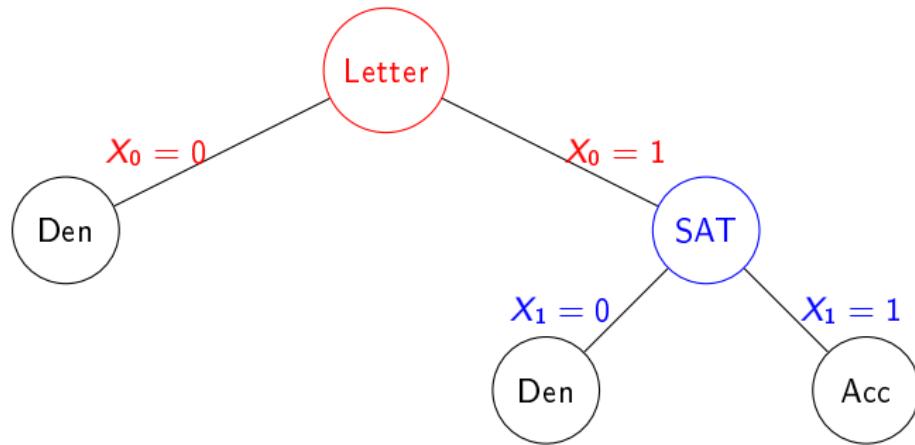
# Inferencia en Redes Bayesianas



- **Inferencia:** calcular  $Pr(C | E)$ , la probabilidad del evento  $C$  dada la evidencia  $E$ .
- El algoritmo que calcula la inferencia es *Variable Elimination*.
- En **polytrees** es **polinomial**. En general es **#P-hard**. Por lo tanto, vamos a utilizar redes bayesianas que sean **polytrees**.



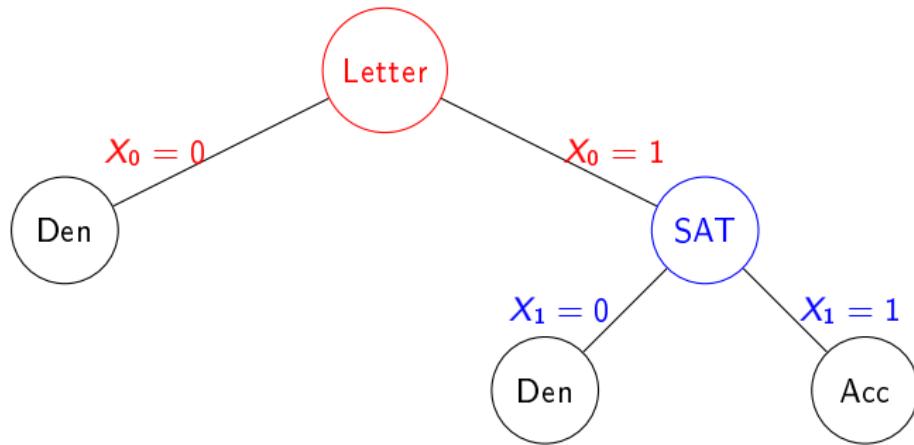
# Árboles de decisión



Árbol de decisión que define si un estudiante va a ser aceptado (accepted/1) o rechazado (denied/0) en su ingreso a una universidad.



# Árboles de decisión



Árbol de decisión que define si un estudiante va a ser aceptado (accepted/1) o rechazado (denied/0) en su ingreso a una universidad.

**Objetivo:** calcular la predicción promedio del árbol condicionada en la evidencia.

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

Algoritmo para las clases

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

# Heurística ASV: Recordando la Fórmula



Recordemos la definición de ASV:

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} \left( \nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}) \right).$$

- Nuestro objetivo es minimizar la cantidad de veces que se evalúa  $\nu$ .



# Heurística ASV: Recordando la Fórmula

Recordemos la definición de ASV:

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} \left( \nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}) \right).$$

- Nuestro objetivo es minimizar la cantidad de veces que se evalúa  $\nu$ .
- La idea de la heurística es identificar los órdenes topológicos que devuelven el mismo resultado al ser evaluados por  $\nu$ .

Podemos agruparlos según la relación  $R^*$ , en la cual dos órdenes topológicos  $\pi^1, \pi^2$  están relacionados si evalúan a lo mismo.



# Heurística ASV: Recordando la Fórmula

Recordemos la definición de ASV:

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} \left( \nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}) \right).$$

- Nuestro objetivo es minimizar la cantidad de veces que se evalúa  $\nu$ .
- La idea de la heurística es identificar los órdenes topológicos que devuelven el mismo resultado al ser evaluados por  $\nu$ .
- Así solo tendremos que evaluar  $\nu$  una vez por cada orden perteneciente a cada clase de equivalencia.

## Relación de Equivalencia $R$



Definimos  $R$  para distinguir clases de equivalencia  $[\pi]_R$  para el feature  $x_i$ :

$$\pi^1 R \pi^2 \iff \{\pi_{<i}^1\} = \{\pi_{<i}^2\}$$

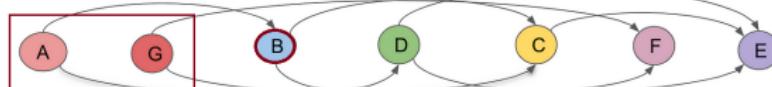
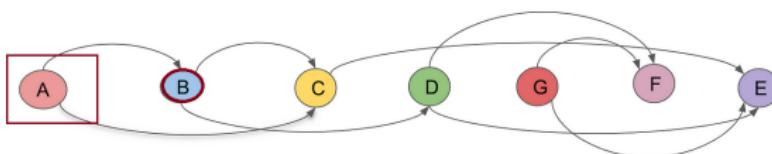
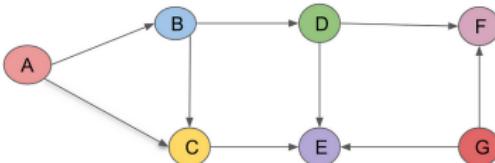
Ambos órdenes evalúan igual:  $\nu(\pi_{<i})$

## Relación de Equivalencia $R$

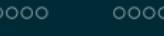
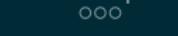


Definimos  $R$  para distinguir clases de equivalencia  $[\pi]_R$  para el feature  $x_i$ :

$$\pi^1 R \pi^2 \iff \{\pi_{\leq i}^1\} = \{\pi_{\leq i}^2\}$$



## Topological Sort

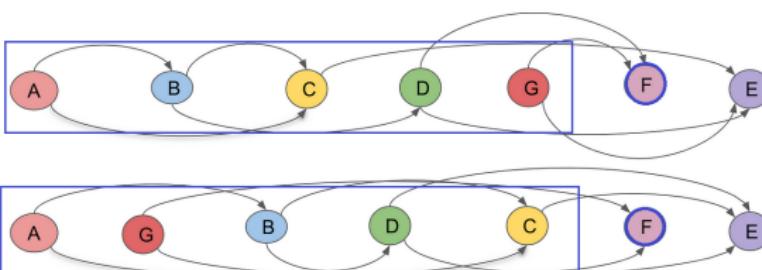
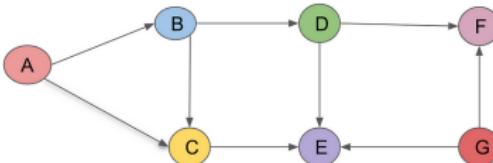


# Relación de Equivalencia $R$



Definimos  $R$  para distinguir clases de equivalencia  $[\pi]_R$  para el feature  $x_i$ :

$$\pi^1 R \pi^2 \iff \{\pi_{<i}^1\} = \{\pi_{<i}^2\}$$



Topological Sort

## Heurística ASV



Denotemos por

$$eqCI(G, x_i) = \{ [\pi]_R \mid \pi \in \text{topos}(G)\}$$

al conjunto de clases de equivalencia respecto de  $R$ .



## Heurística ASV

Denotemos por

$$eqCI(G, x_i) = \{ [\pi]_R \mid \pi \in \text{topos}(G) \}$$

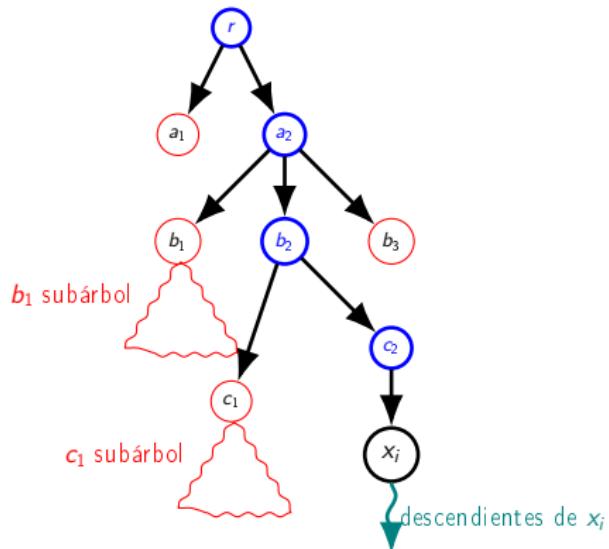
al conjunto de clases de equivalencia respecto de  $R$ . Entonces nuestra fórmula original nos queda:

$$\frac{1}{|\text{topos}(G)|} \sum_{[\pi]_R \in eqCI(G, x_i)} \left( \nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}) \right) \cdot |\pi|$$

en vez de:

$$\frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} \left( \nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i}) \right)$$

# Clases de Equivalencia en *d*trees: Ejemplo

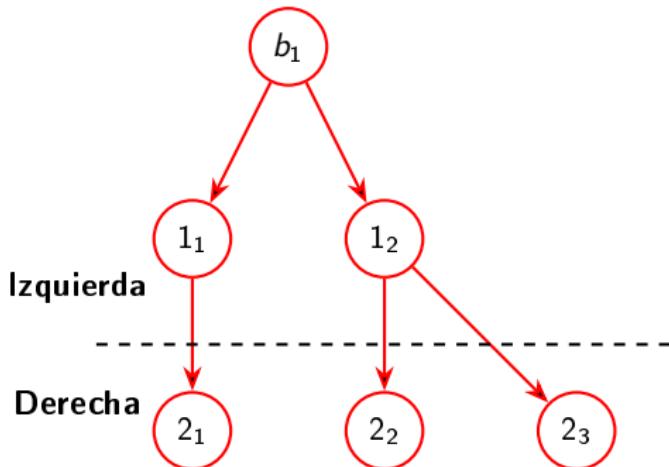


Ejemplo de un *d*tree con  $x_i$  en negro, sus nodos no relacionados marcados en rojo y sus ancestros marcados en azul.



## Subárbol de Nodos No Relacionados: Ejemplo

Tomemos el subárbol de  $b_1$  del ejemplo. Queremos ver las clases de equivalencia posibles dentro de dicho subárbol:



Clase del subárbol. En este caso la clase sería  
 $L([\pi]_R) = \{b_1, 1_1, 1_2\}$ ,  $R([\pi]_R) = \{2_1, 2_2, 2_3\}$ .

## Cota Superior para $\#EC(r)$ en *dtrees*



Queremos una cota teórica para el número de clases de equivalencia en un *dtree*.

### Lemma (Cota superior de clases de equivalencia)

Sea  $T$  un árbol con raíz  $r$ , número de hojas  $l$ , número de nodos  $n$  y altura  $h$  se cumple que:

$$\#EC(r) \leq h^l + 1.$$

## Cota Superior para $\#EC(r)$ en $dtrees$



Queremos una cota teórica para el número de clases de equivalencia en un *d-tree*.

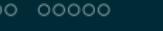
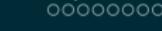
Lemma (Cota superior de clases de equivalencia)

Sea  $T$  un árbol con raíz  $r$ , número de hojas  $l$ , número de nodos  $n$  y altura  $h$  se cumple que:

$$\#EC(r) \leq h^l + 1.$$

## **Observaciones:**

- A menor número de hojas  $l$ , menor cota para  $\#EC(r)$ .



## Cota Superior para $\#EC(r)$ en $d$ trees



Queremos una cota teórica para el número de clases de equivalencia en un *dtree*.

Lemma (Cota superior de clases de equivalencia)

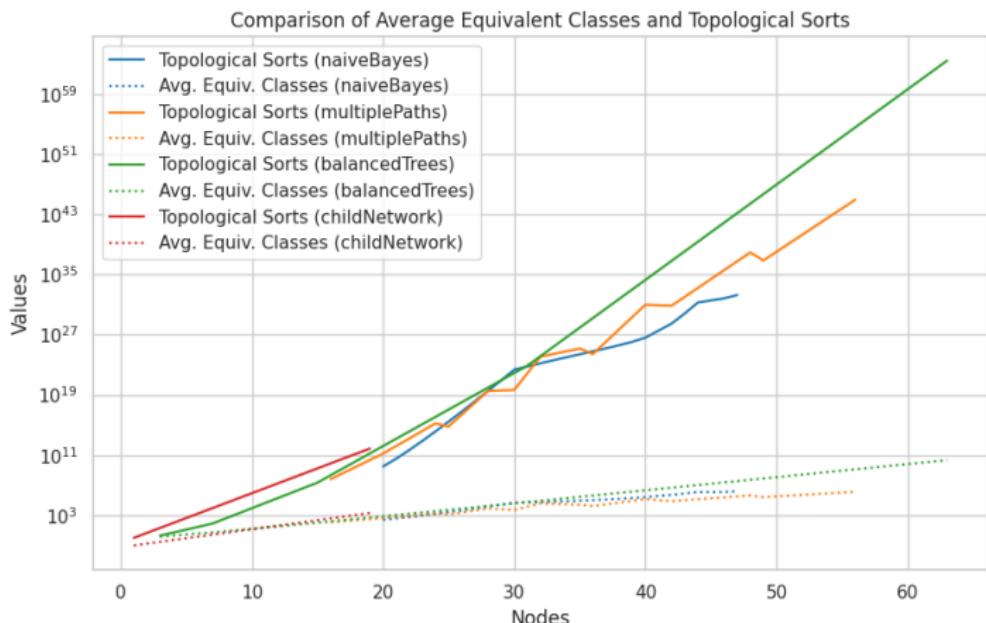
Sea  $T$  un árbol con raíz  $r$ , número de hojas  $l$ , número de nodos  $n$  y altura  $h$  se cumple que:

$$\#EC(r) \leq h^l + 1.$$

## **Observaciones:**

- A menor número de hojas  $l$ , menor cota para  $\#EC(r)$ .
  - La cota de las clases depende de  $h$  y  $l$ , mientras que la de los órdenes topológicos depende de  $n$  ( $O(n!)$ ).

Spoiler : It works!



## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

### Algoritmo para las clases

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

## Cálculo de ASV



Ahora queremos calcular ASV utilizando nuestra heurística:

$$Shap_{M,e,\Pr}^{assym}(x_i) = \frac{1}{|topos(G)|} \sum_{[\pi]_R \in eqCI(G,x_i)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i})) * |[\pi]_R|$$

# Cálculo de ASV



Ahora queremos calcular ASV utilizando nuestra heurística:

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{[\pi]_R \in \text{eqCI}(G, x_i)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i})) * |[\pi]_R|$$

¿Cómo obtenemos las clases de equivalencia?

## Solución Naive



- Generar todos los órdenes topológicos  $orders = \text{topos}(G)$ .

# Solución Naive



- Generar todos los órdenes topológicos  $orders = \text{topos}(G)$ .
- Agrupar cada  $\pi \in orders$  según  $U_\pi =$ , sus nodos no relacionados antes de  $x_i$ .

# Solución Naive



- Generar todos los órdenes topológicos  $orders = \text{topos}(G)$ .
- Agrupar cada  $\pi \in orders$  según  $U_\pi =$ , sus nodos no relacionados antes de  $x_i$ .
- Para cada grupo  $U_\pi$ , guardar un representante y su tamaño.

# Solución Naive



- Generar todos los órdenes topológicos  $orders = \text{topos}(G)$ .
- Agrupar cada  $\pi \in orders$  según  $U_\pi =$ , sus nodos no relacionados antes de  $x_i$ .
- Para cada grupo  $U_\pi$ , guardar un representante y su tamaño.
- **Complejidad:**  $O(n!)$  en el peor caso.

# Algoritmo Recursivo



- Etiquetar  $G$  como:

# Algoritmo Recursivo



- Etiquetar  $G$  como:
  - ① Ancestros  $A$  de  $x_i$ .

# Algoritmo Recursivo



- Etiquetar  $G$  como:
  - ① Ancestros  $A$  de  $x_i$ .
  - ② Descendientes  $D$  de  $x_i$ .

# Algoritmo Recursivo



- Etiquetar  $G$  como:
  - ① Ancestros  $A$  de  $x_i$ .
  - ② Descendientes  $D$  de  $x_i$ .
  - ③ Árboles no relacionados (raíces  $UR$ ).

# Algoritmo Recursivo



- Etiquetar  $G$  como:
  - ① Ancestros  $A$  de  $x_i$ .
  - ② Descendientes  $D$  de  $x_i$ .
  - ③ Árboles no relacionados (raíces  $UR$ ).
- Paso 1: calcular las clases de equivalencia para todos los subárboles no relacionados.

# Algoritmo Recursivo

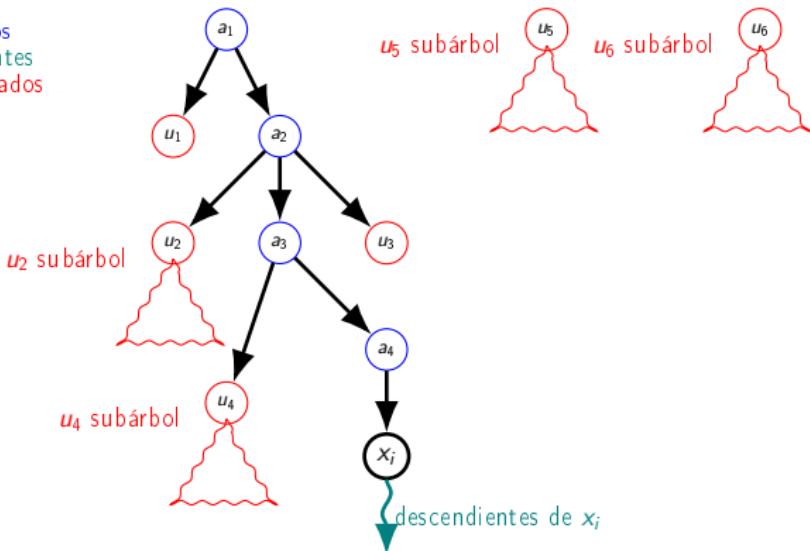


- Etiquetar  $G$  como:
  - ① Ancestros  $A$  de  $x_i$ .
  - ② Descendientes  $D$  de  $x_i$ .
  - ③ Árboles no relacionados (raíces  $UR$ ).
- Paso 1: calcular las clases de equivalencia para todos los subárboles no relacionados.
- Paso 2: fusionar los resultados obtenidos con  $A$  y  $D$ .

## Ejemplo: Grafo Etiquetado

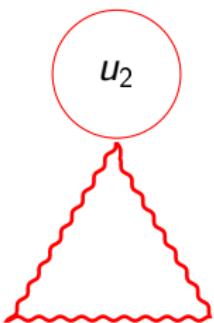


Ancestros  
Descendientes  
No relacionados



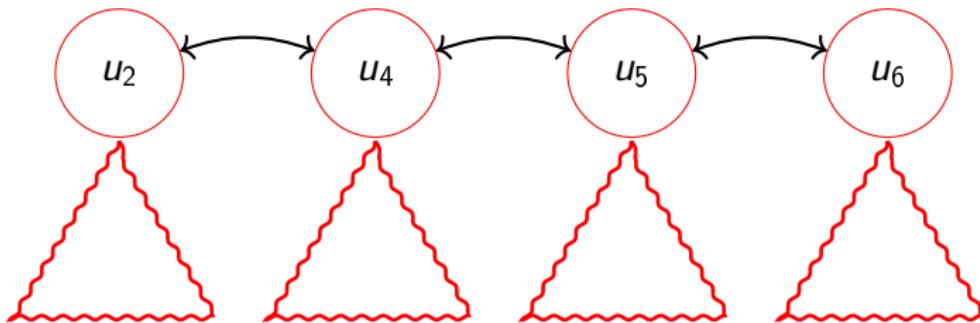
Digrafo etiquetado en base a  $x_i$ : Ancestros (azul), Descendientes (verde), No relacionados (rojo)

## Subárbol $u_2$



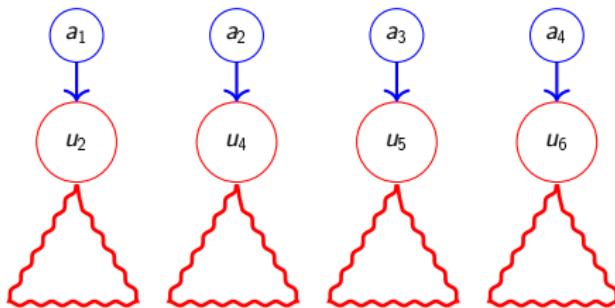
Subárbol aislado de nodos no relacionados

## Paso 1: Combinación de subárboles



Podemos combinar estos subárboles, ya que no hay restricciones entre los mismos

## Paso 2: Fusión con ancestros y descendientes

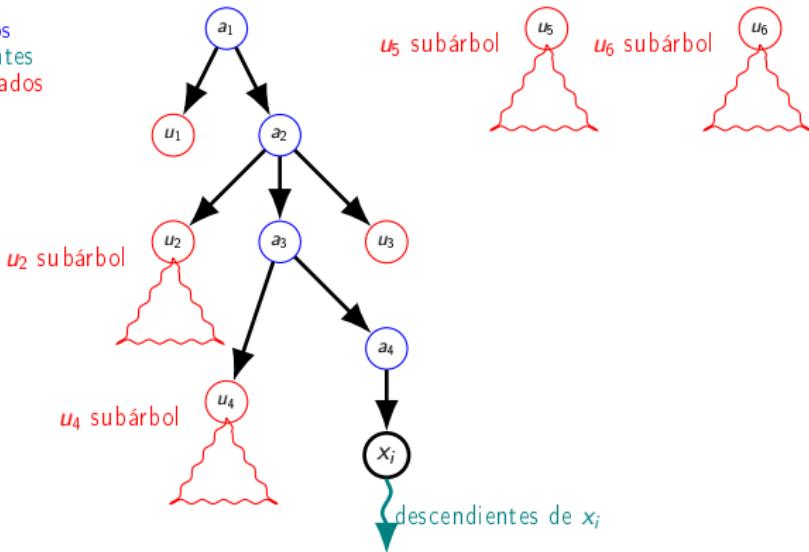


Los ancestros tienen dependencias con los subárboles y los descendientes son libres

## Restricciones en grafo etiquetado



Ancestros  
Descendientes  
No relacionados



Digrafo etiquetado en base a  $x_i$ : Ancestros (azul), Descendientes (verde), No relacionados (rojo)

## Fórmulas amenas



$\text{union}(((\text{repEC}_1, \text{ITopo}_1, \text{rTopo}_1), \dots, (\text{repEC}_{|n|}, \text{ITopo}_{|n|}, \text{rTopo}_{|n|})), n_t) = (\text{class}, l, r)$

$$class = \bigcup_{j=1}^{|n|} repEC_j \cup \{n_t\}$$

$$I = \left( \frac{\sum_{i=1}^{|n|} |L(repEC_i)|}{|L(repEC_1)|, \dots, |L(repEC_{|n|})|} \right) \prod_{i=1}^{|n|} IT_{Topo_i},$$

$$r = \left( \sum_{i=1}^{|n|} |R(repEC_i)|, |R(repEC_1)|, \dots, |R(repEC_{|n|})| \right) \prod_{i=1}^{|n|} rTopo_i$$

# Más fórmulas amenas



$$\#LO(p, i, npa) = \begin{cases} \binom{cp(i, npa)}{npa[1], \dots, npa[i]} & \text{if } |A| = i \\ \sum_{toFill=p}^{p+cp(i, npa)} \sum_{comb \in pb(toFill-p-1, i, npa)} \left( \binom{sum(comb)}{comb_1, \dots, comb_i} \cdot \#LO(toFill, i + 1, hp(comb, npa)) \right) & \text{otherwise} \end{cases}$$

# Complejidad Temporal



## Complejidad de UnrEC

- Costo de union:  $O(n)$ .



## Complejidad Temporal



## Complejidad de UnrEC

- Costo de union:  $O(n)$ .
  - Cantidad de clases generadas:  
 $\leq |EC|$ .

## Definición (Cantidad de clases de equivalencia)

*En vez de usar #clases vamos a usar  $|EC| = Ezequiel Companeetz = Equivalence Classes.$*



# Complejidad Temporal

## Complejidad de UnrEC

## Algoritmo Completo

- Costo de union:  $O(n)$ .
- Cantidad de clases generadas:  
 $\leq |EC|$ .
- Invocado en cada arból  $\leq n$ :

$$O(n \times (n \cdot |EC|)) = O(n^2 |EC|).$$



# Complejidad Temporal

## Complejidad de UnrEC

- Costo de union:  $O(n)$ .
- Cantidad de clases generadas:  $\leq |EC|$ .
- Invocado en cada arból  $\leq n$ :

$$O(n \times (n \cdot |EC|)) = O(n^2 |EC|).$$

## Algoritmo Completo

- Fusionar clase no relacionada con ancestros y descendientes:  $O(n^5 |EC|^2)$  por clase.

## Complejidad Temporal



## Complejidad de UnrEC

- Costo de union:  $O(n)$ .
  - Cantidad de clases generadas:  
 $\leq |EC|$ .
  - Invocado en cada arból  $\leq n$ :
$$O(n \times (n \cdot |EC|)) = O(n^2 |EC|).$$

## Algoritmo Completo

- Fusionar clase no relacionada con ancestros y descendientes:  $O(n^5 |EC|^2)$  por clase.
  - Clases no relacionadas:  $O(n^2 |EC|)$ .



## Complejidad Temporal



## Complejidad de UnrEC

- Costo de union:  $O(n)$ .
  - Cantidad de clases generadas:  
 $\leq |EC|$ .
  - Invocado en cada arból  $\leq n$ :
$$O(n \times (n \cdot |EC|)) = O(n^2 |EC|).$$

## Algoritmo Completo

- Fusionar clase no relacionada con ancestros y descendientes:  $O(n^5 |EC|^2)$  por clase.
  - Clases no relacionadas:  $O(n^2 |EC|)$ .
  - Total:
$$O(n^2 |EC|) + |EC| \times O(n^5 |EC|^2) = O(n^5 |EC|^3).$$

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

# ASV Exacto: Recordando la Fórmula



$$\phi_i^{\text{assym}}(\nu) = \frac{1}{|\text{topos}(G)|} \sum_{[\pi]_R \in \text{eqCI}(G, x_i)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i})) * |[\pi]_R|$$

Dado un DAG  $G$ , nodo  $x_i$  y función característica  $\nu$ .

# ASV Exacto: Pasos del Algoritmo



---

ASVExacto( $G, x_i, v$ )

---

- ①  $\{(C_k, \text{size}_k)\} \leftarrow \text{eqClassSizes}(G, x_i)$
  - ② Para cada  $C_k$ :
    - Evaluamos  $v$  sobre representante de  $C_k$  y multiplicamos por  $\text{size}_k$ .
  - ③ Sumamos los promedios:  $\phi_i^{\text{assym}}$ .
-

## ASV Aproximado



- Misma estructura que exacto, pero  $eqClassSizes$  se aproxima.

## ASV Aproximado



- Misma estructura que exacto, pero  $eqClassSizes$  se aproxima.
  - Parámetro:  $N_{samples}$  (órdenes a samplear).

## ASV Aproximado



- Misma estructura que exacto, pero  $eqClassSizes$  se aproxima.
  - Parámetro:  $N_{samples}$  (órdenes a samplear).
  - Usa `topoSortSampling` + el algoritmo naive que vimos previamente para generar las clases.

## ASV Aproximado



- Misma estructura que exacto, pero  $eqClassSizes$  se aproxima.
  - Parámetro:  $N_{samples}$  (órdenes a samplear).
  - Usa `topoSortSampling` + el algoritmo naive que vimos previamente para generar las clases.



## ASV Aproximado

- Misma estructura que exacto, pero  $\text{eqClassSizes}$  se aproxima.
- Parámetro:  $N_{\text{samples}}$  (órdenes a samplear).
- Usa `topoSortSampling` + el algoritmo naive que vimos previamente para generar las clases.

---

ASVAproximado( $G, x_i, v$ )

---

- ①  $\{(C_k, \text{size}_k)\} \leftarrow \text{sampledEqClassSizes}(G, x_i)$
  - ② Para cada  $C_k$ :
    - Evaluamos  $v$  sobre representante de  $C_k$  y multiplicamos por  $\text{size}_k$ .
  - ③ Sumamos los promedios:  $\phi_i^{\text{assym}}$ .
-



## Comparación: Exacto vs Aproximado

	Exacto	Aproximado
Precisión	100 %	Controlable
Alcance	$d\text{trees}$	Polytrees
Complejidad clases	$O(n^5 EC ^3)$	$O(\text{topoSampling}(N_{samples}))$
Complejidad promedios	$O(v) *  EC $	$O(v) *  EC_{sampled} $

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

# Sampleo aproximado de órdenes topológicos



En esta sección presentamos un algoritmo probabilístico para **muestrear órdenes topológicos** de un DAG causal  $G$ .

- A través de este muestreo vamos a poder **aproximar ASV** con precisión creciente según el número de muestras.

# Sampleo aproximado de órdenes topológicos



En esta sección presentamos un algoritmo probabilístico para **muestrear órdenes topológicos** de un DAG causal  $G$ .

- A través de este muestreo vamos a poder **aproximar ASV** con precisión creciente según el número de muestras.
- La cantidad de muestras necesaria **crece lentamente** con la precisión deseada.

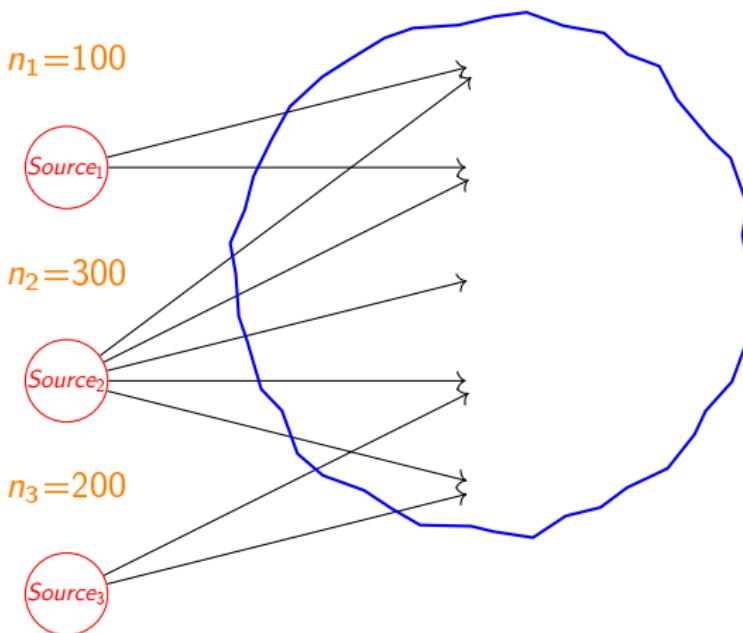
# Sampleo aproximado de órdenes topológicos



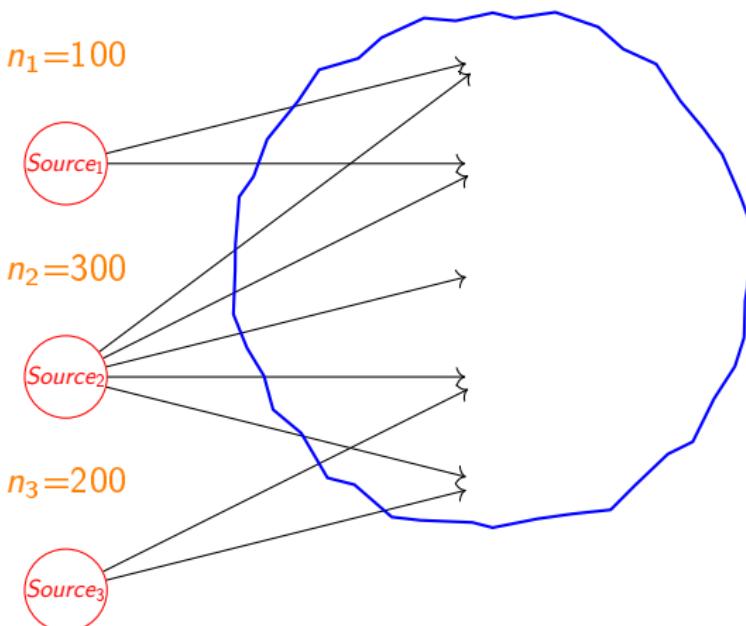
En esta sección presentamos un algoritmo probabilístico para **muestrear órdenes topológicos** de un DAG causal  $G$ .

- A través de este muestreo vamos a poder **aproximar ASV** con precisión creciente según el número de muestras.
- La cantidad de muestras necesaria **crece lentamente** con la precisión deseada.
- Nuestro objetivo inicial: **devolver un orden topológico aleatorio**.

# Algoritmo de sampleo



# Algoritmo de sampleo



$$p(\text{source}_1) = \frac{1}{6}, p(\text{source}_2) = \frac{1}{2}, p(\text{source}_3) = \frac{1}{3}$$

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

## Experimentos: Redes y Preparación



- Utilizamos dos redes bayesianas: *Cancer* y *Child*

# Experimentos: Redes y Preparación



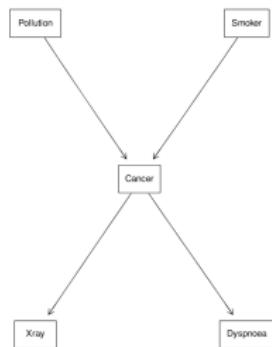
- Utilizamos dos redes bayesianas: *Cancer* y *Child*
- Ambas tienen topología de polytree o pueden adaptarse a una.

# Experimentos: Redes y Preparación



- Utilizamos dos redes bayesianas: *Cancer* y *Child*
- Ambas tienen topología de polytree o pueden adaptarse a una.
- Variables objetivo: Smoker (*Cancer*) y Age (*Child*).

## Redes utilizadas en los experimentos



(a) Red Bayesiana  
*Cancer*



(b) Red Bayesiana *Child*



# Clases de equivalencia vs Órdenes topológicos

## Definición (Fórmula original de ASV)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} w(\pi) (\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i}))$$

## Definición (Heurística con clases de equivalencia)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{[\pi]_R \in \text{eqCI}(G, x_i)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i})) * |[\pi]_R|$$



# Clases de equivalencia vs Órdenes topológicos

## Definición (Fórmula original de ASV)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} w(\pi) (\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i}))$$

## Definición (Heurística con clases de equivalencia)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{[\pi]_R \in \text{eqCI}(G, x_i)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i})) * |[\pi]_R|$$

Métricas:

- **Cardinalidad** de los conjuntos  $\Rightarrow$  cantidad de evaluaciones



# Clases de equivalencia vs Órdenes topológicos

## Definición (Fórmula original de ASV)

$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{\pi \in \text{topos}(G)} w(\pi) (\nu(\pi_{<i} \cup \{i\}) - \nu(\pi_{<i}))$$

## Definición (Heurística con clases de equivalencia)

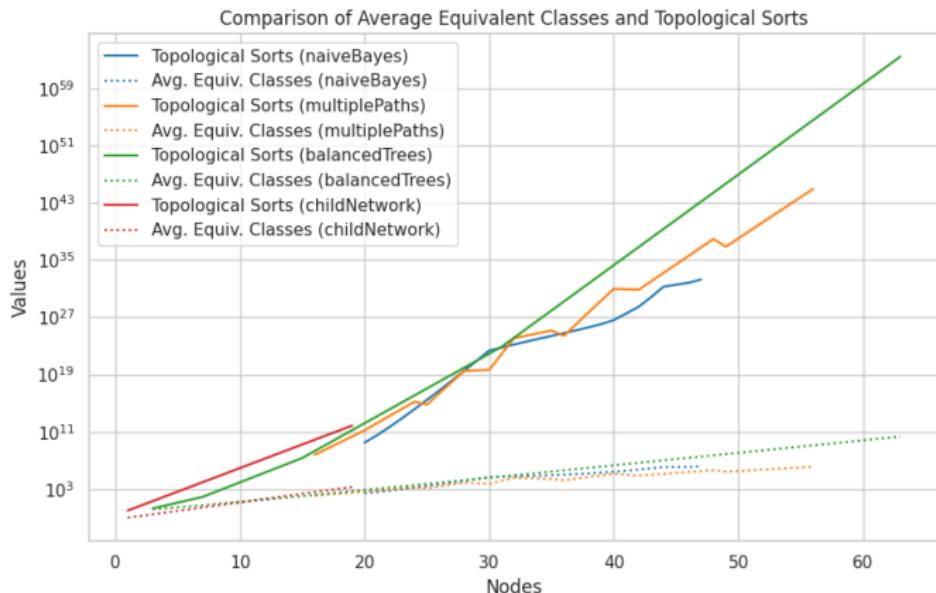
$$Shap_{M,e,\Pr}^{\text{assym}}(x_i) = \frac{1}{|\text{topos}(G)|} \sum_{[\pi]_R \in \text{eqCI}(G, x_i)} (\nu(\pi_{<i} \cup \{x_i\}) - \nu(\pi_{<i})) * |[\pi]_R|$$

Métricas:

- **Cardinalidad** de los conjuntos  $\Rightarrow$  cantidad de evaluaciones

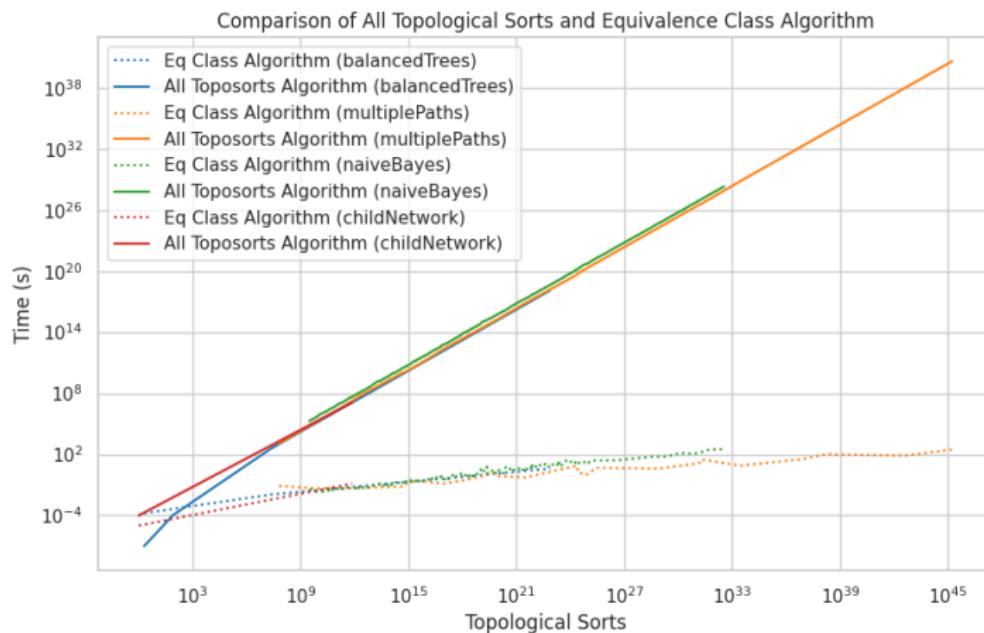


# Cantidad de Clases de equivalencia vs Órdenes topológicos



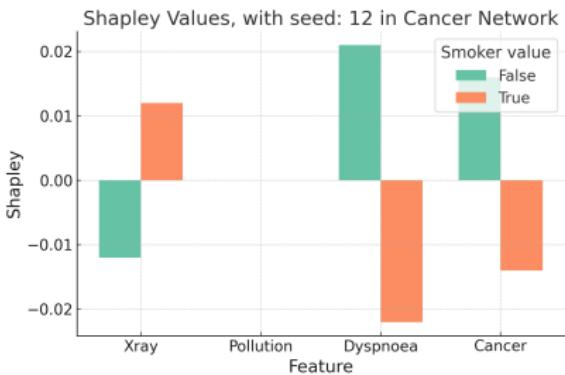
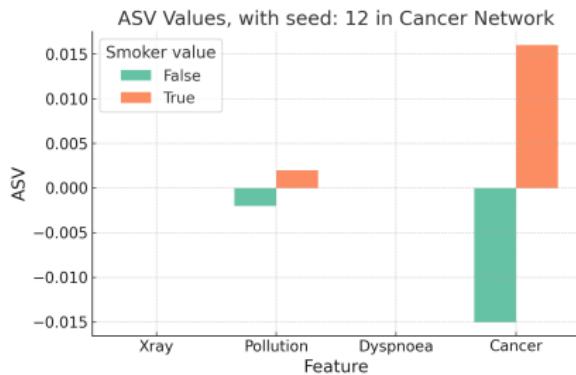
Cantidad de órdenes topológicos vs clases de equivalencia para distintas estructuras de grafos.

## Tiempo de Clases de equivalencia vs Órdenes topológicos



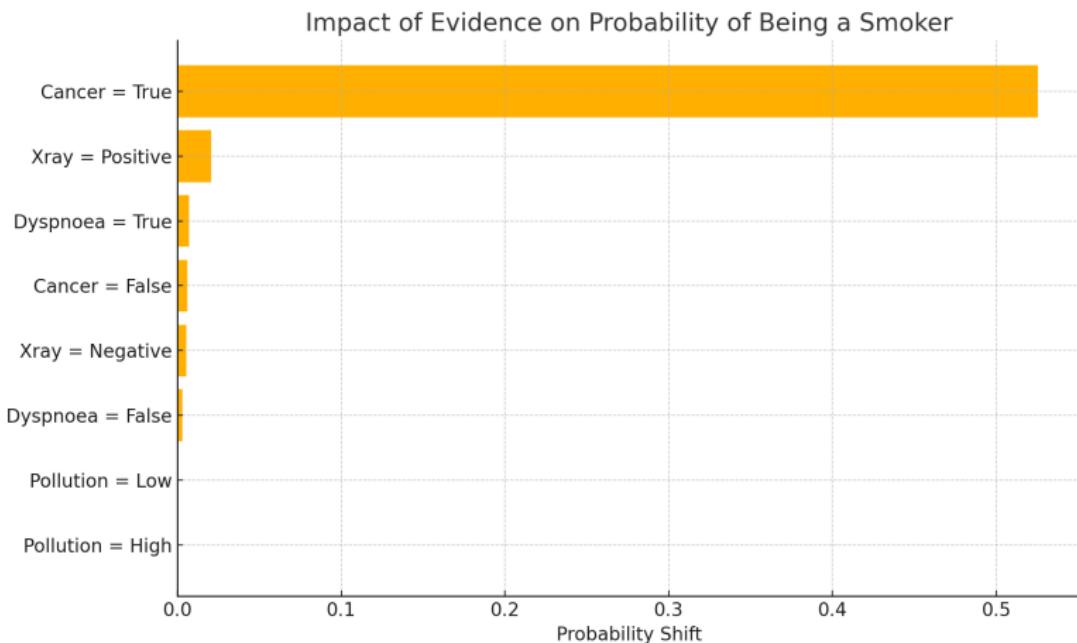
## Tiempo de clases vs topológicos usando el algoritmo de Knuth (networkx)

# Comparación ASV vs SHAP (*Cancer*)



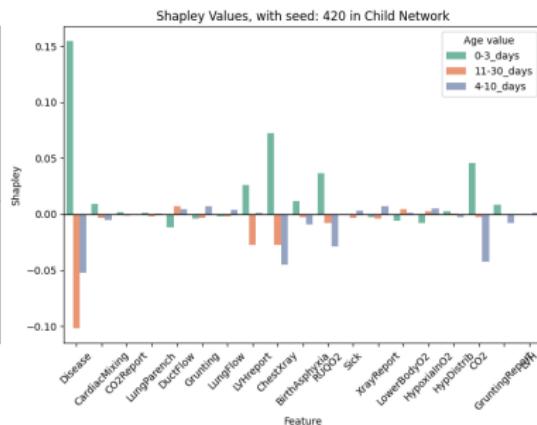
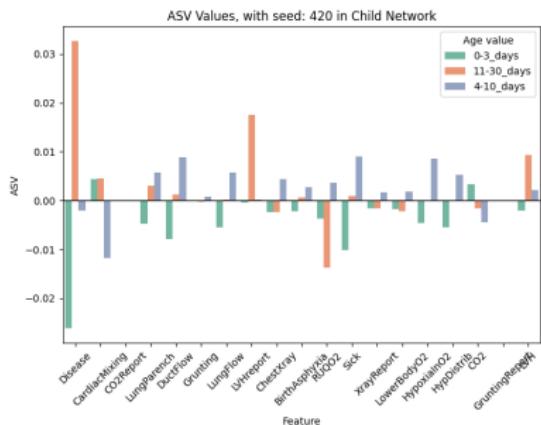
Comparación de los valores obtenidos por *ASV* y *SHAP* para las variables de la red *Cancer*

# Variación de probabilidad para Smoker



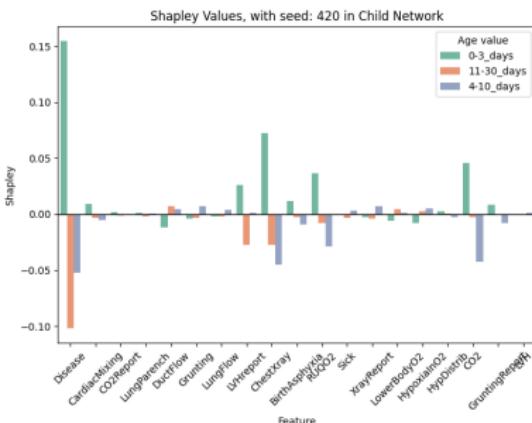
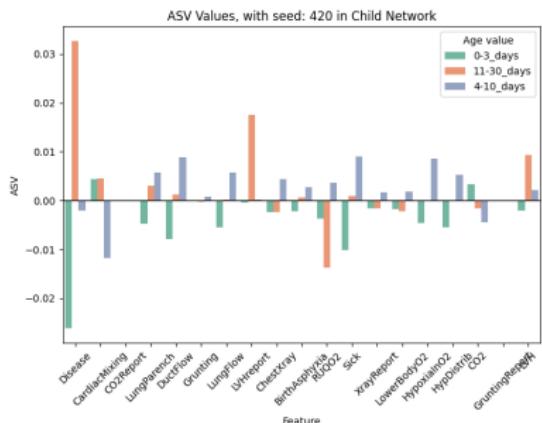
Métrica que analiza cuánto varía la probabilidad de Smoker en base a la evidencia introducida.

# Comparación ASV vs SHAP (*Child*)



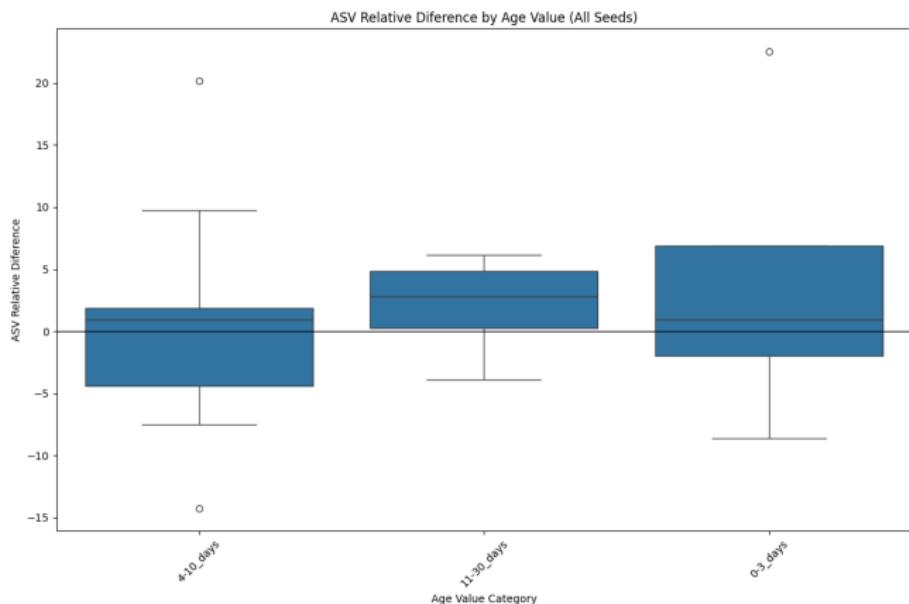
- ASV identifica Sick, DuctFlow, Disease.

# Comparación ASV vs SHAP (*Child*)



- ASV identifica Sick, DuctFlow, Disease.
- SHAP sólo logra detectar Disease.

### Child: ASV exacto vs ASV aproximado (1000 muestreros)



- Error absoluto < 5 % para  $ASV > 0,01$ .
  - Aproximación razonable sin samplear una gran cantidad de órdenes.

## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

# Conclusiones



- Se optimizó el cálculo de ASV en datos con distribuciones bayesianas y árboles de decisión.

# Conclusiones



- Se optimizó el cálculo de ASV en datos con distribuciones bayesianas y árboles de decisión.
- Se demostró la tratabilidad para Naive Bayes.

# Conclusiones



- Se optimizó el cálculo de ASV en datos con distribuciones bayesianas y árboles de decisión.
- Se demostró la tratabilidad para Naive Bayes.
- Se desarrolló un algoritmo exacto eficiente para la predicción promedio en árboles de decisión.



# Conclusiones

- Se optimizó el cálculo de ASV en datos con distribuciones bayesianas y árboles de decisión.
- Se demostró la tratabilidad para Naive Bayes.
- Se desarrolló un algoritmo exacto eficiente para la predicción promedio en árboles de decisión.
- Se definió una heurística basada en clases de equivalencia para reducir las evaluaciones.

# Conclusiones



- Se optimizó el cálculo de ASV en datos con distribuciones bayesianas y árboles de decisión.
- Se demostró la tratabilidad para Naive Bayes.
- Se desarrolló un algoritmo exacto eficiente para la predicción promedio en árboles de decisión.
- Se definió una heurística basada en clases de equivalencia para reducir las evaluaciones.
- **Se construyó un algoritmo de sampleo de órdenes topológicos con performance tratable en grafos con grados acotados.**

# Conclusiones



- Se implementó una versión exacta y otra aproximada para ASV.

# Conclusiones



- Se implementó una versión exacta y otra aproximada para ASV.
- Se comprobó empíricamente que las clases de equivalencia proporcionan una mejora significativa.



# Conclusiones

- Se implementó una versión exacta y otra aproximada para ASV.
- Se comprobó empíricamente que las clases de equivalencia proporcionan una mejora significativa.
- El principal aporte es la optimización de ASV mediante clases de equivalencia respecto de los órdenes topológicos.



# Trabajo Futuro

- Generalizar algoritmo de clases de equivalencia a *polytrees*.



# Trabajo Futuro

- Generalizar algoritmo de clases de equivalencia a *polytrees*.
- Implementar nuevas estrategias de sampleo y conteo.



# Trabajo Futuro

- Generalizar algoritmo de clases de equivalencia a *polytrees*.
- Implementar nuevas estrategias de sampleo y conteo.
- Extender la implementación de ASV para modelos y distribuciones arbitrarios.

# Trabajo Futuro

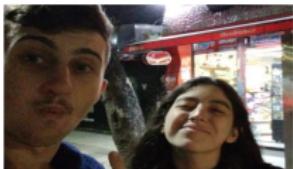


- Generalizar algoritmo de clases de equivalencia a *polytrees*.
- Implementar nuevas estrategias de sampleo y conteo.
- Extender la implementación de ASV para modelos y distribuciones arbitrarios.
- Estudiar propiedades de complejidad del sampleo y conteo de órdenes tópicos.

# Trabajo Futuro



- Generalizar algoritmo de clases de equivalencia a *polytrees*.
- Implementar nuevas estrategias de sampleo y conteo.
- Extender la implementación de ASV para modelos y distribuciones arbitrarios.
- Estudiar propiedades de complejidad del sampleo y conteo de órdenes tópicos.
- Explorar algoritmos alternativos para enumerar órdenes tópicos.



## 1 Introducción

## 2 Grafos Causales

## 3 Heurística ASV

## 4 ASV end to end

## 5 Sampleo

## 6 Experimentos

## 7 Conclusión

## 8 Extra

## Algoritmo de promedio



- ① Recorrer todas las ramas del árbol de decisión, acumulando las decisiones tomadas.

# Algoritmo de promedio



- ① Recorrer todas las ramas del árbol de decisión, acumulando las decisiones tomadas.
- ② Al llegar a una hoja:

# Algoritmo de promedio



- ① Recorrer todas las ramas del árbol de decisión, acumulando las decisiones tomadas.
- ② Al llegar a una hoja:
  - Evaluar la probabilidad de haber alcanzado esa hoja, dada la evidencia.

# Algoritmo de promedio



- ① Recorrer todas las ramas del árbol de decisión, acumulando las decisiones tomadas.
- ② Al llegar a una hoja:
  - Evaluar la probabilidad de haber alcanzado esa hoja, dada la evidencia.
  - Luego multiplicar dicha probabilidad por el valor de salida que retorna la hoja.

# Algoritmo de promedio



- ① Recorrer todas las ramas del árbol de decisión, acumulando las decisiones tomadas.
- ② Al llegar a una hoja:
  - Evaluar la probabilidad de haber alcanzado esa hoja, dada la evidencia.
  - Luego multiplicar dicha probabilidad por el valor de salida que retorna la hoja.
- ③ Sumar todas las contribuciones de cada hoja para obtener la predicción promedio.

## Algoritmo: Predicción Promedio



---

### Algorithm 1 Predicción promedio para árbol de decisión binario

---

```
1: function Mean(node, B, pathCondition, evidence)
2:   if evidence no coincide con pathCondition then
3:     return 0
4:   end if
5:   if node.isLeaf then
6:     return  $Pr_B(\text{pathCondition} \mid \text{evidence}) \cdot \text{node.value}$ 
7:   end if
8:    $X_i \leftarrow \text{node.feature}$ 
9:    $\text{left} \leftarrow \text{Mean}(\text{node.left}, B, \text{pathCondition} \cup \{X_i = 0\}, \text{evidence})$ 
10:   $\text{right} \leftarrow \text{Mean}(\text{node.right}, B, \text{pathCondition} \cup \{X_i = 1\}, \text{evidence})$ 
11:  return  $\text{left} + \text{right}$ 
12: end function
```

---

Complejidad:  $O(i|V| + I \cdot \text{varElim})$ , polinomial si *varElim* lo es (e.g. polytrees).

# Fórmula Shapley



**Fórmula general:**

Definición (Shapley Value)

$$\phi_i(v) = \text{Shapley Value del jugador } i \text{ para la función } v$$

# Fórmula Shapley



**Fórmula general:**

Definición (Shapley Value)

$$\phi_i(v) = \text{Shapley Value del jugador } i \text{ para la función } v$$

Definición

*La función característica se define como:*

$$v : \mathcal{P}(X) \rightarrow \mathbb{R}$$

*Asigna un valor real a cada posible coalición de jugadores, es decir, a cada subconjunto de  $X$ .*

# Fórmula Shapley



## Fórmula general:

### Definición (Shapley Value)

$$\phi_i(v) = \frac{1}{|X|!} \sum_{S \subseteq X \setminus \{i\}} \dots$$

- Se suman todos los subconjuntos  $S$  que no contienen a  $i$ , para ver cuánto colabora  $i$  a cada uno.

# Fórmula Shapley



## Fórmula general:

### Definición (Shapley Value)

$$\phi_i(v) = \frac{1}{|X|!} \sum_{S \subseteq X \setminus \{i\}} |S|!(|X| - |S| - 1)! \dots$$

- Se suman todos los subconjuntos  $S$  que no contienen a  $i$ , para ver cuánto colabora  $i$  a cada uno.
- El término  $|S|!(|X| - |S| - 1)!$  cuenta cuántas veces  $i$  puede llegar justo después de  $S$  en un orden.

# Fórmula Shapley



## Fórmula general:

### Definición (Shapley Value)

$$\phi_i(v) = \frac{1}{|X|!} \sum_{S \subseteq X \setminus \{i\}} |S|!(|X| - |S| - 1)! \cdot (v(S \cup \{i\}) - v(S))$$

- Se suman todos los subconjuntos  $S$  que no contienen a  $i$ , para ver cuánto colabora  $i$  a cada uno.
- El término  $|S|!(|X| - |S| - 1)!$  cuenta cuántas veces  $i$  puede llegar justo después de  $S$  en un orden.
- Se calcula el aporte marginal de  $i$  a  $S$ :  $v(S \cup \{i\}) - v(S)$ .

# Fórmula Shapley



## Fórmula general:

### Definición (Shapley Value)

$$\phi_i(v) = \frac{1}{|X|!} \sum_{S \subseteq X \setminus \{i\}} |S|!(|X| - |S| - 1)! \cdot (v(S \cup \{i\}) - v(S))$$

- Se suman todos los subconjuntos  $S$  que no contienen a  $i$ , para ver cuánto colabora  $i$  a cada uno.
- El término  $|S|!(|X| - |S| - 1)!$  cuenta cuántas veces  $i$  puede llegar justo después de  $S$  en un orden.
- Se calcula el aporte marginal de  $i$  a  $S$ :  $v(S \cup \{i\}) - v(S)$ .
- **Se divide todo por  $|X|!$ , porque se está promediando sobre todas las permutaciones posibles.**

# Fórmula función característica en ML



## Definición (Función característica)

$v_{M,e,\Pr}(S) = \text{Predicción promedio de } M \text{ cuando los features } S \text{ toman los valores de } e$

# Fórmula función característica en ML



## Definición (Función característica)

$$v_{M,e,\Pr}(S) = \sum_{e' \in cw(e,S)} \dots$$

- Se consideran las instancias  $e'$  que coinciden con la entidad  $e$  en los atributos de  $S$ :  $cw(e,S)$ .

# Fórmula función característica en ML



## Definición (Función característica)

$$v_{M,e,\Pr}(S) = \sum_{e' \in cw(e,S)} \Pr[e' | cw(e,S)] \dots$$

- Se consideran las instancias  $e'$  que coinciden con la entidad  $e$  en los atributos de  $S$ :  $cw(e,S)$ .
- Se pondera cada  $e'$  según su probabilidad condicional dado que coincide con  $e$  en  $S$ :  $\Pr[e' | cw(e,S)]$ .

# Fórmula función característica en ML



## Definición (Función característica)

$$\nu_{M,e,\Pr}(S) = \sum_{e' \in \text{cw}(e,S)} \Pr[e' | \text{cw}(e,S)] \cdot M(e')$$

- Se consideran las instancias  $e'$  que coinciden con la entidad  $e$  en los atributos de  $S$ :  $\text{cw}(e,S)$ .
- Se pondera cada  $e'$  según su probabilidad condicional dado que coincide con  $e$  en  $S$ :  $\Pr[e' | \text{cw}(e,S)]$ .
- Se evalúa el modelo  $M$  sobre cada  $e'$ .

# Fórmula función característica en ML



## Definición (Función característica)

$$v_{M,e,\Pr}(S) = \sum_{e' \in \text{cw}(e,S)} \Pr[e' | \text{cw}(e,S)] \cdot M(e')$$

- Se consideran las instancias  $e'$  que coinciden con la entidad  $e$  en los atributos de  $S$ :  $\text{cw}(e,S)$ .
- Se pondera cada  $e'$  según su probabilidad condicional dado que coincide con  $e$  en  $S$ :  $\Pr[e' | \text{cw}(e,S)]$ .
- Se evalúa el modelo  $M$  sobre cada  $e'$ .
- En resumen:  $v(S)$  es la predicción promedio del modelo dejando fijos los features de  $S$ .

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables binarios.

# Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :
  - Lado izquierdo:  $f = i$  con  $i < v$

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :
  - Lado izquierdo:  $f = i$  con  $i < v$
  - Lado derecho:  $f = d$  con  $d \geq v$

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :
  - Lado izquierdo:  $f = i$  con  $i < v$
  - Lado derecho:  $f = d$  con  $d \geq v$
- La probabilidad condicional requiere una **suma de múltiples consultas**.



## Extensión a Features No Binarios

- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :
  - Lado izquierdo:  $f = i$  con  $i < v$
  - Lado derecho:  $f = d$  con  $d \geq v$
- La probabilidad condicional requiere una **suma de múltiples consultas**.
  - Si tuviéramos la consulta  $\{ X_1 \in \{1, 2\}, X_2 = 3 \}$  la resolvemos cómo:

$$\Pr_B(X_1 = 1, X_2 = 3) + \Pr_B(X_1 = 2, X_2 = 3)$$

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :
  - Lado izquierdo:  $f = i$  con  $i < v$
  - Lado derecho:  $f = d$  con  $d \geq v$
- La probabilidad condicional requiere una **suma de múltiples consultas**.
  - Si tuviéramos la consulta  $\{ X_1 \in \{1, 2\}, X_2 = 3 \}$  la resolvemos cómo:

$$Pr_B(X_1 = 1, X_2 = 3) + Pr_B(X_1 = 2, X_2 = 3)$$

- Esto hace que la complejidad ya no sea polinomial.

## Extensión a Features No Binarios



- El algoritmo original funciona con árboles y variables **binarios**.
- Para admitir **features no binarios** adaptamos la inferencia.
- Al llegar a un nodo con umbral  $v$ , dividimos el dominio de  $f$ :
  - Lado izquierdo:  $f = i$  con  $i < v$
  - Lado derecho:  $f = d$  con  $d \geq v$
- La probabilidad condicional requiere una **suma de múltiples consultas**.
  - Si tuviéramos la consulta  $\{ X_1 \in \{1, 2\}, X_2 = 3 \}$  la resolvemos cómo:

$$Pr_B(X_1 = 1, X_2 = 3) + Pr_B(X_1 = 2, X_2 = 3)$$

- Esto hace que la complejidad ya no sea polinomial.
- **No optimizamos esta inferencia**, ya que excede los objetivos de la tesis.

# Conteo de Órdenes en *d*trees



## Fórmula general:

### Definición

Sean  $k_i$  la cantidad de nodos del subárbol  $t_i$ , con  $n = \sum_{i=1}^r k_i$ . La cantidad de órdenes topológicos es:

# Conteo de Órdenes en *d*trees



## Fórmula general:

### Definición

Sean  $k_i$  la cantidad de nodos del subárbol  $t_i$ , con  $n = \sum_{i=1}^r k_i$ . La cantidad de órdenes topológicos es:

$$\#topos(t) = \binom{n}{k_1, \dots, k_r}$$

- Coeficiente multinomial:  $\binom{n}{k_1, \dots, k_r} = \frac{n!}{k_1! \dots k_r!}$  cuenta las maneras de intercalar nodos de subárboles sin alterar su orden interno.

# Conteo de Órdenes en *dtrees*



## Fórmula general:

### Definición (Órdenes Topológicos en un *dtree*)

Sean  $k_i$  la cantidad de nodos del subárbol  $t_i$ , con  $n = \sum_{i=1}^r k_i$ . La cantidad de órdenes topológicos es:

$$\#topos(t) = \binom{n}{k_1, \dots, k_r} \cdot \prod_{i=1}^r \#topos(t_i)$$

- Coeficiente multinomial:  $\binom{n}{k_1, \dots, k_r} = \frac{n!}{k_1! \cdots k_r!}$  cuenta las maneras de intercalar nodos de subárboles sin alterar su orden interno.
- Producto de subárboles:  $\prod_{i=1}^r \#topos(t_i)$  corresponde a las combinaciones posibles dentro de cada subárbol.

# Conteo de Órdenes en *dtrees*



## Fórmula general:

### Definición (Órdenes Topológicos en un *dtree*)

Sean  $k_i$  la cantidad de nodos del subárbol  $t_i$ , con  $n = \sum_{i=1}^r k_i$ . La cantidad de órdenes topológicos es:

$$\#topos(t) = \binom{n}{k_1, \dots, k_r} \cdot \prod_{i=1}^r \#topos(t_i)$$

- Coeficiente multinomial:  $\binom{n}{k_1, \dots, k_r} = \frac{n!}{k_1! \dots k_r!}$  cuenta las maneras de intercalar nodos de subárboles sin alterar su orden interno.
- Producto de subárboles:  $\prod_{i=1}^r \#topos(t_i)$  corresponde a las combinaciones posibles dentro de cada subárbol.
- Combinación final: la fórmula multiplica ambas partes para obtener el total de órdenes topológicos.

## Fórmula de eqClassSizes



Habiendo realizado estos cálculo estamos listos para definir nuestra fórmula para el **conjunto de clases de equivalencia y sus tamaños**.

$$eqClassSizes(G, x_i) = \dots$$

# Fórmula de eqClassSizes



$$eqClassSizes(G, x_i) = \bigcup_{mix \in \prod_{j=1}^{|UR|} UnrEC(ur_j)} \dots$$

**Nota:**

- Cada  $mix$  es una combinación (producto cartesiano) de las clases de cada  $ur_j \in UR$ .

## Fórmula de eqClassSizes



$$\text{eqClassSizes}(G, x_i) = \bigcup_{\text{mix}} \left( \text{eqCI}(A, D, \text{mix}), \text{eqClassTopos}(A, D, \text{mix}) \right)$$

¿Qué hace?

- $\text{eqCI}(A, D, \text{mix})$  fusiona **ancestros *A*, descendientes *D*** y la combinación *mix*.

## Fórmula de eqClassSizes



$$\text{eqClassSizes}(G, x_i) = \bigcup_{\text{mix} \in \prod_{j=1}^{|UR|} \text{UnrEC}(ur_j)} (\text{eqCI}(A, D, \text{mix}), \text{eqClassTopos}(A, D, \text{mix}))$$

### Componentes finales:

- $\text{eqCI}(A, D, \text{mix})$ : representa la clase resultante tras fusionar.

## Fórmula de eqClassSizes



$$\text{eqClassSizes}(G, x_i) = \bigcup_{\text{mix} \in \prod_{j=1}^{|UR|} \text{UnrEC}(ur_j)} (\text{eqCI}(A, D, \text{mix}), \text{eqClassTopos}(A, D, \text{mix}))$$

### Componentes finales:

- $\text{eqCI}(A, D, \text{mix})$ : representa la clase resultante tras fusionar.
- $\text{eqClassTopos}(A, D, \text{mix})$ : cantidad de órdenes topológicos de esa clase.

[



## fragile]Algoritmo leftOrders

---

*leftOrders( $A$ ,  $actual\ ancestor$ ,  $nodes\ to\ place$ ,  $position$ )*

---

- ① Definimos donde colocar *actual ancestor* en base a *position* y a cuántos nodos tenemos disponibles en *nodes to place*, generando *new position*.
- ② Luego seleccionamos cuántos nodos de cada unrelated tree vamos a usar para llenar todas las posiciones entre *position* y *new position*, generando *new nodes*.
- ③ Eliminamos los *new nodes* de los *nodes to place*, puesto que ya los colocamos, actualizando nuestros nodos disponibles.
- ④ Realizamos el llamado recursivo actualizando la posición, nuestros nodos disponibles y nuestro ancestro actual.

## Intuición de leftOrders



- Recorre los ancestros en orden.

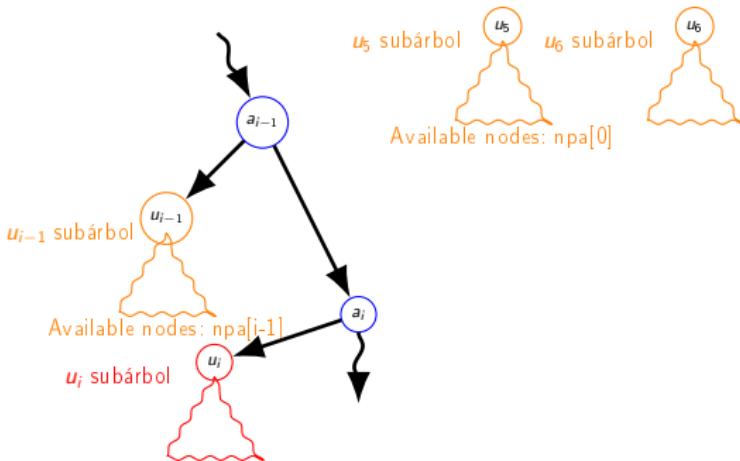
# Intuición de leftOrders



- Recorre los ancestros en orden.
- En cada paso reparte los nodos no relacionados en los “huecos” antes del ancestro:



# Iteración: Nodos Disponibles



- Solo los nodos de  $u_{i-1}$  pueden llenar el hueco antes de  $a_i$ .

# Sampleo Toposorts

---

## Algorithm 2 SampleoTopoSort( $D$ )

---

- ① **Calculamos una probabilidad  $p$  para cada uno de los nodos fuente del DAG.**
  - ① Para cada  $s \in S$  lo removemos del DAG  $D$ , y contamos la cantidad de órdenes topológicos en  $D - \{s\}$  ( $\text{toposorts}_s$ ), este valor es la cantidad de órdenes que comienzan con  $s$ .
  - ② Luego a cada  $s \in S$  le asignamos una probabilidad  $p(s) = \frac{\text{toposorts}_s}{\#\text{topos}(D)}$ .
- ② **Sampleamos** sobre  $S$  utilizando  $p$  para obtener nuestro primer nodo *start*.
- ③ **Eliminamos** a *start* de  $D$  y llamamos al algoritmo recursivamente con  $\text{SampleoTopoSort}(D - \{\text{start}\})$ , guardando el resultado en *orden*.
- ④ **Devolvemos** *start + orden* como el orden topológico sampleado.