Aprendizaje de redes de Markov basado en independencias específicas del contexto: formalización y aplicación en datos espaciales

por

Alejandro Alberto Edera

Director: Dr. Facundo Bromberg

Codirector: Dr. Guillermo Leguizamón

Agradecimientos

A mi círculo cercano. A todas aquellas personas que contribuyeron, directa o indirectamente, a ser posible este trabajo. A todas aquellas nobles personalidades en la historia de la humanidad que a través de su ejemplo influyeron en mí. Al azar y la necesidad de las circunstancias por ofrecerme la posibilidad de profundizar en mis estudios.

Índice general

Índice de figuras

Índice de cuadros

Lista de Algoritmos

Acrónimos

BLM *Bottom-up Learning Markov network structure.*

CS *Context Specific.*

CSI *Context Specific Interaction.*

CSGS *Context Specific Peter and Clark.*

CMLL *Conditional Marginal Log-Likelihood.*

CSPC *Context Specific Peter and Clark.*

D-map *Dependency map.*

DTSL *Decision Tree Structure Learning.*

GS *Grow Shrink.*

GSMN *Grow Shrink Markov Network.*

GSSL *Generate Select Structure Learning.*

HHC *Hiton Hill Climbing.*

HHC-MN *Hiton Hill Climbing Markov Network.*

IBMAP-HC *Independent-Based Maximum A Posteriori Hill Climbing.*

I-map *Independency map.*

KL *Kullback Leibler.*

LBFGS *Limited-memory Broyden–Fletcher–Goldfarb–Shanno.*

NCMLL *Normalized Conditional Marginal Log-Likelihood.*

NP *Nondeterministic Polynomial.*

PC *Peter and Clark.*

Capítulo 1

Introducción

Todos los hombres desean naturalmente saber.

Aristóteles.

El *aprendizaje de máquinas* puede ser definido como el conjunto de métodos (o algoritmos) que pueden automáticamente descubrir *patrones* (o regularidades) en un conjunto D de observaciones (o datos empíricos) [67, Capítulo 1]. Quizás, una de las características distintivas de los algoritmos de aprendizaje de máquinas es su habilidad de mejorar el descubrimiento de patrones a medida que aumenta el conjunto de observaciones [64].

Como consecuencia, un algoritmo de aprendizaje de máquinas modifica su funcionamiento en base a los datos disponibles, sin la necesidad de que un experto humano reescriba su programa o rediseñe su lógica. Sin lugar a dudas, esto tiene interesantes paralelismos con cómo los seres humanos aprenden a través de la experiencia. Por ejemplo, se puede pensar que el conjunto de observaciones utilizadas por un algoritmo se corresponde con un cúmulo de "experiencias", a partir de las cuales, un algoritmo puede *inducir conceptos* [64, Capítulo 2].

En aprendizaje de máquinas, los problemas de aprendizaje pueden ser divididos en dos grandes categorías [41, 65]: *aprendizaje supervisado* y *aprendizaje no supervisado*. La diferencia entre ambos es que, en el aprendizaje supervisado, el conjunto de datos está "etiquetado". Un conjunto de datos está etiquetado, cuando cada observación en

D está asociada a una solución potencial o respuesta esperada (etiqueta), la cual es provista por un "profesor" que guía (supervisa) el aprendizaje [64]. Gran parte del trabajo de aprendizaje de máquinas ha sido focalizado en el aprendizaje supervisado. Sin embargo, hay un interés cada vez mayor en el aprendizaje no supervisado, debido a que muchos problemas en la práctica pueden ser planteados en términos de aprendizaje no supervisado (no existe etiqueta conocida para ciertos datos, o no hay un profesor/experto que pueda etiquetarlos) [45].

Precisamente, el problema abordado por este trabajo es un problema de aprendizaje no supervisado. En un problema de aprendizaje no supervisado, los algoritmos de aprendizaje de máquinas "descubren regularidades, *características* o *estructura* desde conjuntos de datos no etiquetados"[1] [45]. En la práctica, estos **patrones o estructuras** automáticamente descubiertos por una máquina resultan significativos a la hora de analizar el conjunto D. Según [11], hay dos objetivos al momento de analizar un conjunto de datos:

Predicción. Determinar la posibilidad de ocurrencia de observaciones futuras;

Información. Extraer información sobre los fenómenos subyacentes que generaron al conjunto D.

Independientemente del objetivo perseguido, el análisis de las observaciones es una valiosa *fuente de saber*, la cual resulta clave al momento de tomar decisiones bajo *incertidumbre*.

Uno de los problemas fundamentales en el área de aprendizaje de máquinas es la estimación de distribuciones de probabilidad multivariadas desde un conjunto D de datos. Este problema es de sumo interés debido a que las distribuciones obtenidas resultan útiles para hacer predicciones o para extraer información. Ambos elementos resultan fundamentales al momento de tomar decisiones, ya que permiten reducir la incertidumbre inherente en cualquier problema del mundo real.

Por ejemplo, a través de distribuciones de probabilidad se puede definir un *modelo del lenguaje* [84]. Los modelos del lenguaje son ampliamente utilizados para *recono-*

[1] Los énfasis son nuestros.

cimiento del habla [4] y *traducción automática* [90]. Similarmente, la utilización de distribuciones de probabilidad resulta muy útil para problemas de bioinformática, e.g. *alineamiento de secuencias* [21], *predicción de genes* [83] y *predicción de la estructura de las proteínas* [27].

Por otro lado, en medicina, el diagnóstico de ciertas enfermedades por médicos presenta una *alta variabilidad* (no hay consenso), lo cual complica la toma de decisiones sobre cuáles acciones realizar para superar una determinada enfermedad. En particular, éste es el caso al diagnosticar *trastornos de cardiopatía isquémica* (*coronary heart disease*, en inglés). Como se muestra en [82], para reducir la alta variabilidad al diagnosticar dicha enfermedad, se puede estimar una distribución de probabilidad desde un conjunto de imágenes de ultrasonido del corazón. Dicha distribución de probabilidad puede luego ser utilizada por los médicos para asistirlos durante la toma de decisiones.

Sin embargo, el problema de estimación de distribuciones desde observaciones presenta grandes desafíos. Estos desafíos surgen de la *intratabilidad espacial* al representar una distribución de probabilidad, y la *intratabilidad temporal* al intentar buscar una distribución de probabilidad que se ajuste al conjunto de datos dado. En otras palabras, en el caso general, se puede demostrar que el problema de representar una distribución y el problema de estimar una distribución desde un conjunto de datos son problemas NP [46]. A pesar de ello, estos desafíos han podido ser ingeniosamente abordados en determinadas situaciones. Por ejemplo, existen modelos matemáticos que permiten representar eficientemente distribuciones de probabilidad, como también, existen algoritmos de aprendizaje de máquinas que logran automatizar la estimación de distribuciones.

Para estimar distribuciones desde observaciones, es necesario establecer varios *supuestos* sobre el problema. Quizás, algunos de los supuestos más básicos son los que determinan *cómo representar una distribución de probabilidad*. Por ejemplo, los *modelos probabilísticos gráficos*, ver [46], asumen que una distribución puede ser representada a través de dos componentes: una *estructura* y un *conjunto de parámetros*. La estructura representa cualitativamente patrones que determinan interacciones entre

las variables de la distribución. Por otro lado, el conjunto de parámetros, definidos en base a la estructura, cuantifica la posibilidad de los sucesos (o eventos) que resultan de combinar los diferentes estados de las variables que interaccionan. Como veremos más adelante, una de las posibles formas de pensar la estructura es como un grafo. Al representar una estructura como un grafo, cada uno de sus nodos está asociado a una variable de la distribución, mientras que el conjunto de aristas entre los nodos representa interacciones entre variables.

A grandes rasgos, existen dos familias de modelos probabilísticos gráficos [72, § 3.3.3]. Una son las *redes de Bayes* y la otra, de especial interés para este trabajo, son las **redes de Markov**. En términos sencillos, ambas familias *asumen* determinados tipos de interacciones entre las variables de una distribución. Concretamente, las redes de Bayes asumen *interacciones causales*, es decir, las variables de una distribución representan causas o efectos que al interaccionar representan relaciones de causa y efecto. Por esta razón, la utilización de las redes de Bayes resulta más natural para problemas relacionados con *razonamiento causal* [80, 85]. Por otro lado, las redes de Markov asumen **interacciones espaciales**, es decir, las variables de una distribución representan puntos en un espacio multidimensional (no necesariamente euclidiano). Dependiendo de la localización de un punto, y la definición de una función de "distancia"[2], se logra determinar interacciones entre *puntos vecinos*. Por lo tanto, las redes de Markov son más naturales para problemas de *análisis espacial* [91].

Para ilustrar más concretamente interacciones espaciales, es interesante pensar que el cuerpo humano puede ser visto como un sistema sumamente complejo, formado por varias partes altamente especializadas. Así, la presencia de una enfermedad en el cuerpo puede ser explicada a través de la co-ocurrencia (o interacción) de diferentes anormalidades entre sus partes. En medicina, esto último es formalmente conocido como *comorbilidad*. Como se muestra en [88], la estructura de una red de Markov puede ser utilizada para explorar la co-ocurrencia de dos o más enfermedades (*multiples comorbilidades*), lo cual puede aportar información sobre las causas de ciertas *enfermedades crónicas*.

[2]Aquí por distancia nos referimos en su sentido más amplio, es decir, una *métrica*.

Para estimar una red de Markov, el problema suele dividirse en dos subproblemas: el *aprendizaje de la estructura* y, dado una estructura, la *estimación de parámetros*. En general, para estimar redes de Markov, los algoritmos usualmente se focalizan en resolver uno de ambos subproblemas (*dívide et ímpera*, o, *divide y vencerás*). En este trabajo está especialmente focalizado en el primer subproblema, el aprendizaje de la estructura. Como veremos a lo largo de este trabajo, la estructura de una distribución tiene un rol decisivo en varios aspectos. En términos generales, este rol consiste en que la estructura permite representar y manipular eficientemente una distribución.

En la práctica, el aprendizaje de la estructura es abordado por los así llamados *algoritmos de aprendizaje de estructura*, los cuales son algoritmos especialmente diseñados para aprender la estructura de una red de Markov desde un conjunto de datos. Un rasgo distintivo de estos algoritmos es que su diseño está fuertemente influenciado por cuál es el objetivo final al analizar los datos, el cual, como hemos mencionado, puede ser predicción, información, o una combinación de ambos. Para ser más concreto, según [46, § 16.2] y [67, § 26.1], existen dos objetivos principales para estimar distribuciones y, en consecuencia, para aprender sus estructuras. Uno es denominado "*estimación de densidad*" (*density estimation*) y, el otro objetivo, es denominado "*descubrimiento de conocimiento*" (*knowledge discovery*). A continuación, describiremos más en detalle cada objetivo.

En la estimación de densidad, una estructura es principalmente aprendida para obtener una distribución que resulte altamente precisa para tareas de predicción. Por ejemplo, en problemas de *filtrado colaborativo*, una distribución es estimada desde un conjunto de datos para luego ser utilizada para predecir preferencias. Típicas predicciones en estos problemas incluyen predecir cuáles películas podrían gustarle a un persona en base a sus preferencias en otras películas, o en base a las preferencias de personas relacionadas (e.g., amigos) [42, 54]. En este tipo de problemas, la estructura es utilizada para determina cómo las películas interaccionan entre sí, o cómo las personas se encuentran relacionadas entre sí. En base a estas interacciones codificadas por la estructura, una distribución de probabilidad puede ser construida.

En descubrimiento de conocimiento, una estructura es principalmente aprendida para ser utilizada como una fuente de conocimiento desde la cual un humano (o una máquina) puede extraer información. Sin entrar en particularidades, esta información describe cómo las diferentes variables de un dominio en particular (e.g., en un problema) interaccionan entre sí. Por ejemplo, en nuestro ejemplo anterior sobre la co-ocurrencia de enfermedades, utilizando estudios médicos de varios pacientes con desordenes crónicos, se estimaron redes de Markov. Las estructuras de dichas redes fueron entonces utilizadas como una fuente de conocimiento para extraer información sobre la co-ocurrencia entre diferentes desordenes crónicos. Por ejemplo, únicamente a través del análisis de las estructuras, se hallaron algunas interesantes interacciones entre *enfermedades cardiovasculares*, e.g., trastornos de cardiopatía isquémica, y *trastornos musculoesqueléticos*, e.g., artrosis y artritis.

Para los fines de este trabajo, proponemos clasificar los algoritmos de aprendizaje de estructuras en base a los objetivos anteriores. De esta manera, para aprender la estructura de una red de Markov, tenemos *algoritmos de estimación de densidad* [75, 62, 18, 89, 56] y *algoritmos de descubrimiento de conocimiento* [87, 61, 13, 3, 81]. Una diferencia clave entre ambos grupos de algoritmos es *el tipo de estructura de datos utilizada para representar la estructura de una red de Markov*. En términos generales, la decisión de cómo representar la estructura impacta directamente en el diseño de un algoritmo de aprendizaje de estructura, porque la lógica de un algoritmo se encuentra *íntimamente relacionada* con su representación. Por lógica, nos referimos a la forma en que un algoritmo de aprendizaje de estructuras "aprende" una estructura desde un conjunto de datos. Analicemos esto último en más detalle.

En general, para representar una estructura, los algoritmos de estimación de densidad utilizan un *conjunto de características*[3], mientras que los algoritmos de descubrimiento de conocimiento utilizan un *grafo no dirigido*. Al estimar densidades, un conjunto de características es una representación altamente flexible que permite representar complejas estructuras mediante interacciones entre variables con estados

[3]Una característica puede representarse como una función indicador, la cual *indica* si un determinado subconjunto de las variables toma un estado en particular.

asignados. Desde el punto de vista del descubrimiento de conocimiento, un grafo es una representación altamente interpretable formada por dos componentes "visualizables": un *conjunto de nodos* interconectados a través de un *conjunto de aristas*. Es decir, un grafo codifica interacciones entre variables a través de la presencia o ausencia de sus aristas.

Desde el punto de vista algorítmico, la utilización de un conjunto de características reduce el problema de aprendizaje de estructuras a un problema de *inducción de características* [75], mientras que, al utilizar un grafo, el problema de aprendizaje de estructuras se reduce a un problema de *construcción de un grafo* [71]. Como resultado, la lógica subyacente de los algoritmos de ambos grupos es bastante diferente entre sí, la cual depende fuertemente de cómo la estructura es representada.

Dentro de los algoritmos de descubrimiento de conocimiento, encontramos un grupo bien conocido de algoritmos de aprendizaje de estructuras, los así llamados *algoritmos basados en restricciones* [87, 61, 13, 3, 81]. Para construir un grafo, estos algoritmos presentan una característica distintiva: la utilización de una *prueba estadística de independencia*. Básicamente, una prueba estadística de independencia es un método estadístico que permite determinar si una hipótesis tiene soporte (o no) en un conjunto de datos dado. En otras palabras, una prueba estadística es usualmente utilizada para determinar si existe suficiente evidencia para rechazar (o no) una hipótesis. De este modo, una prueba estadística puede pensarse como una especie de caja negra, donde dada una hipótesis y un conjunto de datos como entrada, retorna un valor numérico (e.g., el *valor p*) el cual es utilizado para rechazar (o no) dicha hipótesis.

En base a lo anterior, la construcción de un grafo por un algoritmo basado en restricciones puede ser, hablando en términos generales, descripto de la siguiente manera. Un algoritmo basado en restricciones formula la presencia (o ausencia) de aristas entre nodos en un grafo como una o varias hipótesis. Luego, utilizando una prueba estadística de independencia, el algoritmo determina si dicha(s) hipótesis tienen soporte en el conjunto de datos. En base a los resultados obtenidos, el algoritmo entonces construye un grafo (o modifica un grafo inicial) de tal modo de satisfacer dicha(s) hi-

pótesis. Ahora, este proceso puede ser repetido nuevamente, mediante la formulación de nuevas hipótesis, las cuales son formuladas en base a los resultados obtenidos. Así, en un número finito de repeticiones, el algoritmo eventualmente construye un grafo desde el conjunto de datos. Por esta forma de construir el grafo, dichos algoritmos son denominados "basados en restricciones", porque los resultados obtenidos por la prueba estadística de independencia *restringe* el rango de posibles grafos a construir por el algoritmo.

Sin embargo, existen situaciones donde los algoritmos basados en restricciones aprenden grafos que potencialmente pueden "ocultar información" [32], i.e., el grafo no muestra determinadas interacciones entre las variables [24]. Desde el punto de vista del descubrimiento de conocimiento, tales situaciones no son deseables debido a que van en detrimento del objetivo perseguido por dichos algoritmos [32]. En particular para este trabajo, veremos que este fenómeno sucede cuando el conjunto de datos presenta un patrón llamado **independencia específica del contexto** [10, 33, 28, 43, 44, 46].

Las independencias específicas del contexto pueden pensarse como complejas interacciones entre las variables, las cuales surgen dependiendo de un *contexto*. Un contexto está dado cuando un subconjunto de las variables del dominio toma un estado determinado. Como consecuencia, las independencias específicas del contexto producen estructuras cuyas interacciones *varían en función del contexto* [10]. Desafortunadamente, *esta variabilidad en las interacciones no puede ser codificada al utilizar un único grafo como representación de la estructura*, debido a que, sintácticamente, un grafo no puede representar diferentes interacciones entre un par fijo de nodos. Por lo tanto, si los datos presentan independencias específicas del contexto, los algoritmos basados en restricciones no resultan efectivos para descubrir dichos patrones.

Pero, analicemos este problema más a fondo. Básicamente, dicho problema surge como una consecuencia de *asumir que un grafo es una representación perfecta para la estructura de una distribución*, donde por "perfecto" nos referimos a que un grafo tiene la capacidad de poder codificar todas las interacciones presentes en una distribución. Así, para solucionar lo anterior, uno podría proponer una representación alternativa para la estructura de una red de Markov. Por ejemplo, proponiendo una representa-

ción más flexible que un grafo simple, la cual permita codificar interacciones entre variables que estén en función de un contexto. Usando esta representación alternativa, podríamos lograr que los algoritmos basados en restricciones aprendan *estructuras más precisas*, es decir, estructuras que no oculten las independencias específicas del contexto.

Sin embargo, la solución propuesta no es tan fácil de llevar a la práctica. El principal problema surge de un hecho que previamente hemos señalado: la lógica de cualquier algoritmo de aprendizaje de estructuras está íntimamente relacionada con la representación utilizada. Así, al cambiar la representación de la estructura, la lógica de un algoritmo podría verse comprometida (sería obsoleta para la nueva representación). De esta manera, para superar dicho problema, podemos señalar dos vías.

Una vía consiste en proponer una nueva representación de la estructura, junto con nuevos algoritmos de descubrimiento de conocimiento especialmente diseñados para utilizar dicha representación. Una desventaja de esta vía es que no podemos reutilizar un amplio rango de algoritmos de aprendizaje estructura, i.e., los algoritmos basados en restricciones. Justamente, al revisar la literatura [43, 44, 28, 69], podemos encontrar que todo los trabajos relacionados con independencias específicas del contexto proponen nuevas representaciones de la estructura de una red de Markov, las cuales están acompañadas por *nuevas formas de aprender la estructura de una red de Markov*.

La otra vía que, según nuestro conocimiento, aún no ha sido profundamente explorada en la literatura, consiste en proponer una nueva representación de la estructura que pueda codificar las independencias específicas del contexto y, que además, pueda ser utilizada por algoritmos de aprendizaje de estructuras ya diseñados. Precisamente, este trabajo sigue rigurosamente esta última vía.

En base a lo anterior, la contribución de este trabajo se centra principalmente en **formalizar** una nueva representación de una red de Markov, la cual hemos denominado *modelos canónicos*, y, en base a esta nueva representación, proponemos un enfoque para adaptar cualquier algoritmo basado en restricciones para utilizar esta nueva re-

presentación. Resumiendo, la contribución de este trabajo involucra los siguientes dos puntos:

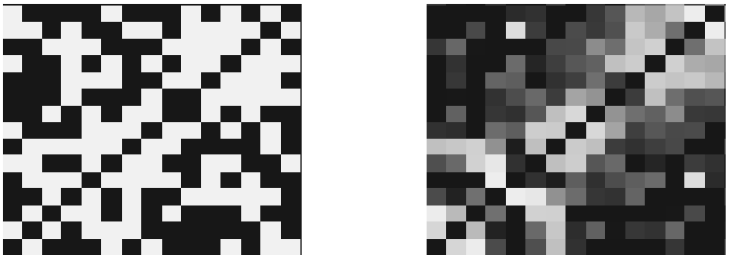
I. la presentación de los modelos canónicos como una nueva representación de una red de Markov;

II. la presentación de un enfoque para aprender la estructura de un modelo canónico desde un conjunto de datos, donde dicho enfoque está basado en la (re)utilización de los algoritmos basados en restricciones.

Formalmente hablando, un modelo canónico es un caso especial de un modelo estadístico llamado *modelo CSI* [43, 44], donde las siglas CSI provienen del inglés y significan *Context Specific Interaction*. La característica distintiva de los modelos canónicos es que su estructura puede ser representada como *un ensamble de grafos no dirigidos*, donde cada grafo en el ensamble es denominado *grafo canónico*. Matemáticamente hablando, un grafo canónico es un grafo no dirigido asociado a un *contexto canónico*, donde un contexto es canónico cuando todas las variables de una distribución tienen un estado asignado.

En consecuencia, un grafo canónico codifica un conjunto *muy particular* de interacciones entre variables, es decir, interacciones que sólo surgen desde un contexto canónico, o desde otra perspectiva, *interacciones entre estados de variables*. Desde el punto de vista de un grafo no dirigido, un grafo canónico se encuentra seriamente limitado en el tipo de interacciones e independencias que puede codificar. Sin embargo, al ensamblar un conjunto de grafos canónicos, se logra codificar un amplio rango de interacciones e independencias, debido a que la combinación de los distintos grafos permite expresar interacciones e independencias mucho más complejas, tales como las independencias específicas del contexto.

Para ilustrar este hecho, mostraremos un ejemplo concreto. La Figura 1-1 muestra la estructura de una distribución arbitraria con $n = 15$ variables, la cual presenta independencias específicas del contexto. Dicha estructura es representada en dos formas diferentes: en la Figura 1-1a, la estructura es representada por un grafo, mientras



(a) Grafo no dirigido. (b) Ensamble de grafos.

Figura 1-1: Diferentes representaciones para una estructura arbitraria. En la Figura 1-1a, la estructura es representada por un grafo, mientras que, en la Figura 1-1b, la estructura es representada por un ensamble de grafos. En cada figura, un píxel está asociado al estado de una arista (presente/ausente). Un píxel negro indica la presencia de una arista. Un píxel blanco indica la ausencia de una arista. El resto de la escala del gris denota diferentes superposición de ambos estados.

que, en la Figura 1-1b, la estructura es representada como un ensamble de K grafos canónicos.

En ambas figuras en la Figura 1-1, un grafo es representado como sigue. Sea $G = (V, E)$ un grafo no dirigido, donde V denota el conjunto de nodos y $E \subset V \times V$ denota el conjunto de aristas, tal que $|V| = n$. Un grafo G puede ser representado como una matriz m de $n \times n$ celdas, donde cada fila $i \in \{ 1, \ldots, n \}$ y columna $j \in \{ 1, \ldots, n \}$ está asociada a cada nodo en V. Así, cada arista $(i, j) \in E$ puede ser asociada a una celda $m[i, j]$. Según el valor tomado por la celda $m[i, j]$, se puede indicar la presencia o ausencia de la arista (i, j). Decimos que $(i, j) \in E$, si $m[i, j] = 1$, o, decimos que $(i, j) \notin E$, si $m[i, j] = -1$.

Debido a que consideramos que un grafo G es no dirigido, entonces toda arista $(i, j) \in E$ implica que la arista $(j, i) \in E$. En consecuencia, la matriz m asociada a un grafo G es simétrica, es decir, $m[i, j] = m[j, i]$ para todo par $i, j \in V$. En base a lo anterior, una estructura representada por un único grafo puede ser equivalentemente representada por una matriz m, mientras que un ensamble de grafos canónicos, puede ser representado por varias matrices, una por cada grafo en el ensamble. Sin embargo, cada una de las matrices de un ensamble de grafos pueden combinarse en una única matriz de la siguiente manera.

Sea $\{ G_k \}_{k=1}^{K}$ un ensamble de grafos, donde G_k es un grafo particular en el ensamble. Entonces, según lo comentado previamente, cada grafo G^k en el ensamble puede ser representado por una matriz m_k. De este modo, el ensamble $\{ G_k \}_{k=1}^{K}$ involucra $\{ m_k \}_{k=1}^{K}$ matrices, donde m_k es la matriz asociada al k-ésimo grafo en el ensamble. Ahora, dado el conjunto $\{ m_k \}_{k=1}^{K}$ de matrices, éstas pueden ser "ensambladas" en una única matriz m de la siguiente manera: $m = \frac{1}{K} \sum_{k}^{K} m_k$.

Finalmente, para representar gráficamente una matriz m (por ejemplo, una que representa un único grafo o un ensamble de grafos), podemos utilizar una matriz de píxeles, donde cada píxel está asociado a una celda de m. Según el valor tomado por $m[i, j]$, el píxel (i, j) toma un valor dentro de la escala de gris asociada al segmento de línea real $[-1, 1]$.

En base a todo lo discutido, estamos en condiciones de poder analizar la Figura 1-1. Comparando las Figuras 1-1a (un único grafo) y 1-1b (un ensamble de grafos), podemos apreciar cómo el ensamble de grafos logra representar interacciones mucho más complejas que aquellas representadas en un único grafo. Esta "complejidad" que emerge desde el ensamble es una consecuencia de superponer las interacciones e independencias de cada uno de los grafos que conforman el ensamble. Por esta razón, decimos que un grafo simple puede "ocultar información", es decir, pueden existir interacciones (como las representadas por el ensamble de grafos) que no pueden ser correctamente representadas por un grafo simple, produciéndose una pérdida de información.

Para demostrar las ventajas en términos prácticos de utilizar los modelos canónicos como una nueva representación de una red de Markov, diseñamos un enfoque que aprender la estructura de un modelo canónico desde un conjunto de datos. En términos sencillos, este enfoque puede verse como un meta-algoritmo que construye un estructura a través de la construcción de un conjunto de grafos canónicos, donde cada grafo canónico es construido por un algoritmo de aprendizaje de estructuras. Particularmente, debido a que un grafo canónico puede ser visto como un grafo simple asociado a un contexto, proponemos un método para construir un grafo canónico utilizando *cualquier algoritmo basado en restricciones*.

Para que un algoritmo basado en restricciones construya un grafo canónico, es necesario realizar una modificación en ellos. Dicha modificación se basa en la siguiente observación: todo algoritmo basado en restricciones construye un grafo utilizando una prueba estadística como caja negra. Así, modificando la prueba estadística para que esté en función de un contexto canónico, se logra que, un algoritmo basado en restricciones, construya un grafo canónico en vez de un grafo simple. Debido a que la prueba estadística es utilizada como caja negra, dicha modificación *no tiene impacto en la lógica del algoritmo* (pero sí en sus resultados). Por lo tanto, una vez modificada una prueba estadística, esta puede ser utilizada con cualquier algoritmo basado en restricciones para aprender grafos canónicos.

En particular, en este trabajo mostraremos cómo utilizar nuestro enfoque de aprendizaje de estructuras con dos algoritmos basados en restricciones bien conocidos por la comunidad: el algoritmo PC [87] y el algoritmo GS [61, 13][4]. Estas adaptaciones (o instanciaciones de nuestro enfoque) son llamadas el algoritmo *CSPC* y el algoritmo *CSGS*. El algoritmo CSPC consiste de una adaptación del algoritmo PC. En cambio, el algoritmo CSGS adapta el algoritmo GS. En términos sencillos, la principal diferencia entre el algoritmo PC y el algoritmo GS es el método utilizado para construir un grafo (o grafo solución) desde un conjunto de datos.

Más concretamente, para construir un grafo solución, ambos algoritmos comienzan desde un grafo inicial, desde el cual el algoritmo PC remueve aristas, mientras que el algoritmo GS, principalmente, agrega aristas. Como consecuencia, el grafo inicial desde el cual cada algoritmo comienza la construcción es diferente. En el caso del algoritmo PC, el grafo inicial es *completo* (todo par de nodos está conectado por una arista), mientras que el grafo inicial del algoritmo GS es *vacío* (todo par de nodos no tiene una arista).

Como veremos más adelante, estos dos tipos de métodos de construcción pueden considerarse como prototípicos dentro de la familia de los algoritmos basados en restricciones. En otras palabras, cualquier algoritmo basado en restricciones podría considerarse como un método que remueve o agrega aristas en un grafo inicial con la

[4]El significado de las siglas de estos algoritmos será introducido más adelante en este trabajo.

intención de obtener un grafo solución.

De este modo, la única diferencia entre los algoritmos CSPC y CSGS es en el método utilizado para construir un grafo canónico. Como veremos, esta simple diferencia repercute fuertemente en las complejidades temporales de ambos algoritmos. Más concretamente, CSPC presenta una alta complejidad temporal en comparación a CSGS, mientras que las estructuras obtenidas por ambos algoritmos resultan ser estadísticamente similares. Esto se debe a que el algoritmo GS (en comparación al algoritmo PC) *evita computaciones redundantes*, lo cual reduce significativamente la complejidad temporal sin perder exactitud en el resultado final.

Para evaluar empíricamente nuestro enfoque, propusimos una extensiva evaluación empírica. A grandes rasgos, dicha evaluación tiene dos objetivos. El primero, y principal, consiste en mostrar que las estructuras aprendidas por nuestro enfoque resultan ser más exactas que las estructuras aprendidas por diferentes algoritmos basados en restricciones. El segundo objetivo, secundario, consiste en analizar nuestro enfoque frente a los algoritmos de estimación de densidad. El primer objetivo apunta a dar soporte a nuestra propuesta. Por otro lado, el segundo objetivo apunta a mostrar el rendimiento de nuestro enfoque frente a una amplia gama de algoritmos utilizados en la práctica.

Para alcanzar los dos objetivos mencionados, nuestra evaluación fue dividida en dos partes: una *evaluación con conjuntos de datos sintéticos* y otra *evaluación con conjuntos de datos del mundo real*. En la evaluación con datos sintéticos, los algoritmos aprenden estructuras desde datos generados desde *distribuciones conocidas*. Esto nos permite medir con exactitud la precisión de las estructuras aprendidas por los algoritmos, debido a que las estructuras aprendidas pueden ser comparadas con la estructura real del problema.

Por otro lado, en la evaluación con los conjuntos de datos del mundo real, los algoritmos aprenden estructuras desde datos generados desde *distribuciones desconocidas*, es decir, sus estructuras son desconocidas. En particular, utilizamos 18 conjuntos de datos presentados en la literatura para evaluar algoritmos de aprendizaje de estructuras [18, 54, 89, 57, 56, 36]. Para medir la precisión de una estructura aprendida

por un algoritmo, utilizamos *el método de retención* (*holdout method* en inglés), una variante simple de *validación cruzada*. En este método, cada conjunto de datos es dividido en dos particiones, el *conjunto de entrenamiento* y el *conjunto de prueba*. Usando el conjunto de entrenamiento, se obtiene una red de Markov (estructura + parámetros), la cual es luego utilizada para predecir el conjunto de prueba, es decir, *la red de Markov se usa para una tarea de inferencia* (es la única alternativa que queda debido a que la estructura de la distribución es desconocida).

Sin embargo, como mostraremos más adelante, la precisión que puede alcanzar una red de Markov está sujeta al número de errores (imprecisiones) en su estructura. En otras palabras, una estructura errónea no puede lograr buenas predicciones. O más formalmente, mientras más precisa es la red de Markov aprendida (es decir, más similar son la estructura y los parámetros con respecto a la distribución que generó los datos), mayor es la probabilidad de predecir correctamente el conjunto de prueba. Para predecir el conjunto de prueba con una red de Markov, utilizamos el *conditional marginal log-likelihood* (CMLL) [51], una forma de realizar dichas predicciones con una amplia aceptación en la práctica [46, 18, 56, 89, 36, 25].

Los resultados obtenidos en ambas evaluaciones muestran interesantes tendencias. En la evaluación con conjuntos de datos sintéticos, nuestro enfoque logra aprender estructuras más precisas que los algoritmos basados en restricciones. Similarmente, en la evaluación con conjuntos de datos del mundo real, nuestro enfoque obtiene redes de Markov más precisas que las obtenidas por los algoritmos basados en restricciones. Curiosamente, en esta evaluación, los algoritmos basados en restricciones son ampliamente superados por los algoritmos de estimación de densidad[5], siendo únicamente nuestro enfoque el que logra obtener resultados competitivos. Adicionalmente, al comparar los algoritmos CSPC y CSGS, pudimos determinar que ambos algoritmos aprenden estructuras estadísticamente similares, pero, CSGS presenta una complejidad temporal significativamente más baja que CSPC.

[5]Esta superioridad de los algoritmos de estimación de densidad, frente a los algoritmos basados en restricciones es esperable, porque los algoritmos de estimación de densidad están justamente diseñados para tareas de inferencia.

Todas las tendencias mencionadas remarcan la importancia de codificar independencias específicas del contexto en la estructura. Por otro lado, estos resultados también remarcan el impacto qué tiene la representación de la estructura en la calidad de los resultados obtenidos por un algoritmo de aprendizaje de estructuras. Por ejemplo, nuestras dos instanciaciones (CSPC y CSGS) obtuvieron resultados mucho más preciso que los algoritmos PC y GS, donde la única diferencia entre cada par de algoritmos es la representación de la estructura (los algoritmos CSPC y CSGS usan conjuntos de grafos, mientras que PC y GS utilizan un único grafo). Así, los modelos canónicos pueden verse como un interesante forma de representar una red de Markov, cuya estructura puede ser aprendida utilizando cualquier algoritmo basado en restricciones. Como resultado, nuestro enfoque permite reutilizar los algoritmos basados en restricciones para resolver una nueva gama de problemas, donde la estructura no necesariamente puede ser representada por un único grafo.

1.1. Aportes de la tesis

Esta sección tiene como objetivo describir, a grandes rasgos, los diferentes aportes que fueron gestados como consecuencia del proceso de búsqueda de esta tesis. Para esto, decidimos describir las diferentes tareas que fueron encadenándose desde el comienzo de la investigación. A partir de la descripción de dichas tareas, señalaremos los diferentes aportes, los cuales forman las bases fundacionales de este trabajo.

Al principio, la investigación comenzó con la necesidad de comprender cuál era el impacto de codificar independencias específicas del contexto en una estructura. El origen de esta necesidad se debe a que justamente un gran número de algoritmos de aprendizaje de estructuras frecuentemente utilizados en la práctica, específicamente los algoritmos basados en restricciones, no aprenden dichas independencias. Así, se realizó una revisión bibliográfica sobre los diferentes algoritmos del estado del arte para el aprendizaje de la estructura de una red de Markov. El objetivo de la revisión fue cubrir un amplio rango de algoritmos con la intensión de comprender los diferentes enfoques existentes para aprender la estructura.

Como resultado, primero comprendimos las ventajas que pueden ser obtenidas al codificar independencias específicas del contexto en la estructura [10, 16], donde dicha ventaja principalmente consiste en la reducción del número de parámetros de una red de Markov; permitiendo obtener redes de Markov más exactas. Segundo, observamos que una de las decisiones cruciales en el diseño de un algoritmo de aprendizaje de estructuras era la decisión de cómo representar la estructura.

En base a cómo era representada la estructura, se empezó a vislumbrar la categorización de los algoritmos de aprendizaje de estructura, la cual fue presentada en la introducción de este capítulo. En otras palabras, según la representación de la estructura, los algoritmos de aprendizaje pueden ser clasificados como algoritmos de descubrimiento de conocimiento (con los algoritmos basados en restricciones como principal exponente) y los algoritmos de estimación de densidad.

Analizando los algoritmos de estimación de densidad, observamos que, a diferencia de un grafo no dirigido, un conjunto de características tiene naturalmente la capacidad de poder codificar independencias específicas del contexto. (Esto será estudiado en detalle en el Capítulo 2.) De este modo, en comparación a los algoritmos basados en restricciones, los algoritmos de estimación de densidad cuentan con la capacidad de aprender estructuras que codifiquen independencias específicas del contexto.

Sin embargo, el objetivo de los algoritmos de estimación de densidad no es aprender estructuras exactas (como es el caso de los algoritmos basados en restricciones), sino más bien, obtener estructuras complejas que resulten en distribuciones de probabilidad precisas. Es por esta razón que decidimos diseñar un algoritmo que en cierto modo reuniera las ventajas de ambas clases de algoritmos. En otras palabras, la idea era diseñar un algoritmo que aprendiera estructuras más exacta que las obtenidas por los algoritmos basados en restricciones (e.g., a través del aprendizaje de independencias específicas del contexto) pero utilizando una representación de la estructura lo suficiente flexible, tal como la utilizada por los algoritmos de estimación de densidad.

En base a esta última observación, diseñamos un algoritmo denominado el *algoritmo CS*[6]. Las siglas del nombre se deben a *Context Specific* (específico del contexto).

[6]En este trabajo, el algoritmo CS no es presentado. Esto se debe a varias razones, algunas de

La razón del nombre radica en que el algoritmo CS fue exclusivamente diseñado para aprender independencias "específicas del contexto". En muy pocas palabras, el algoritmo CS se basa en la idea de que un grafo puede ser equivalentemente codificado por un conjunto de características. Así, usando un conjunto de características obtenido desde un grafo, el algoritmo CS codifica independencias específicas del contexto sobre dichas características.

De este modo, para obtener un conjunto de características que codifiquen independencias específicas del contexto, el algoritmo CS tiene como entrada: un conjunto de datos y un grafo. Utilizando el grafo de entrada, el algoritmo CS genera un conjunto de características. Luego, en base al conjunto de datos, el algoritmo CS modifica el conjunto de características con el objetivo de codificar posibles independencias específicas del contexto presentes en el conjunto de datos.

Una desventaja que presenta el algoritmo CS es que únicamente puede aprender independencias específicas del contexto, es decir, éste no puede aprender ningún otro tipo de independencia. Es por esta razón que el algoritmo CS recibe como parámetro de entrada un grafo no dirigido, ya que dicho grafo puede codificar independencias que no pueden ser aprendidas por el algoritmo CS. Como veremos en el Capítulo 2, un grafo no dirigido únicamente puede codificar las bien conocidas *independencias condicionales*. De este modo, la idea es que el grafo de entrada usado por el algoritmo CS sea obtenido por medio de otro algoritmo, por ejemplo, por algún algoritmo basado en restricciones, evitando que el algoritmo CS aprenda independencias condicionales.

A pesar de esta desventaja, un punto destacable del algoritmo CS es que, para decidir cómo modificar las características, utiliza una prueba estadística de independencia. Precisamente, como mostraremos en el Capítulo 4, la idea de utilizar una prueba estadística para identificar independencias específicas del contexto es la base con la cual los algoritmos CSPC y CSGS deciden cómo modificar la topología de un grafo canónico.

---

las cuales son: (i) el algoritmo CS no utiliza el concepto de grafos canónicos (quizás, la principal contribución de este trabajo); (ii) la correctitud de los resultados obtenidos por el algoritmo CS no está acompañados por garantías teóricas; y (iii) el algoritmo CS es sumamente ineficiente, en términos temporales, en comparación a los algoritmos CSPC y CSGS.

Una vez implementado el algoritmo CS, diseñamos un experimento para evaluar la exactitud de las estructuras obtenidas por dicho algoritmo. El experimento se basa en la siguiente hipótesis. Si la estructura de una red de Markov presenta independencias específicas del contexto, entonces el algoritmo CS, a diferencia de un algoritmo basado en restricciones, debería poder aprender una estructura más exacta (i.e., una que codifica independencias específicas del contexto) desde un conjunto de datos muestreado desde dicha red de Markov.

Para verificar esto último, desde un conjunto de datos obtenido desde una red de Markov que presentaba independencias específicas del contexto, un grafo fue aprendido utilizando un algoritmo basado en restricciones llamado el *algoritmo HHC* (*HITON hill-climbing*) [3]. Luego, utilizando como grafo de entrada, el grafo aprendido por HHC, un conjunto de características fue obtenido por el algoritmo CS. Luego, se evaluó la exactitud de estas dos estructuras.

Para evaluar la exactitud de las estructuras aprendidas por ambos algoritmos, i.e. el grafo aprendido por HHC y el conjunto de características aprendido por CS, utilizamos el siguiente método. (Este método de evaluación es exactamente el mismo que utilizaremos en el Capítulo 5 para evaluar los resultados de los algoritmos CSPC y CSGS.) Utilizando las estructuras aprendidas por ambos algoritmos, se construyeron dos distribuciones (a través de la estimación de sus parámetros). Luego, utilizando cada una de estas distribuciones, se midió "qué tan diferente" era dicha distribución con respecto a la red de Markov que generó el conjunto de datos. Como veremos en el Capítulo 2, existe una correlación entre la precisión de una distribución y la exactitud de su estructura. En base a esto, se puede determinar que tan exacta es una estructura.

Según lo esperado, la distribución obtenida desde la estructura aprendida por el algoritmo CS fue menos diferente de la red de Markov subyacente en comparación a la distribución obtenida desde la estructura aprendida por el algoritmo HHC. En base a dichos resultados, se confeccionó un artículo donde se presentó formalmente el algoritmo CS. Dicho artículo fue publicado en el *XVII Congreso Argentino de Ciencias de la Computación* [22].

Sin embargo, el algoritmo CS tiene varias desventajas. Quizás la principal de ellas es que el diseño del algoritmo CS tiene poco sustento teórico, por lo tanto, las estructuras aprendidas por CS no son necesariamente correctas. Por ejemplo, la modificación del conjunto de características a través de una prueba estadística de independencia se basaba principalmente en intuiciones en vez de fundamentos formales.

Por estas razones, realizamos un nuevo trabajo para profundizar en cómo utilizar una prueba de independencia como método para verificar independencias específicas del contexto. (Gran parte de estos resultados se encuentran en el Capítulo 4.) Éste resultado teórico fue presentado en un artículo titulado "*Markov random fields factorization with context-specific independences*". Dicho artículo nunca fue publicado formalmente en ningún congreso o revista, sin embargo, éste se encuentra disponible en un repositorio online de reportes técnicos [23].

En base a los resultados teóricos previamente comentados, diseñamos un nuevo algoritmo basado en independencias el cual denominamos el algoritmo CSPC [7]. A diferencia del algoritmo CS, el diseño del algoritmo CSPS utilizó ideas del algoritmo CS combinadas con un algoritmo basado en independencias bien conocido: el algoritmo PC [87]. El algoritmo CSPC es similar al algoritmo CS en el sentido que utiliza un conjunto de características como representación de la estructura. En contraste, el algoritmo CSPC se diferencia del algoritmo CS en qué CSPC no necesita un grafo de entrada, es decir, CSPC puede aprender independencias específicas del contexto e independencias condicionales.

En pocas palabras, el algoritmo CSPC se basaba en la idea siguiente idea. Debido a que hay una equivalencia entre un grafo y un conjunto de características, entonces, como consecuencia, dos observaciones pueden ser extraídas. (Esto último será estudiado en más detalle en el Capítulo 2.) Primero, dado un conjunto de características equivalente a un grafo, entonces un subconjunto de dichas características

[7]Esta versión de CSPC puede considerarse como una versión primigenia del algoritmo CSPC presentado en este trabajo. La principal diferencia entre ambas versiones de CSPC es la forma en cómo es representada la estructura. La versión de CSPC comentada aquí utilizada un conjunto de características, mientras que la versión de CSPC presentada en este trabajo utiliza un conjunto de grafos canónicos.

puede ser asociado a la presencia (o ausencia) de una arista en el grafo. Segundo, existe un mapeo entre operaciones sobre un grafo y operaciones sobre un conjunto de características.

En base a las dos observaciones previas, dado un conjunto inicial de características, el algoritmo CSPC "remueve aristas en el conjunto de características" con el objetivo de codificar posibles independencias presentes en un conjunto de datos. La decisión de "remover una arista" es tomada en base a la ejecución de una prueba estadística de independencia.

Implementado el algoritmo CSPC, éste fue comparado empíricamente frente a otros algoritmos basados en independencia, entre ellos el propio algoritmo PC. Los resultados obtenidos mostraron que CSPC logró aprender estructuras más exacta en comparación a todos los algoritmos utilizados en la evaluación empírica. En base a dichos resultados, se confeccionó un nuevo artículo, el cual fue enviado a *the 25th IEEE International Conference on Tools with Artificial Intelligence* (ICTAI) [24].

Una vez aceptado el artículo previamente mencionado, recibimos una invitación de los organizadores de ICTAI para publicar dicho artículo en un "especial issue" en *the International Journal on Artificial Intelligence Tools* (IJAIT). Para enviar el artículo para su publicación en IJAIT, una de las exigencias consistía en realizar una extensión significativa de los contenidos del artículo presentado en ICTAI, tanto a nivel teórico como a nivel experimental.

Dentro de las tareas involucradas para extender el trabajo, la principal tarea consistió en la confección de un elegante marco teórico para explicar formalmente el funcionamiento de CSPC. Esta marco teórico son los modelos canónicos, los cuales presentaremos formalmente en el Capítulo 3. Como consecuencia de este nuevo marco teórico, también presentamos una nueva versión de CSPC que utilizaba un conjunto de grafos canónicos como representación de la estructura, esta versión de CSPC es la que se presenta en el Capítulo 4.

Por otro lado, otras de las extensiones del trabajo de ICTAI fueron las siguientes. Primero, la utilización de un mayor número de algoritmos de aprendizaje de estructuras de redes de Markov para realizar una evaluación más exhaustiva. Dentro

de estos nuevos algoritmos, se destaca la utilización de algoritmos de estimación de densidad, los cuales presentaremos en el Capítulo 5. Segundo, una extensión de los resultados empíricos, los cuales abarcaron la utilización de conjuntos de datos obtenidos desde varios problemas del mundo real (e.g. filtrado colaborativo, clasificación de documentos, etc.), los cuales también mostraremos en el Capítulo 5.

La extensión del artículo de ICTAI fue aceptada y publicada en IJAIT [25].

Luego del algoritmo CSPC, el resto de la investigación se focalizó en resolver una de las desventajas de CSPC: su complejidad temporal. Para lograr esto, la idea consistió en diseñar un nuevo algoritmo que pudiese alcanzar la misma exactitud que CSPC pero con un menor costo computacional. Teniendo este objetivo en mente, se diseñó el algoritmo CSGS (el cual presentaremos en el Capítulo 4). El diseño de dicho algoritmo es bastante similar a CSPC pero, en vez de utilizar el algoritmo PC, se utilizó otro algoritmo basado en independencias: el algoritmo GS [13].

En términos de complejidad temporal, se puede demostrar que el algoritmo GS tiene una menor complejidad temporal en comparación al algoritmo PC. Adicionalmente, la reducción en complejidad temporal en GS no implica una pérdida significativa en la calidad de las estructuras aprendidas.

En base a esto, formulamos la siguiente hipótesis: las estructuras aprendidas por CSGS deberían ser significativamente equivalentes a las obtenidas por CSPC, pero CSGS debería aprender dichas estructuras con una menor complejidad temporal en comparación a CSPC. Para verificar esta hipótesis, implementamos el algoritmo CSGS y realizamos una evaluación empírica. Esta evaluación es presentada en el Capítulo 5.

Los resultados obtenidos en dicha evaluación confirmaron nuestra hipótesis, es decir, mostraron que CSGS obtenía estructuras similares a CSPC, pero CSGS tenía una complejidad temporal mucho menor que CSPC. Esto nos permitió confeccionar un nuevo artículo, el cual fue enviado a *the 14th Ibero-American Conference on Artificial Intelligence* [26].

1.2. Visión general de los restantes capítulos

Esta sección ofrece una visión general de este trabajo. A grandes rasgos, este trabajo está formado por seis capítulos, cada uno de los cuales aborda un tema específico. Adicionalmente, cada capítulo se encuentra acompañado por una introducción y un resumen, los cuales resaltan los principales puntos abordados y sus relaciones.

Los capítulos que conforman este trabajo y sus respectivos objetivos se detallan a continuación. Como ya hemos apreciado, el Capítulo 1 tiene como objetivo ofrecer una introducción de este trabajo. El Capítulo 2 tiene como objetivo ofrecer varios conceptos básicos, los cuales resultan sumamente útiles para entender profundamente nuestras contribuciones. El Capítulo 3 tiene como objetivo introducir formalmente nuestra principal contribución, los modelos canónicos junto con una propuesta para representar su estructura, un conjunto de grafos canónicos. El Capítulo 4 tiene como objetivo presentar nuestra segunda contribución, un enfoque para aprender un conjunto de grafos canónicos desde un conjunto de datos. El Capítulo 5 presenta una evaluación empírica que muestra las ventajas que presenta nuestro enfoque, en términos de la exactitud de las estructuras aprendidas, frente a otros algoritmos de aprendizaje de estructuras. Este trabajo culmina con el Capítulo 6. En dicho capítulo, extraeremos algunas conclusiones generales y mostraremos algunas vías de trabajo futuro. Finalmente, este trabajo incluye un apéndice, el Apéndice A. Este apéndice presenta algunos axiomas claves sobre el concepto de independencia, los cuales serán de utilidad para demostrar varios resultados a lo largo del trabajo.

Tomando como base la visión general previamente comentada, ofrecemos una visión más detallada de algunos de los principales contenidos.

El Capítulo 2 está formado por dos grandes partes. Por un lado, la Sección 2.1 describe minuciosamente el concepto de independencia, principalmente las definiciones de independencia condicional e independencia específica del contexto. En base a estas definiciones, la Sección 2.1.1 describe formalmente el concepto de "estructura de una distribución de probabilidad".

Por otro lado, dicho capítulo también presenta los métodos involucrados para el

aprendizaje de una red de Markov desde un conjunto de datos. Como hemos mencionado, este problema puede ser dividido en dos subproblemas: el aprendizaje de estructuras y la estimación de parámetros.

El aprendizaje de estructuras es presentado en la Sección 2.3. Para entender este problema, entre otras cosas, estudiaremos dos algoritmos prototípicos del enfoque basado en restricciones: los algoritmos PC y GS, presentados en las Secciones 2.4.3 y 2.4.4, respectivamente. Estos algoritmos son de sumo interés para nuestro trabajo debido a que, en el Capítulo 4, los utilizaremos como base para mostrar nuestro enfoque para aprender grafos canónicos.

La estimación de parámetros es estudiada en la Sección 2.5. Entre otras cosas, mostraremos que la estimación de parámetros dependen directamente de ejecutar inferencia sobre una red de Markov. De este modo, la Sección 2.6 describe cómo realizar inferencia sobre una red de Markov.

En el Capítulo 3, los modelos canónicos serán presentados de forma gradual para facilitar su comprensión. La Sección 3.2 presenta un resumen de los diferentes enfoques existentes en la práctica para representar independencias específicas del contexto. Dentro de estos enfoques, la Sección 3.3 presentará los modelos CSI. En base a los modelos CSI, la Sección 3.5 presenta los modelos divididos, un modelo propuesto en la práctica para representar la estructura de un modelo CSI. Por otro lado, en la Sección 3.6, presentaremos los modelos canónicos, nuestra propuesta para representar la estructura de un modelo CSI. Como mostraremos en la Sección 3.6.3, la estructura de un modelo canónico puede ser representada como un conjunto de grafos canónicos. Finalmente, en la Sección 3.7, mostraremos que un conjunto de grafos canónicos tiene la capacidad de representar independencias específicas del contexto así como independencias condicionales.

El Capítulo 4, además de presentar nuestro enfoque para aprender modelos canónicos desde un conjunto de datos, también presenta en su Sección 4.2 cómo utilizar una prueba estadística de independencia para verificar independencias específicas del contexto. La Sección 4.3.2 presenta los algoritmos CSPC y CSGS. La Sección 4.4 describe la complejidad temporal de ambos algoritmos. Finalmente, la Sección 4.5

presenta un método para obtener un conjunto de características desde un conjunto de grafos canónicos.

El Capítulo 5 está divido en dos grandes partes. La primera parte describe en detalle todos los elementos involucrados en nuestra experimentación. Así, la Sección 5.1 describe los diferentes conjuntos de datos que utilizamos en la experimentación. La Sección 5.2 detalla los diferentes algoritmos que utilizamos para aprender una red de Markov desde un conjunto de datos. La Sección 5.3 describe la metodología llevada acabo en la experimentación. Por otro lado, la segunda parte, específicamente las Secciones 5.4 y 5.5, analiza los resultados obtenidos de la experimentación.

Capítulo 2

Antecedentes

Las matemáticas poseen no sólo la verdad, sino una suprema hermosura – una hermosura fría y austera, como la de una escultura.

Bertrand Russell.

Este capítulo se focaliza en presentar varios conceptos básicos utilizados a lo largo de este trabajo. A grandes rasgos, este capítulo esta estructurado en dos grandes partes: representación y aprendizaje de redes de Markov. La primera parte muestra algunas de las estructuras de datos (o representaciones) utilizadas para codificar una red de Markov. La segunda parte muestra algunos de los métodos para estimar (inducir o aprender) una red de Markov desde un conjunto de observaciones D.

Como veremos, la representación y la estimación de una red de Markov están entrelazadas, debido a que la estimación depende, en cierto punto, de la representación elegida. Antes de ver en profundidad cada uno de estos conceptos, presentaremos una visión general de ambos. Para este fin, comenzaremos hablando sobre el conjunto de observaciones D.

Figura 2-1: La naturaleza como una caja negra.

Estrictamente hablando, cualquier observación es obtenida, en última instancia, desde la naturaleza. Desde el punto de vista del ser humano, la naturaleza es un

gran misterio, es decir, tenemos *incertidumbre* sobre cuáles son, y cómo trabajan, sus mecanismos internos. Como consecuencia, la naturaleza suele ser representada como una *caja negra* [11], tal como se muestra en la Figura 2-1.

Matemáticamente hablando, una caja negra es una función desconocida que tiene un conjunto X de variables de entrada y un conjunto Y de variables de salida. Así, una observación de la caja negra puede ser representada como una tupla $(X = x, Y = y)$ (o *estado*), donde (x, y) es tan sólo una de las posibles combinaciones de estados que pueden tomar las variables de entrada y salida, respectivamente.

Adicionalmente, se asume que esta caja negra es (o está regida por) una *distribución de probabilidad subyacente*, denotada por $p^\star(X, Y)$. En nuestro trabajo, usualmente se desconoce cuáles variables son entradas y cuáles variables son salidas, entonces, por fines prácticos, podemos simplemente agruparlas las variables X e Y en un único conjunto de variables, el cual denotaremos como X. En base a esto, una distribución puede ser simplemente denotada como $p^\star(X)$.

Una distribución de probabilidad es una función $p(X)$ que asocia a todo estado x una probabilidad $p(x)$, donde $p(x) \in [0, 1]$. Una probabilidad es una medida de la posibilidad de que el estado x ocurra en un determinado dominio. Una distribución se diferencia de una *relación estrictamente determinista*, e.g. $Y = f(X)$, porque las variables de una distribución son *aleatorias*. Una variable es aleatoria si sus estados están sujetos al azar ("randomness").

En la práctica, la distribución $p^\star(X)$ es desconocida pero se asume *computable* [86]. Al ser computable, una distribución de probabilidad puede ser representada como un modelo matemático, por ejemplo, un *modelo paramétrico* $M_\theta = (k, \theta)$, donde θ es un conjunto (o vector) de parámetros en el espacio de parámetros $\Theta \equiv \mathbb{R}^k$. Si las $|X| = n$ variables son discretas, entonces el modelo paramétrico puede ser representado como una "gran tabla" de parámetros, donde cada entrada asocia un estado de las variables, $X = x$, a un parámetro. Decimos "gran tabla" porque, si asumimos que las variables son binarias (cada variable toma dos estados $\{ 0, 1 \}$), la tabla contiene $|\theta| = 2^n$ parámetros. Por ejemplo, consideremos una simple distribución con 119 variables

binarias[1], su representación como tabla involucraría $|\theta| = 2^{119}$ parámetros.

Para comprender la bastedad del número 2^{119}, consideremos lo siguiente. Tomemos uno de los objetos más pequeños conocidos por el hombre: un protón[2]. Según las últimas estimaciones, el *radio* de un protón es de aproximadamente $0{,}88 \times 10^{-15}$ metros. Ahora, utilicemos este objeto para almacenar la tabla, por ejemplo, asumiendo que podemos almacenar un parámetro sobre la superficie de un protón. Como resultado, para almacenar la tabla completa, necesitaríamos 2^{119} protones. A continuación, organicemos los 2^{119} protones en un segmento. Este segmento tendría una longitud aproximada de 110000 años luz, es decir, el diámetro promedio de nuestra Vía Láctea...

Una forma eficiente para representar un modelo paramétrico (y por ende una distribución), consiste en asumir que la distribución subyacente tiene una *forma funcional*, es decir, las variables de la distribución se encuentran agrupadas (o relacionadas) a través de funciones. Al asumir esto, entonces la distribución subyacente puede ser expresada como un modelo multiplicativo, es decir, una distribución puede ser *factorizada* (o descompuesta) en un *producto de funciones paramétricas más simples*, donde "simple" implica baja dimensionalidad o un menor número de parámetros.

De este modo, en vez de utilizar una única "gran tabla", una distribución puede ser representada por varias "tablas pequeñas". Al factorizar una distribución, nos permite obtener grandes reducciones en el número de parámetros. Para ilustrar esto, consideremos la distribución anterior con $119 = |X|$ variables. Asumamos que esta distribución puede ser expresada como un producto de $|X|$ funciones paramétricas, es decir, una función por variable. Dado que cada variable es binaria, cada función pueden ser representada como una tabla con sólo 2 parámetros (uno por cada estado de la variable). Esta factorización reduce el número de parámetro de 2^{119} a tan sólo 2×119 parámetros. Si alineamos 2×119 protones obtendríamos un segmento cuya longitud es aproximadamente 4×10^{-13} metros. Para ilustrar lo pequeño de este

[1]En nuestra evaluación empírica manipularemos algunas distribuciones que superan las 119 variables.

[2]Existen objetos más pequeños que un protón (e.g., un electrón). Sin embargo, debido a la dualidad "onda-partícula", estos objetos más pequeños se comportan más como una onda en vez de una partícula, dejando así de ocupar un lugar específico en el espacio.

segmento, el diámetro del organismo más pequeño conocido actualmente[3] es de 17×10^{-9}.

Precisamente, las redes de Markov, y más ampliamente los modelos probabilísticos gráficos, asumen que una distribución tiene una forma funcional. Más concretamente, asumen una familia de distribuciones conocida como *distribuciones de Boltzmann* (o distribuciones de Gibbs). Una distribución de Boltzmann puede ser expresada como un producto de funciones paramétricas. Particularmente, una red de Markov es una distribución de Boltzmann la cual es usualmente representada por un modelo paramétrico, denotado por (S, θ_S). De este modo, una red de Markov (S, θ_S) está formada por: S, un objeto matemático llamado la *estructura*, cuyo objetivo es determinar la factorización de la distribución en un conjunto de funciones paramétricas; y el conjunto θ_S de parámetros de las funciones paramétricas obtenidas por la factorización. Dada una red de Markov (S, θ_S), una distribución $p(X)$ se define como:

$$p(X) = \frac{1}{Z} \prod_{i:\, S \models C} \phi_i(X_C), \tag{2.1}$$

donde $\phi_i(X_C)$ es una función paramétrica, usualmente llamada *factor* o *potencial*[4], la cual está definida sobre un subconjunto de variables, $X_C \subseteq X$, llamado el *alcance* del factor; y Z es una constante, usualmente llamada *función de partición*[5], que garantiza que el producto de factores resulte en una probabilidad (un número entre 0 y 1). La notación $S \models C$ denota que el conjunto de índices C sigue lógicamente de la topología de la estructura S.

El componente central de la Ecuación 2.1 es la estructura S de la red de Markov. La función de la estructura es restringir el rango de posibles formas de una distribución de Boltzmann a través de la reducción del número de parámetros. En otras

[3] *El Circovirus porcino tipo I.*

[4] Una función potencial mide el grado de compatibilidad asociado a cada estado $x_c \in val(C)$, donde el grado de compatibilidad es expresado a través de un parámetro.

[5] El nombre "función de partición" tiene su origen en mecánica estadística. Muchas de las primeras contribuciones en mecánicas estadística provienen de países de habla alemana (Alemania, el ex-Imperio Austríaco, etc.). Por ejemplo, el uso de la letra Z proviene de la palabra alemana *Zustandssumme*, que significa "suma sobre estados".

palabras, la estructura *descompone* (o *factoriza*) una distribución de Boltzmann en un conjunto particular de factores [39]. Se puede demostrar que esta descomposición es válida siempre y cuando la estructura represente *independencias* entre variables presentes en la distribución subyacente (más detalle en la Sección 2.1). Precisamente, las independencias representadas por S son las que determinan los alcances de los factores, y, en consecuencia, el conjunto de parámetros θ_S de la red de Markov. Por ejemplo, consideremos una red de Markov (S, θ_S) arbitraria, el número $|\theta_S|$ de parámetros está limitado por el factor cuyo alcance es mayor, es decir, $|\theta_S| \propto 2^{k^\star}$, donde $k^\star = \underset{S \models C}{\arg\max} |X_C|$, donde C sigue lógicamente de la topología de la estructura S.

Para estimar una red de Markov, la única información que tenemos es un conjunto de $m = |D|$ observaciones[6] obtenidas desde $p^\star(X)$. Usando el conjunto D como evidencia, una red de Markov puede ser inducida desde D utilizando *razonamiento Bayesiano*[7] [64, § 6]:

$$(\hat{S}, \hat{\theta}_{\hat{S}}) = \underset{S \in \mathcal{S}, \theta_S \in \Theta}{\arg\max} \; p(S, \theta_S | D),$$

donde Θ denota el espacio de parámetros y $\mathcal{S}$ el espacio de estructuras candidatas. Así, la red de Markov $(\hat{S}, \hat{\theta}_{\hat{S}})$ se corresponde con el máximo global de la distribución $p(S, \theta_S|D)$. En la práctica, esta estimación suele hacerse en dos pasos. Aplicando la regla de la cadena sobre $p(S, \theta_S|D)$, tenemos que:

$$p(S, \theta_S|D) = p(S|D)p(\theta_S|S, D).$$

En base a esta descomposición, la idea es utilizar las distribuciones $p(S|D)$ y

[6]Las observaciones D se suelen asumir IID (*independent and identically distributed*), es decir, cada observación fue muestreada desde $p^\star(X)$ *independientemente* de las restantes observaciones en D y la probabilidad de cada variable es *idéntica* a la de las restantes variables.

[7]El principal componente del razonamiento Bayesiano es el *teorema de Bayes*. Este teorema recibe su nombre en honor a Thomas Bayes. Bayes fue un sacerdote, filósofo y matemático inglés. En abril de 1761, Bayes muere dejando sin publicar un ensayo que contenía las bases del teorema de Bayes. Este ensayo, titulado "Essay towards solving a problem in the doctrine of changes", fue publicado en forma póstuma por un amigo de Bayes. En la época en que fue escrito el ensayo, la frase "doctrine of changes" puede ser entendida como teoría de la probabilidad. Como se describe en [77], "este ensayo no presenta un descubrimiento sobre la naturaleza del universo sino un enfoque sobre cómo nosotros realizamos inferencias sobre el mundo natural.".

$p(\theta_S|S, D)$ para aprender la estructura y estimar los parámetros, respectivamente. Para esto, primero asumiremos una estructura S arbitraria pero fija, de este modo el conjunto de parámetros más probable en el espacio de parámetros Θ, denotado por $\hat{\theta}_S$, puede ser obtenido del siguiente modo:

$$\hat{\theta}_S = \arg\max_{\theta_S \in \Theta} p(\theta_S|S, D). \tag{2.2}$$

La tarea de resolver esta ecuación es conocida como *estimación de parámetros*, ver [46, § 20]. Para estimar los parámetros, es necesario ejecutar *inferencia* en la red de Markov para determinar el valor de la función de partición Z. Para entender esto último, primero hay que notar que "inferencia", en términos generales, consiste en computar una distribución condicional desde una distribución $p(X)$. Una distribución condicional se define como $p(X_Q|X_E)$, donde X_Q y X_E son dos conjuntos disjuntos en X: las *variables de consulta* y las *variables de evidencia*, respectivamente. En base a esto, se puede verificar que el valor de la función de partición Z es equivalente a computar la distribución de probabilidad $p(\emptyset|X_V)$, es decir, una distribución de probabilidad donde las variables de consulta son igual a $X_Q = \emptyset$ y las variables de evidencia son igual a $X_E = X_V$.

Sin embargo, para aprender los parámetros, es necesario previamente conocer la estructura S de la distribución. En muchos problemas prácticos, la estructura es usualmente desconocida o imposible de construir por expertos humanos (debido a la complejidad del dominio, e.g., un elevado número de variables). En tales casos, podemos utilizar la distribución $p(S|D)$ para aprender una estructura de la siguiente manera:

$$
\begin{aligned}
\hat{S} &= \arg\max_{S \in \mathcal{S}} p(S|D), \\
\hat{S} &= \arg\max_{S \in \mathcal{S}} \frac{p(S,D)}{p(D)}, \\
\hat{S} &= \arg\max_{S \in \mathcal{S}} \frac{p(D|S)p(S)}{p(D)}, \\
\hat{S} &\propto \arg\max_{S \in \mathcal{S}} p(D|S). \qquad (2.3)
\end{aligned}
$$

Debido a que el conjunto D es fijo, entonces $\frac{1}{p(D)}$ es una constante y puede ser removida. Por otro lado, sin un conocimiento previo que permita establecer preferencias entre las diferentes estructuras en $\mathcal{S}$, entonces la probabilidad $p(S)$ puede ser asumida uniforme; no afectando la maximización. La tarea de resolver la Ecuación 2.3 es conocida como *aprendizaje de estructuras*, para más detalles ver [46, § 16.4].

Resumiendo, para estimar una red de Markov desde el conjunto D, primero estimamos su estructura $\hat{S}$ con la Ecuación (2.3); y entonces, usando la estructura $\hat{S}$, estimamos el conjunto $\hat{\theta}_{\hat{S}}$ utilizando la Ecuación (2.2).

En lo que resta de este capítulo, presentaremos en más detalle ambas tareas. Primero nos centraremos en la representación de una red de Markov. Para esto, en la Sección 2.1 comenzaremos introduciendo formalmente el concepto de independencia y mostraremos que la estructura de una red de Markov puede definirse como un conjunto de independencias. En la Sección 2.2 presentaremos diferentes representaciones de la estructura de una red de Markov. Luego, en la Sección 2.3, ofreceremos una descripción detallada sobre el aprendizaje de estructuras. Particularmente, en la Sección 2.4, mostraremos cómo trabajan los algoritmos basados en restricciones a través del estudio de dos algoritmos basados en restricciones bien conocidos: el algoritmo PC (Sección 2.4.3) y el algoritmo GS (Sección 2.4.4). Por otro lado, en la Sección 2.5, mostraremos diferentes métodos para realizar la estimación de parámetros. Debido a que la estimación de parámetros tiene como subrutina el uso de inferencia, la Sección 2.6 describirá en qué consiste ejecutar inferencia sobre una red de Markov.

Notación

Usaremos V para denotar una secuencia finita de índices, donde dicho conjunto tiene $|V| = n$ índices. Un índice en V es denotado en minúscula, e.g., $a, b \in V$. En contraste, un índice en mayúscula denota un subconjunto de índices, e.g., $A, B \subseteq V$. Denotamos un conjunto de variables aleatorias indexadas por V como $X_V = \{ X_a, X_b, \dots \}$, por razones de simplicidad diremos $X_V \equiv X$, cuando asumimos un conjunto V fijo. Usamos X_A para denotar un subconjunto de variables en X, i.e., $X_A = \{ X_a : a \in A \subseteq V \}$.

Denotamos como $val(a)$ al conjunto de estados que puede tomar una variable $X_a \in X$. Igualmente, dado un subconjunto arbitrario de variables X_W con $W \subseteq V$, denotamos como $val(W)$ al conjunto de estados que puede tomar X_W, es decir, $val(W) = \underset{a \in W}{\times} val(a)$. Usamos el símbolo $x \equiv x_V$ como un estado genérico en $val(V)$.

Un estado x puede ser visto como una conjunción de estados marginales, uno por cada variable $a \in V$, i.e. $x \equiv \bigwedge_a x_a$, $a \in V$. Dado cualquier estado x, usaremos x_A para denotar el sub-estado en x para las variables X_A. Denotamos como $X = x^k$ (o simplemente x^k) al hecho de que la variable X toma el k-ésimo estado en $val(V)$.

Por otro lado, dado un dominio definido por un conjunto de variables X_V decimos que un estado $x \in val(V)$ es *canónico*, si toda variable X_a, $a \in V$ tiene un estado asignado. Por simplicidad, sólo asumiremos variables binarias, es decir, para cada variable $X_a \in X$, su conjunto de estados es $val(a) = \{ 0, 1 \}$, sin embargo, cualquiera de nuestros resultados puede extenderse con naturalidad a conjuntos de estados no binarios.

Teoría de grafos

A continuación introduciremos algunos conceptos esenciales sobre teoría de grafos, los cuales serán utilizados recurrentemente en todo este trabajo. Para facilitar la comprensión de los mismos, los conceptos serán ilustrando utilizando el grafo mostrado en la Figura 2-2. Un *grafo no dirigido* es denotado por $G = (V, E)$, donde V es un conjunto de nodos y $E \subset V \times V$ es un conjunto de aristas no dirigidas. Debido a que

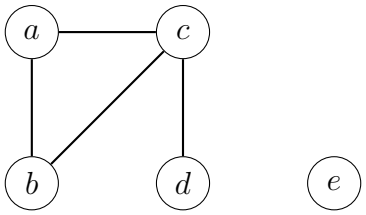


Figura 2-2: Un grafo no dirigido $G = (V, E)$.

un conjunto de nodos puede verse como un conjunto de índices, sobrecargaremos la notación V para también denotar los nodos de un grafo. En la Figura 2-2, el grafo G tiene como conjunto de nodos, el conjunto $V = \{\, a, b, c, d, e \,\}$, y, como conjunto de aristas, el conjunto $E = \{\, (a, b), (a, c), (b, c), (c, d) \,\}$.

Dos nodos $a, b \in V$ son adyacentes en G si existe una arista $(a, b) \in E$. Las *adyacencias de un nodo* $a \in V$ son denotadas por $G(a) = \{\, b \in V \setminus \{a\} \colon (a, b) \in E \,\}$. En la Figura 2-2, tenemos que las adyacencias del nodo c es el conjunto $G(c) = \{\, a, b, d \,\}$, y las adyacencias del nodo e es el conjunto $G(e) = \emptyset$.

Un grafo es *vacío* si, para todo par $a, b \in V$, tenemos que $(a, b) \notin E$. Similarmente, un grafo es *completo* (o totalmente conectado) si, para todo par $a, b \in V$, $(a, b) \in E$. El *grado* de un nodo en G es $|G(a)|$. El *grado máximo* de G es el grado máximo de sus nodos: $\arg\max_{a \in V} |G(a)|$. En la Figura 2-2, el grado máximo de G es 3.

Sea $G = (V, E)$ un grafo no dirigido, y sea U un subconjunto propio de V. Un grafo $G' = (U, E')$ es un *subgrafo* de G, si $E' = \{(a, b) \in E \colon a, b \in U\}$, o equivalentemente, $E' = (U \times U) \cap E$. En la Figura 2-2, podemos obtener el siguiente subgrafo $G' = (V \setminus \{\, c \,\}, \{\, (a, b) \,\})$, el cual resulta de remover el nodo c (y todas sus aristas) en G.

Dado un grafo $G = (V, E)$ y una lista de nodos $a, b, c, d, \ldots$ de V, decimos que la lista de nodos es un *camino*, si, para todo par de nodos consecutivos en la lista (i.e., $(a, b), (b, c), (c, d), \ldots$), existe una arista en E. En la Figura 2-2, tenemos que $a - b - c - d$ es un camino en G.

Dado un grafo G, un nodo a puede *alcanzar* un nodo b en G, o b es *alcanzable* desde a en G, si existe un camino en G entre ellos. En otro caso, decimos que b no es alcanzable desde a, o similarmente, que los nodos a y b *están separados* en G. En la Figura 2-2, el nodo b es alcanzable desde cualquier nodo en $\{\, a, c, d \,\} \subseteq V$; pero no

es alcanzable desde el nodo e.

Ahora, dado un grafo $G = (V, E)$ donde el nodo a es alcanzable desde el nodo b, denotamos como $\text{sep}_G(a, b; U)$ al hecho de que ambos nodos están separados al remover el conjunto U de nodos en G, es decir, a y b están separados en $G' = (V \setminus U, E')$. En la Figura 2-2, tenemos que $\text{sep}_G(b, e; \emptyset)$ y $\text{sep}_G(b, d; \{ c \}) \equiv \text{sep}_{G'}(b, d; \emptyset)$, donde $G' = (V \setminus \{ c \}, \{ (a, b) \})$.

El concepto de separabilidad de nodos puede ser extendido a conjuntos de nodos. Sea A y B dos conjuntos disjuntos de V, decimos que A y B están separados en G si, para todo $a \in A$ y $b \in B$, tenemos que $\text{sep}_G(a, b; \emptyset)$. Sea A, B y U tres subconjuntos disjuntos de V, y sea $G' = (V \setminus U, E')$ un subgrafo de G. Decimos que A está separado de B dado U, denotado por $\text{sep}_G(A, B; U)$, si $\text{sep}_{G'}(A, B; \emptyset)$ se satisface en G'.

Un subconjunto C de V es un *clique* de G si el subgrafo $G' = (C, E')$ es completo. Por ejemplo, en el grafo de la Figura 2-2, el conjunto $\{ b, c, d \}$ no es un clique en G porque $(b, d) \notin E$, pero los siguientes subconjuntos:

$$\{ a, b, c \}, \{ a, b \}, \{ a, c \}, \{ b, c \}, \{ c, d \}, \{ a \}, \{ b \}, \{ c \}, \{ d \}, \{ e \}$$

son todos cliques en G.

Un clique C de G es *máximo* si, para todo $a \in V \setminus C$, el subgrafo $G' = (C \cup \{ a \}, E')$ no es completo. El conjunto de cliques máximos en un grafo G es denotado por $\mathcal{C}(G)$. Por ejemplo, en la Figura 2-2, el conjunto $\mathcal{C}(G)$ es igual a:

$$\{ \{ a, b, c \}, \{ c, d \}, \{ e \} \}.$$

2.1. Independencias

Como dijimos anteriormente, una estructura codifica interacciones entre variables. Estableciendo esto a la inversa, una estructura también codifica no interacciones entre variables o *independencias* entre variables. Quizás, las independencias más amplia-

mente conocidas son las así llamadas *independencias condicionales.*

Definición 2.1. *Independencia condicional* [19]. Sea $p(X)$ una distribución de probabilidad sobre X. Sean A, B, y U subconjuntos arbitrarios y disjuntos de V. Entonces X_A es *condicionalmente independiente* de X_B dado X_U, denotado por $I(X_A \perp X_B \mid X_U)$, si y solo si

$$p(x_A|x_B, x_U) = p(x_A|x_U), \tag{2.4}$$

siempre y cuando, para todo estado en $val(B \cup U)$, $p(x_{B\cup U})$ es positiva, es decir, $p(x_{B\cup U}) > 0$. $I(\cdot \perp \cdot \mid \cdot)$ es denominado como un *supuesto de independencia.*

Como es notado por Dawis [19], una manera intuitiva de pensar un supuesto $I(X_A \perp X_B \mid X_U)$ es que la distribución condicional de X_A, dado X_B y X_U, se encuentra completamente determinada por los valores de X_U, resultando X_B superfluo una vez que X_U es conocido. En ciencia de la información, un supuesto de independencia tiene una interesante interpretación relacionada con el concepto de *relevancia.* Supongamos que cada variable $X_a \in X$ representa una porción de conocimiento, entonces el conocimiento de X_U, implica que el conocimiento de X_B es *irrelevante* para comprender X_A.

En base a lo anterior, los supuestos de independencia permiten reducir la complejidad espacial de representar una distribución $p(X)$. Supongamos que $p(X)$ es representada como un modelo paramétrico (e.g., una tabla de parámetros), los supuestos de independencia presentes en $p(X)$ nos permite *reducir el número de parámetros* para representar $p(X)$. Por ejemplo, dado una distribución arbitraria $p(X_A|X_B, X_U)$, una representación paramétrica requeriría definir $2^{|A\cup B\cup U|}$ parámetros (un parámetro por cada posible estado de las variables). Sin embargo, si el supuesto $I(X_A \perp X_B \mid X_U)$ está presente, entonces $p(X_A|X_B, X_U) = p(X_A|X_U)$, implicando una reducción de $2^{|A\cup B\cup U|}$ a sólo $2^{|A\cup U|}$ parámetros.

Un caso particular de las independencias condicionales son las *independencias marginales.* Decimos que las variables X_A son *marginalmente independientes* de X_B si $I(X_A \perp X_B \mid \emptyset)$. En otras palabras, una independencia marginal es una independencia

condicional donde el conjunto condicionante es vacío, i.e., $X_U = \emptyset$. En base a la Definición 2.1, se puede demostrar que $p(X_A, X_B) = p(X_A) \times p(X_B)$, si $I(X_A \perp X_B \mid \emptyset)$ se cumple en $p(X_A, X_B)$.

A pesar de los beneficios de las independencias condicionales, un supuesto de independencia condicional impone una restricción bastante fuerte sobre una distribución. Por ejemplo, si $I(X_A \perp X_B \mid X_U)$ está presente en $p(X)$, $p(X)$ debe satisfacer la condición $p(X_A, X_B|X_U) = p(X_A|X_U)p(X_B|X_U)$ para **todos los estados de** X_U. En otras palabras, si existe al menos un estado $x_U \in val(U)$ tal que $p(X_A, X_B|x_U) \neq p(X_A|x_U)p(X_B|x_U)$, entonces el supuesto $I(X_A \perp X_B \mid X_U)$ **no se cumple** en $p(X)$. Las *independencias específicas del contexto* son una forma de relajar la definición de independencia condicional [10].

Definición 2.2. *Independencia específica del contexto* [1]. Sea $p(X)$ una distribución de probabilidad sobre X. Sean A, B, U, y W subconjuntos arbitrarios y disjuntos de V, y x_W un estado arbitrario de X_W en $val(W)$. Entonces X_A es *contextualmente independiente* de X_B dado X_U y el contexto x_W, denotado por $I(X_A \perp X_B \mid X_U, x_W)$, si y solo si

$$p(x_A|x_B, x_U, x_W) = p(x_A|x_U, x_W), \tag{2.5}$$

siempre y cuando, para todo estado en $val(B \cup U \cup W)$, $p(x_{B \cup U \cup W})$ es positivo, es decir, $p(x_{B \cup U \cup W}) > 0$.

Para facilitar la comprensión del concepto de independencia específica del contexto, es sumamente útil pensar dicha independencia como una independencia condicional donde un subconjunto de las variables condicionantes tienen un estado asignado. Precisamente, dicho estado se corresponde con el contexto de una independencia específica del contexto. Consecuentemente, una independencia condicional puede ser definida de una forma alternativa, utilizando como bloque constructor el concepto de independencia específica del contexto.

Concretamente, dado cualquier supuesto de independencia condicional $I(X_A \perp X_B \mid X_W)$, dicho supuesto puede ser equivalentemente expresado como una conjunción

de independencias específicas del contexto como sigue [43, § 2.2][23]:

$$I(X_A \perp X_B \mid X_W) \equiv \bigwedge_{x_W \in val(W)} I(X_A \perp X_B \mid x_W). \tag{2.6}$$

Como resultado, dado una distribución $p(X)$ arbitraria, una *independencia condicional* $I(X_A \perp X_B \mid X_W)$ se satisface en $p(X)$, si y solo si, para todo contexto $x_W \in val(W)$, la *independencia específica del contexto* $I(X_A \perp X_B \mid x_W)$ también se satisface en $p(X)$.

Siguiendo la Definición 2.2, la independencia $I(X_A \perp X_B \mid x_W)$ es una independencia específica del contexto donde x_W es el contexto y el conjunto condicionante X_U es el conjunto $\emptyset$. De este modo, una independencia condicional únicamente se satisface en $p(X)$, si la Ecuación (2.8) se satisfacen en $p(X)$. Por esta razón, las independencias condicionales suelen ser denominadas como *independencias simétricas*, i.e., una independencia condicional se cumple para todos los estados de sus variables condicionantes. En contraste, las independencias específicas del contexto suelen ser denominadas *independencias asimétricas* [35], i.e., una independencia específica del contexto sólo se cumple para un estado específico.

Por otro lado, un hecho sumamente interesante de las independencias específicas del contexto, y *crucial para los resultados presentados por este trabajo*, es el siguiente resultado [44, 38, 23].

Lema 2.1. Sea $p(X)$ una distribución positiva, y sea $I(X_A \perp X_B \mid X_U, x_W)$ un supuesto de independencia específica del contexto en $p(X)$. Entonces, el supuesto $I(X_A \perp X_B \mid X_U, x_W)$ es equivalente a un supuesto de independencia condicional, $I(X_A \perp X_B \mid X_U)$, en la distribución condicionada por el contexto x_W.

Demostración. Dado que $I(X_A \perp X_B \mid X_U, x_W)$ se satisface en $p(X)$, entonces por la Definición 2.2 tenemos que:

$$p(X_A|X_B, X_U, x_W) = p(X_A|X_U, x_W),$$

utilizando la definición de distribución condicional, lo anterior puede ser reexpresado como:

$$\frac{p(X_A, X_B, X_U, x_W)}{p(X_B, X_U, x_W)} = \frac{p(X_A, X_U, x_W)}{p(X_U, x_W)}.$$

Utilizando al regla de la cadena en ambos lados obtenemos la siguiente igualdad:

$$\frac{p(X_A, X_B, X_U|x_W)p(x_W)}{p(X_B, X_U|x_W)p(x_W)} = \frac{p(X_A, X_U|x_W)p(x_W)}{p(X_U|x_W)p(x_W)},$$
$$\frac{p(X_A, X_B, X_U|x_W)}{p(X_B, X_U|x_W)} = \frac{p(X_A, X_U|x_W)}{p(X_U|x_W)}.$$

Ahora, por sustitución, decimos que $p'(\cdot) \equiv p(\cdot|x_W)$, donde el dominio de $p'(\cdot)$ es $X_{V'}$ con $V' = V \setminus W$. Así, la igualdad previa puede ser equivalentemente expresada como:

$$\frac{p'(X_A, X_B, X_U)}{p'(X_B, X_U)} = \frac{p'(X_A, X_U)}{p'(X_U)}.$$

Finalmente, utilizando el teorema de Bayes obtenemos la siguiente igualdad:

$$p'(X_A|X_B, X_U) = p'(X_A|X_U).$$

En base a la Definición 2.1, la igualdad anterior implica que $I(X_A \perp X_B \mid X_U)$ se satisface en $p'(X_{V'})$ y, por la sustitución, entonces $I(X_A \perp X_B \mid X_U)$ también se satisface en $p(X_{V'}|x_W)$. □

En otras palabras, un supuesto $I(X_A \perp X_B \mid X_U, x_W)$ que se satisface en $p(X)$ implica que el supuesto $I(X_A \perp X_B \mid X_U)$ también se satisface en $p(X_{V\setminus W}|x_W)$. Como veremos en el próximo capítulo, este hecho es sumamente importante porque, usando independencias condicionales, podemos determinar la presencia de independencias específicas del contexto en una distribución. Así, si queremos detectar la presencia de un supuesto de independencia en un conjunto de datos muestreados desde una distribución $p(X)$, los métodos para resolver dicho problema están diseñados para

detectar supuestos de independencia condicional, no supuestos de independencias específicas del contexto. Usando el Lema 2.1, cualquiera de estos métodos pueden ser adaptados para detectar independencias específicas del contexto.

Finalmente, introduciremos un concepto más sobre independencia, el cual jugará un rol clave durante el Capítulo 3, donde introduciremos la principal contribución de este trabajo. Este concepto es la base desde la cual surge la definición de independencia específica del contexto y, consecuentemente, la definición de independencia condicional.

Definición 2.3. *Independencia instanciada.* Sea $p(X)$ una distribución de probabilidad sobre X. Sean A, B y U subconjuntos arbitrarios y disjuntos de V, y sea x un *estado canónico* arbitrario de X en $val(V)$. Entonces x_A es *condicionalmente independiente* de x_B dado x_U, denotado por $I(x_A \perp x_B \mid x_U)$, si y solo si

$$p(x_A|x_B, x_U) = p(x_A|x_U), \tag{2.7}$$

siempre y cuando, $p(x_{B\cup U})$ sea positivo, es decir, $p(x_{B\cup U}) > 0$.

Decimos que la Definición 2.3 es el concepto sobre el cual se construyen las Definiciones 2.1 y 2.2 porque ambas independencias son definidas con respecto a *variables aleatorias*, mientras que una independencia instanciada es definida con respecto a *estados tomados por variables aleatorias*. En otras palabras, el concepto de independencia entre variables aleatorias se construye sobre el concepto de independencia entre estados [19], donde en teoría de probabilidad, un estado de variable se corresponde con un *evento*.

Para ilustrar esto último, la Definición 2.4 en [46] introduce el concepto de *independencia condicional entre variables aleatorias*, donde dicho concepto toma como base el concepto de *independencia condicional entre eventos*; la Definición 2.3 en [46]. Usualmente en la práctica ambos conceptos, independencia entre variables e independencia entre eventos, no son diferenciados. Por ejemplo, en [46], las Definiciones 2.3 y 2.4 tienen el mismo nombre, i.e., ambas son llamadas "independencias condicionales". Sin embargo, para los fines de este trabajo, necesitamos diferenciar entre independen-

cias entre estados de variables (eventos) e independencias entre variables aleatorias. Por esta razón, al introducir la Definición 2.3, es posible identificar, sin ambiguedades, las independencias sobre eventos.

Alternativamente, el concepto de independencia instanciada necesariamente implica el concepto de *dependencia instanciada*. Por ejemplo, una variable X_A es dependiente de otra variable X_B cuando los estados de X_A dependen de los estados de X_B, i.e., cuando es posible determinar los estados de X_A a partir de los estados de X_B. Así, la dependencia entre X_A y X_B necesariamente implica que ocurren dependencias instanciadas entre las variables. Por ejemplo, el estado $X_A = x_A$ puede ser dependiente de cualquier estado $X_B = x_B$. Así, siempre que en el mundo ocurre el estado $X_B = x_B$, resulta que X_A toma el estado x_A.

En base a lo anterior, la importancia de la Definición 2.3 radica en que, tomando cualquier independencia condicional, dicha independencia puede ser expresada como un conjunto de independencias instanciadas. Por ejemplo, al analizar la Ecuación (2.4), podemos observar que una independencia condicional, $I(X_A \perp X_B \mid X_U)$, se cumple en $p(X)$, si y solo si $p(x_A|x_B, x_U) = p(x_A|x_U)$ para todo $x \in val(A \cup B \cup U)$. Por la Definición 2.3, tenemos que $p(x_A|x_B, x_U) = p(x_A|x_U)$ es justamente una independencia instanciada, la cual podemos expresar simbólicamente como $I(x_A \perp x_B \mid x_U)$.

De esta manera, el valor de verdad de cualquier independencia condicional puede ser expresado como una conjunción de valores de verdad de independencias instanciadas. Formalmente,

$$I(X_A \perp X_B \mid X_U) \equiv \bigwedge_{x \in val(A \cup B \cup U)} I(x_A \perp x_B \mid x_U). \tag{2.8}$$

Similarmente, esto puede ser aplicado a una independencia específica del contexto. Es decir, el valor de verdad de cualquier independencia específica del contexto puede ser expresado como una conjunción de valores de verdad de independencias instanciadas. Formalmente,

$$I(X_A \perp X_B \mid X_U, x_W) \equiv \bigwedge_{x' \in val(A \cup B \cup U)} I(x'_A \perp x'_B \mid x'_U, x_W). \tag{2.9}$$

Resumiendo, estas dos últimas ecuaciones nos muestran que **cualquier independencia condicional o independencia específica del contexto presente en una distribución pueden ser completamente caracterizada utilizando independencias instanciadas.** Como veremos a lo largo de este trabajo, esto último permite reducir considerablemente el problema de manipular independencias, ya que utilizando independencias instanciadas podemos expresar cualquier independencia condicional o independencia específica del contexto.

2.1.1. Estructura

En esta sección ofreceremos una definición formal de una estructura de una distribución. Además, presentaremos algunas relaciones útiles entre diferentes estructuras. Primero, definiremos $\mathcal{I}$ como el conjunto de todos los posibles supuestos de independencia sobre X, donde cada $I(\cdot \perp \cdot \mid \cdot) \in \mathcal{I}$ puede ser una independencia condicional, una independencia específica del contexto o una independencia instanciada. Sin embargo, debido a que cualquier supuestos de independencia puede ser expresado en términos de independencias instanciadas, entonces el conjunto $\mathcal{I}$ puede ser más concretamente definido como:

$$\mathcal{I} = \bigcup_{A,B,U \subseteq V} \bigcup_{x \in val(A \cup B \cup U)} I(x_A \perp x_B \mid x_U), \tag{2.10}$$

donde A, B y U son subconjuntos disjuntos de V; y x es un estado para las variables $X_{A \cup B \cup U}$. En otras palabras, el conjunto $\mathcal{I}$ está formado por todos las posibles independencias instanciadas sobre X, una por cada posible triplete (A, B, U) y cada posible estado x en $val(A \cup B \cup U)$.

Ahora, dado un conjunto $\mathcal{I}$, una función booleana puede ser utilizada para asignar

un valor de verdad (verdadero/falso) a cada independencia $I \in \mathcal{I}$. Diremos que una estructura S es justamente una posible relación entre supuestos de independencias y valores de verdad. Particularmente, cualquier distribución $p(X)$ puede utilizarse para definir esta relación. Por ejemplo, para cualquier $I \in \mathcal{I}$, utilizando una distribución $p(X)$ y la Definición 2.3 podemos determinar el valor de verdad de I. Dada una distribución $p(X)$, denotamos como $\mathcal{I}_{p(X)}$, al subconjunto de independencias en $\mathcal{I}$ que cumplen con la Definición 2.3, es decir,

$$\mathcal{I}_{p(X)} = \{\ I \in \mathcal{I} \colon I \text{ cumple la Definición 2.3 en } p(X)\ \}\,. \tag{2.11}$$

Diremos que $\mathcal{I}_{p(X)}$ es la estructura de $p(X)$, la cual denotaremos como $S \equiv \mathcal{I}_{p(X)} \subseteq \mathcal{I}$. Ahora, en base a los valores de verdad asignados por S al conjunto $\mathcal{I}$, las Ecuaciones (2.8) y (2.9) nos permiten obtener el valor de verdad para cualquier independencia condicional e independencia específica del contexto, respectivamente.

Ahora, vamos a definir relaciones entre estructuras. Sean S_1 y S_2 dos estructuras arbitrarias sobre $\mathcal{I}$. Si $S_1 \subseteq S_2$, entonces decimos que S_1 es un *Independency-map*, o *I-map*, de S_2. Similarmente, si $S_1 \supseteq S_2$, entonces decimos que S_1 es un *Dependency-map*, o *D-map*, de S_2. Particularmente, $S_1 \subseteq S_2$ y $S_1 \supseteq S_2$, entonces decimos que S_1 es un *perfect map* de S_2.

Ilustraremos como utilizar las relaciones previas. Asumamos que $S^\star$ es la estructura de una distribución arbitraria $p(X)$, y $\hat{S}$ es una estructura arbitraria[8] sobre $\mathcal{I}$. Si $\hat{S}$ es un I-map de $S^\star$, entonces decimos que $\hat{S}$ puede codificar "*falsas dependencias*" con respecto a $p(X)$, o *errores del tipo II*. Un error del tipo II implica que $\hat{S}$ no contiene un supuesto de independencia[9] que se satisface en $p(X)$. En contraste, si $\hat{S}$ es un D-map de $S^\star$, entonces $\hat{S}$ puede codificar "*falsas independencias*" con respecto $p(X)$, o *errores del tipo I*. Un error del tipo I implica que $\hat{S}$ contiene un supuesto de

[8]Una estructura arbitraria puede ser obtenida simplemente decidiendo aleatoriamente el valor de verdad de cada supuesto en $\mathcal{I}$; o a través de un sofisticado algoritmo, el cual decida el valor de verdad de cada independencia.

[9]Por supuesto de independencia, nos referimos a cualquier supuesto que siga las Definiciones 2.1, 2.2, o 2.3.

independencia que no se satisface en $p(X)$. Finalmente, si $\hat{S}$ es un perfect-map de $S^\star$, entonces $\hat{S}$ no contiene errores con respecto a $p(X)$.

2.2. Representación de la estructura

Para representar una estructura S, se suele asumir que la relación $I(\cdot \perp \cdot \mid \cdot)$ satisface las *propiedades de Markov* (o *axiomas de un grafo*), consultar Apéndice A para más detalles. En tal caso, se dice que una estructura S es un *graphoid* [73]. Se puede demostrar que, si S es graphoid, entonces las independencias condicionales presentes en S pueden ser representadas por un grafo G [71].

Un grafo permite codificar **independencias condicionales** a través de la *ausencia* de sus aristas. Más formalmente, se puede probar que un grafo G codifica un supuesto de independencia $I(X_A \perp X_B \mid X_U)$ si la propiedad $\text{sep}_G(A, B; U)$ se cumple en G [71, Teorema 1]. Intuitivamente, si toda interacción (o camino) entre los conjuntos A y B pasa a través del conjunto U, entonces A y B quedan separados al tener $\text{sep}_G(A, B; U)$ en G.

Generalizando lo recientemente discutido, decimos que un grafo G codifica el siguiente conjunto de supuestos de independencia sobre X:

Definición 2.4. Sea $G = (V, E)$ un grafo no dirigido. Definimos el conjunto de *independencias globales* asociadas a G, denotado por $\mathcal{I}(G)$, como

$$\mathcal{I}(G) = \{\ I(X_A \perp X_B \mid X_U)\ :\ \text{sep}_G(A, B; U)\ \}\,. \tag{2.12}$$

Esta definición muestra que un grafo G tiene el potencial para codificar todas las independencias condicionales presentes en $\mathcal{I}(G)$. Si $|V| = n$, entonces el conjunto $\mathcal{I}(G)$ puede tener como máximo $|\mathcal{I}(G)| = 4^n = 2^{2n}$ independencias condicionales. *En consecuencia, a través de un grafo, una estructura de datos polinomial, es posible codificar un número exponencial de independencias.*

Vale mencionar que, cuando una estructura es un graphoid, dicha estructura satis-

face el *principio de localidad*. Este principio establece que cualquier variable $X_a \in X$ se encuentra únicamente influenciado por un conjunto de *variables vecinas*, las cuales son ampliamente conocidas como la *manta de Markov* (*Markov blanket*) [72].

Definición 2.5. La manta de Markov de una variable $X_a \in X$ es cualquier subconjunto $S \subseteq V$ que satisface

$$I(X_a \perp X_{V \setminus (S \cup \{a\})} \mid X_S), a \notin S. \tag{2.13}$$

Una manta de Markov es llamada la *mínima manta de Markov* de X_a, si ninguno de los subconjuntos propios de dicha manta de Markov satisface la Definición 2.5. La mínimo manta de Markov es sumamente importante, debido a que puede interpretarse como el conjunto mínimo de variables que bloquea a la variable X_a de la influencia de todas las restantes variables. Como veremos más adelante, este conjunto cumple un rol protagónico al intentar aprender un grafo desde un conjunto de datos. Esto se debe a dos razones [72, Teorema 4]. Primero, se puede demostrar que, para cada variable $X_a \in X$, su mínima manta de Markov es *única*. Segundo, hablando en términos generales, la manta de Markov de una variable X_a **coincide** con el conjunto de adyacencias del nodo a. En la Sección 2.3, definiremos más formalmente este último punto, mientras tanto diremos que, dado un grafo $G = (V, E)$, la manta de Markov de un nodo $a \in V$ se corresponde con el conjunto $G(a)$.

En consecuencia, un grafo G también codifica el siguiente conjunto de supuestos de independencia:

$$\mathcal{I}^{\ell}(G) = \{\, I(X_a \perp X_{V \setminus S} \mid X_S) \colon \operatorname{sep}_G(a, V \setminus S; S) \text{ para todo } a \in V \,\}, \tag{2.14}$$

donde el conjunto $\mathcal{I}^{\ell}(G)$ es usualmente conocido como el conjunto de *independencias locales* del grafo. Si $|V| = n$, entonces el conjunto $\mathcal{I}^{\ell}(G)$ puede tener como máximo $|\mathcal{I}^{\ell}(G)| = n \sum_{i=0}^{n-1} \binom{n-1}{i} \approx n2^n$ independencias condicionales. Se puede demostrar el

teorema $\mathcal{I}(G) \implies \mathcal{I}^{\ell}(G)$. Adicionalmente, cuando una distribución de probabilidad es positiva[10], el converso del teorema anterior se cumple. Más concretamente, sea $p(X)$ una distribución *positiva*, y G un grafo I-map de $p(X)$. Entonces, el conjunto $\mathcal{I}^{\ell}(G)$ es equivalente al conjunto $\mathcal{I}(G)$ [50, Teorema 3.7]. Como veremos más adelante, este resultado es útil para codificar supuestos de independencia en un grafo, porque codificando un número polinomial de independencias locales, $|\mathcal{I}^{\ell}(G)|$, estamos codificando un número exponencial de independencias globales, $|\mathcal{I}(G)|$.

Finalmente, desde un grafo G, se puede identificar un conjunto adicional de independencias:

$$\mathcal{I}^{p}(G) = \{\ I(X_a \perp X_b \mid X_{V \setminus \{a,b\}}) \colon \operatorname{sep}_G(a, b; V \setminus \{\ a, b\ \}) \text{ para todo } a, b \in V\ \}\,. \tag{2.15}$$

Estas independencias son llamadas el conjunto de *independencias por parejas* del grafo G. En palabras, X_a es condicionalmente independiente de X_b dadas todas las restantes variables del dominio, $X_{V \setminus \{a,b\}}$, si la arista (a, b) no existe en G. Si $|V| = n$, entonces el conjunto $\mathcal{I}^{p}(G)$ puede tener como máximo $|\mathcal{I}^{p}(G)| = \binom{n}{2} \approx n^2$ independencias condicionales.

Tal como se muestra en [50, Teorema 3.7], si $p(X)$ es positiva, tenemos el siguiente resultado

$$\mathcal{I}(G) \iff \mathcal{I}^{\ell}(G) \iff \mathcal{I}^{p}(G). \tag{2.16}$$

Quizás, uno de los hechos más relevantes de la Ecuación (2.16) es que simplemente determinado si cada par de nodos (a, b) en un grafo G están o no conectados nos garantiza codificar todas las posibles independencias en el conjunto $\mathcal{I}(G)$. Como veremos más adelante, en la Sección 2.3, la Ecuación (2.16) ofrece las bases para definir un método para construir un grafo que codifique todas las independencias condicionales

[10]Una distribución $p(X)$ es positiva, si $p(x) > 0$ para todo estado $x \in val(V)$.

presentes en una distribución.

Usando las independencias codificadas por un grafo G, podemos factorizar una distribución de Boltzmann. Sea $p(X)$ una distribución de Boltzmann sobre X (Eq. (2.1)), y sea $G = (V, E)$ un grafo no dirigido, entonces $p(X)$ *factoriza* sobre G si

$$p(X = x) = \frac{1}{Z} \prod_{k:\ C \in \mathcal{C}(G)} \phi_k(x_C). \tag{2.17}$$

En palabras, la topología de G determina el conjunto de factores de la distribución. Más concretamente, cada clique máximo $C \in \mathcal{C}(G)$ determina el alcance de un determinado factor $\phi_k(X_C)$, por ende, habrá tantos factores como cliques máximos en $\mathcal{C}(G)$.

Según [46, Teorema 4.1], si $p(X)$ factoriza sobre G, entonces G es un I-map de $p(X)$[11]. El converso de este teorema, demostrado por Hammersley y Clifford a principios de la década de los 70 [39], es conocido como el "teorema de Hammersley-Clifford". Sea $p(X)$ una distribución *positiva*[12], si G es un I-map de $p(X)$, entonces $p(X)$ es una distribución de Boltzmann que factoriza sobre G.

Por lo tanto, si el grafo G no es un I-map de $p(X)$, codifica errores del tipo I (independencias falsas), entonces al factorizar $p(X)$ sobre G, el producto de factores no es igual a $p(X)$. Es decir, la *factorización es incorrecta*. En contraste, errores del tipo II (dependencias falsas) no producen factorizaciones incorrectas [72]. Sin embargo, la presencia de errores del tipo II impacta negativamente en los objetivos de estimación de densidad y descubrimiento de conocimiento (dichos objetivos fueron descriptos en el Capítulo 1).

Para ver esto último, es útil considerar que un error del tipo II implica una arista adicional, o *arista espuria*, en el grafo. Una arista espuria produce superconjuntos en algunos de los cliques máximos del grafo, agregando *parámetros espurios* en algunos factores. Como se muestra en [76], redes de Markov con parámetros espurios son propensas a tener baja precisión predictiva, afectando el objetivo de estimación de

[11] G es un I-map de $p(X)$ si todo supuesto $I \in \mathcal{I}_G$ se satisface en $p(X)$.

[12] Si una distribución de Boltzmann es positiva, entonces todas sus funciones paramétricas deben ser positivas.

densidad. Similarmente, en el caso de descubrimiento de conocimiento, una arista espuria *oculta* "independencias relevantes para un intérprete humano" [32].

Es importante notar que el simple hecho de utilizar un grafo como representación de la estructura de una distribución puede implicar errores del tipo II. Esta situación sucede cuando pretendemos representar con un grafo la estructura de una distribución $p(X)$ que presenta independencias específicas del contexto. En base a lo discutido anteriormente, al utilizar un grafo como representación, solamente podemos codificar las independencias condicionales presentes en $p(X)$, pero no así sus independencias específicas del contexto.

Para ver más en detalle lo anterior, consideremos que tenemos un grafo G que representa la estructura de la distribución $p(X)$ mencionada anteriormente, ¿cómo interpretamos las independencias falsas codificadas en G (i.e., sus dependencias)? Más concretamente, según la Ecuación (2.8), el valor de verdad de cualquier independencia condicional está en función de una conjunción de valores de verdad de independencias específicas del contexto. Así, si una independencia condicional es falsa en el grafo G, esto puede deberse a dos motivos: i) todas las independencias específicas del contexto involucradas en la conjunción son falsas; o ii) *al menos una de las independencias específicas del contexto en la conjunción es falsa* (lo cual hace falsa a una independencia condicional). Precisamente, en éste último caso, la utilización de un grafo como representación introduce errores del tipo II.

Ilustremos lo anterior a través de un ejemplo más concreto. Supongamos una distribución $p(X_a, X_b, X_w)$ donde la independencia condicional $I(X_a \perp X_b \mid X_w)$ es falsa. Sin embargo, esta independencia es falsa porqué, al descomponerla con la Ecuación (2.8), obtenemos:

$$I(X_a \perp X_b \mid X_w) = I(X_a \perp X_b \mid X_w = 0) \wedge I(X_a \perp X_b \mid X_w = 1),$$

donde el supuesto $I(X_a \perp X_b \mid X_w = 1)$ es falso en $p(X)$, mientras que $I(X_a \perp X_b \mid X_w = 0)$ es verdadero en $p(X)$. Al codificar $I(X_a \perp X_b \mid X_w)$ en un grafo G, solamente podemos codificar que las variables X_a y X_b son condicionalmente independiente de

la variable X_w. Esta codificación produce una pérdida de información, debido a que X_a y X_b son en realidad independientes en el contexto $X_w = 0$.

En la práctica, para hacer frente a tales situaciones, suele utilizarse otras representaciones alternativas de la estructura. Una de las más ampliamente utilizadas es introducida a continuación.

2.2.1. Modelos log-linear

Una representación alternativa para la estructura es un *conjunto $\mathcal{F}$ de características*. Una característica, denotada por $f_C^k \in \mathcal{F}$, representa una interacción (o dependencia) entre las variables X_C en el contexto x_C^k, donde $x_C^k \in val(C)$. Usualmente, esta interacción entre las variables es codificada mediante una *función indicador* (o *delta de Kronecker*). De este modo, una característica f_C^k es formalmente definida como sigue:

$$f_C^k(x_U) \equiv \mathbf{1}(x_{C\cap U}, x_{C\cup U}^k) = \begin{cases} falso & \text{si } x_{C\cap U} \neq x_{C\cup U}^k, \\ verdadero & \text{si } x_{C\cap U} = x_{C\cup U}^k; \end{cases}$$

donde x_u es un estado arbitrario en $val(U)$ y $x_C^k \in val(C)$.

Veamos un ejemplo de una característica. Dado el conjunto $V = \{ a, b, c \}$ y una característica f_C^k, donde $x_C^k = (X_a = 0 \wedge X_b = 1)$, entonces, ante las siguientes situaciones, dicha característica retorna los siguientes valores:

- $f_C^k(X_a = 0 \wedge X_b = 1 \wedge X_c = 0) = verdadero$,
- $f_C^k(X_a = 0 \wedge X_b = 1 \wedge X_c = 1) = verdadero$,
- $f_C^k(X_b = 0) = verdadero$,
- $f_C^k(X_b = 1) = falso$.

Usando un conjunto de características $\mathcal{F}$ sobre X, y un conjunto de parámetros

w, una distribución $p(X)$ puede ser definida como sigue:

$$p(x) = \frac{1}{Z} \exp\left\{ \sum_{j:\, f_C^k \in \mathcal{F}} w_j f_C^k(x) \right\}, \tag{2.18}$$

donde $w_j \in w$ es el parámetro asociado a la j-ésima característica en $\mathcal{F}$, el cual es usualmente llamado el *peso* (weight) de la característica. La Ecuación (2.18) es ampliamente conocida como *modelo log-linear*.

Para entender cómo un conjunto $\mathcal{F}$ puede codificar supuestos de independencias, mostraremos la relación que existe entre una distribución de Boltzmann y un modelo log-linear. Primero, mostraremos como obtener un modelo log-linear desde una distribución de Boltzmann. Segundo, mostraremos como obtener una distribución de Boltzmann desde un modelo log-linear. Una consecuencia del primer resultado es que un grafo puede ser expresado como un conjunto de características, mientras que una consecuencia del segundo resultado es que, desde un conjunto de características, puede ser inducido un grafo. Finalmente, utilizaremos los resultados previos para mostrar como un conjunto de características codifica supuestos de independencia.

Sea $p(X)$ una distribución positiva. Sea G un grafo I-map de $p(X)$. Entonces, un modelo log-linear puede ser obtenido desde $p(X)$ de la siguiente manera:

$$p(x) = \frac{1}{Z} \prod_{i:\, C \in \mathcal{C}(G)} \phi_i(x_C),$$

utilizando la igualdad $p(x) = \exp(\log(p(x)))$,

$$\begin{aligned} p(x) &= \frac{1}{Z} \exp\left\{ \log\left\{ \prod_{i:\, C \in \mathcal{C}(G)} \phi_i(x_C) \right\} \right\}, \\ &= \frac{1}{Z} \exp\left\{ \sum_{i:\, C \in \mathcal{C}(G)} \log \phi_i(x_C) \right\}. \end{aligned}$$

Debido a que una función paramétrica $\phi_i(X_C)$, representada como una tabla, puede

ser expresada como una combinación lineal de funciones indicador (características) definidas sobre cada entrada, donde el peso de cada característica es definido como $w_C^k = \log \phi_C(x_C^k)$. Entonces la ecuación anterior es equivalente a:

$$\begin{aligned} p(x) &= \frac{1}{Z} \exp \left\{ \sum_{C \in \mathcal{C}(G)} \sum_{k=1}^{2^{|C|}} w_C^k f_C^k(x) \right\}, \\ &= \frac{1}{Z} \exp \left\{ \sum_{j \colon f_C^k \in \mathcal{F}} w_j f_C^k(x) \right\}. \end{aligned}$$

Ahora, mostraremos como obtener una distribución de Boltzmann desde un modelo log-linear. Dado el siguiente modelo log-linear definido sobre $\mathcal{F}$

$$p(X) = \frac{1}{Z} \exp \left\{ \sum_{j \colon f_C^k \in \mathcal{F}} w_j f_C^k(x) \right\},$$

entonces, por cada característica f_C^k, podemos definir una función paramétrica $\psi_j(X_C)$ cuyo alcance es X_C. Los parámetros de $\psi_j(X_C)$ son todos ceros excepto el parámetro correspondiente a x_C^k, cuyo valor es w_j. Esto se debe al hecho de que un modelo log-linear puede estar formado por características adicionales al conjunto $\mathcal{F}$ cuyo pesos son cero. En base a lo anterior, obtenemos

$$p(x) = \frac{1}{Z} \exp \left\{ \sum_{j \colon f_C^k \in \mathcal{F}} \psi_j(x_C) \right\},$$

aplicando la igualdad $\log(\exp(\psi_j(X_C))) = \log \phi_j(X_C)$ obtenemos

$$p(x) = \frac{1}{Z} \exp \left\{ \sum_{j \colon f_C^k \in \mathcal{F}} \log \phi_j(x_C) \right\}.$$

Ahora, sumando aquellos potenciales cuyo alcance es un subconjunto del alcance de otro potencial, podemos obtener una nueva sumatoria de potenciales, donde el alcance de cada potencial no es un subconjunto del alcance de los restantes potenciales.

$$\begin{aligned} p(x) &= \frac{1}{Z} \exp\left\{ \sum_{i\colon C \in \mathcal{C}(\mathcal{F})} \log \phi_i(x_C) \right\}, \\ &= \frac{1}{Z} \exp\left\{ \log\left\{ \prod_{i\colon C \in \mathcal{C}(\mathcal{F})} \phi_i(x_C) \right\} \right\}, \\ &= \frac{1}{Z} \prod_{i\colon C \in \mathcal{C}(\mathcal{F})} \phi_i(x_C), \end{aligned}$$

donde $\mathcal{C}(\mathcal{F})$ denota el conjunto de alcances. Debido a que hay un alcance por potencial, esto equivale, en una distribución de Boltzmann, a un potencial por clique máximo en un grafo. En consecuencia, utilizando los alcances $\mathcal{C}(\mathcal{F})$ podemos definir un grafo G tal que:

$$p(x) = \frac{1}{Z} \prod_{i\colon C \in \mathcal{C}(G)} \phi_i(x_C).$$

En base a lo anterior, podemos inducir un grafo G desde un conjunto $\mathcal{F}$ a través del siguiente procedimiento [51]. Dado un grafo vacío $G = (V, E = \emptyset)$, agregamos una arista (a, b) en el conjunto E, si existe una característica $f_C^k \in \mathcal{F}$ tal que $a, b \in C$. Utilizando el grafo inducido G, podemos utilizar el concepto de separabilidad entre nodos para determinar los supuestos de independencia codificados por $\mathcal{F}$. Decimos que el conjunto $\mathcal{F}$ codifica el supuesto $I(X_A \perp X_B \mid X_U)$, denotado por $\text{sep}_{\mathcal{F}}(A, B; U)$, si el grafo inducido G satisface $\text{sep}_G(A, B; U)$.

En forma más general, podemos generalizar el resultado anterior, aplicándolo a cualquier subconjunto de características. Dado un estado $x_U \in val(U)$ y un conjunto $\mathcal{F}$ de características, decimos que

$$\mathcal{F}[x_U] = \{\ f_C^k \in \mathcal{F} \colon f_C^k(x_U) = verdadero\ \}\,, \tag{2.19}$$

representa todas las interacciones en $\mathcal{F}$ asociadas al contexto x_U. Similarmente, decimos que

$$\mathcal{F}[U] = \bigcup_{x_u \in val(U)} \mathcal{F}[x_U],$$

representa el subconjunto de características en $\mathcal{F}$ que satisfacen cualquier contexto $x_U \in val(U)$[13].

Usando las definiciones previas, decimos que un conjunto $\mathcal{F}$ codifica una *independencia específica del contexto*, e.g. $I(X_A \perp X_B \mid X_U, x_W)$, lo cual denotamos con $\text{sep}_{\mathcal{F}[x_W]}(X_A, X_B; X_U)$, si $\text{sep}_G(X_A, X_B; X_U)$ se cumple en el grafo G inducido desde el subconjunto $\mathcal{F}[x_W] \subseteq \mathcal{F}$.

2.3. Aprendizaje de estructuras

Como mostramos en la Ecuación (2.3), el aprendizaje de la estructura consiste en encontrar la estructura más probable, $\hat{S}$, dentro de un espacio de estructuras $\mathcal{S}$ dado un conjunto de datos D como evidencia. Este problema puede posarse como un problema de búsqueda, donde un algoritmo de aprendizaje de estructuras simplemente es una estrategia que explora el espacio $\mathcal{S}$ con el objetivo de encontrar una solución, idealmente, la solución óptima: la estructura más probable $\hat{S}$.

Sin embargo, esta tarea no es trivial. Concretamente, el número de estructuras en el espacio $\mathcal{S}$ es exponencial en el número de variables, imposibilitando una búsqueda exhaustiva sobre $\mathcal{S}$. De este modo, para encontrar una solución, se utilizan *algoritmos de búsqueda local*. Tales algoritmos comienzan su búsqueda desde un *punto inicial* en el espacio $\mathcal{S}$. Si el punto inicial no es una solución, entonces se mueve hacia un *punto vecino*. Ahora, definiendo el punto vecino como un nuevo punto inicial, el pro-

[13]Consecuentemente, $\mathcal{F} \equiv \mathcal{F}[V]$.

cedimiento anteriormente descripto puede ser repetido. La búsqueda eventualmente finaliza cuando un punto maximiza algún *función objetivo* (i.e., el punto es considerado una "solución aceptable"), o cuando se alcanza una determinada *condición de terminación*.

En base a lo discutido, el rendimiento de un algoritmo de búsqueda local se encuentra fuertemente entrelazada, entre otras cosas, por la definición del espacio de búsqueda $\mathcal{S}$. Por definición del espacio $\mathcal{S}$, nos referimos a los siguientes dos puntos:

- cuáles son los elementos que pertenecen al espacio $\mathcal{S}$ (*extensión*); y
- cómo se relacionan entre sí dichos elementos (*organización*).

Con respecto a la extensión del espacio, primero podemos pensar que dicho espacio está formado por "todas las estructuras posibles", donde según lo visto en la Sección 2.1.1, una estructura es un conjunto de supuestos de independencia. Como también vimos, podemos establecer una relación entre dos estructuras arbitrarias $S_1, S_2 \in \mathcal{S}$. Para ilustrar esto, consideremos lo siguiente. Si la estructura S_1 codifica un superconjunto de las independencias codificadas por S_2 (i.e., S_2 es un D-map de S_1), entonces decimos que S_1 *es más específica que* S_2. Similarmente, si S_1 codifica un subconjunto de las independencias codificadas por S_2 (i.e., S_1 es un I-map de S_2), entonces S_1 *es más general que* S_2. En otras palabras, una estructura es más específica que otra, cuando al factorizar una distribución de Boltzmann (ver la Ecuación (2.17)), obtenemos un menor número de parámetros.

Esta relación entre dos estructuras es una relación *más-específico-que* (more-specific-than relation) [63]. Un conjunto de elementos que satisface dicha relación tiene un *orden parcial*. En consecuencia, el espacio S puede ser representado por un *diagrama de Hasse*. En un diagrama de Hasse, podemos identificar dos elementos extremos, o estructuras extremas en el espacio $\mathcal{S}$. Una es la estructura más general, es decir, no codifica independencias. Por ejemplo, el grafo completo (no factoriza una distribución de Boltzmann). El otro extremo es la estructura más específica, es decir, codifica todas las independencias posibles. Por ejemplo, el grafo vacío (factoriza una distribución de Boltzmann como un producto de funciones univariadas). Entre estos

dos extremos, todas las restantes estructuras se encuentran ordenadas a través de la relación *más-específico-que*.

Esta organización del espacio sugiere dos enfoques naturales para explorarlo: uno es el enfoque *top-down* y otro es el enfoque *bottom-up*. El enfoque *top-down* utiliza la estructura más general como punto inicial e iterativamente se mueve hacia una estructura más específica [87, 3, 89]. Alternativamente, un algoritmo *bottom-up* utiliza la estructura más específica como punto inicial e iterativamente se mueve hacia una estructura más general [75, 18, 78, 13, 81]. Para evitar toda duda, ilustremos estos dos enfoques.

Consideremos que cada estructura en el espacio $\mathcal{S}$ es representada como un grafo no dirigido, tal como hace cualquier algoritmos basados en restricciones. En tal caso, consideremos un grafo arbitrario $G = (V, E) \in \mathcal{S}$ como punto inicial. Siguiendo el enfoque *top-down*, la idea para encontrar el grafo solución consiste en *especializar* el grafo G. Un grafo es especializado cuando éste codifica un nuevo supuesto de independencia. Para codificar un nuevo supuesto de independencia en un grafo G, simplemente removemos alguna de sus aristas, i.e., $E = E \setminus \{ (a, b) \}$ donde (a, b) es la arista removida.

A la inversa, en el enfoque *bottom-up*, el grafo solución es obtenido a través de *generalizar* el grafo G. Un grafo es generalizado cuando éste deja de codificar un supuesto de independencia. Para que un grafo G deje de codificar un supuesto de independencia, simplemente agregamos una nueva arista a dicho grafo, i.e., $E = E \cup \{ (a, b) \}$ donde (a, b) es la nueva arista.

En base a esto, un algoritmo *top-down* se caracteriza por utilizar un grafo completo como punto inicial, el cual es especializado al remover sus aristas; mientras que, un algoritmo *bottom-up*, utiliza un grafo vacío como punto inicial, el cual es generalizado al agregar nuevas aristas. En general, porque el enfoque *top-down* comienza desde el grafo completo, el enfoque *top-down* es más propenso a errores del tipo II en comparación al enfoque *bottom-up*. Por otro lado, como el enfoque *bottom-up* comienza desde el grafo vacío, el enfoque *bottom-up* es más propenso a errores del tipo I en comparación al enfoque *top-down*. Más adelante, en las Secciones 2.4.3 y 2.4.4 pre-

sentaremos dos algoritmos basados en restricciones que siguen el enfoque *top-down* y *bottom-up*, respectivamente.

	Representaciones	
	grafo no dirigido	conjunto de características
$\lvert\mathcal{S}\rvert$	$2^{\binom{n}{2}}$	2^{3^n}

Cuadro 2.1: Tamaño del espacio de estructuras en base a la representación de la estructura.

Como vimos anteriormente, los algoritmos de aprendizaje de estructuras pueden agruparse como algoritmos de estimación de densidad o algoritmos de descubrimiento de conocimiento. Ambos grupos se caracterizan por la representación de la estructura utilizada. Los algoritmos de estimación de densidad utilizan un conjunto de características, mientras que los algoritmos de descubrimiento de conocimiento utilizan un grafo. La representación afecta la *extensión del espacio* $\mathcal{S}$. Para ilustrar este hecho, en el Cuadro 2.1, mostramos como el tamaño del espacio $\mathcal{S}$ es afectado según la representación utilizada. Por ejemplo, usando un grafo $G = (V, E)$ como representación, tenemos $\binom{n}{2}$ posibles aristas para $|V| = n$ nodos, por lo tanto el espacio $\mathcal{S}$ contiene $2^{\binom{n}{2}}$ grafos. Similarmente, usando un conjunto $\mathcal{F}$ de características, tenemos 3^n funciones indicador para n variables (i.e., cada variable individual puede aparecer o no, si aparece, entonces puede tomar dos valores), por lo tanto el espacio $\mathcal{S}$ tiene 2^{3^n} conjuntos de características.

Por lo tanto, la representación de la estructura **traza** *el límite del espacio de búsqueda*, dividiendo lo que puede ser buscado de lo que no puede ser buscado[14]. Para ver la importancia de este hecho, uno podría preguntarse: ¿qué sucede si la solución óptima (o buenas aproximaciones a la solución óptima) no se encuentran en el espacio

[14]Es importante notar que la representación de la estructura puede ser vista como un "lenguaje", es decir, el lenguaje utilizado por un algoritmo para *describir* una estructura. De esta manera, la representación de la estructura puede ser analizada desde el punto de vista de la filosofía del lenguaje. Uno de lo más grandes pensadores en dicho campo fue Ludwig Josef Johann Wittgenstein, un filósofo australiano-británico profundamente influenciado por Bertrand Russell. Wittgenstein escribió entre 1913 y 1918 el *Tractatus Logico-Philosophicus* [92]. Este trabajo ha tenido un gran impacto en la filosofía del lenguaje. En términos generales, la filosofía de Wittgenstein trató de mostrar que muchos de los problemas filosóficos no son problemas del todo, sino que son el resultado de un **mal uso del lenguaje** [9].

$\mathcal{S}$? O, alternativamente ¿qué sucede si la solución óptima y sus aproximaciones están del "otro lado"[15] del límite del espacio $\mathcal{S}$, es decir, dichas soluciones existen pero únicamente puede ser descriptas por otra representación? Estas soluciones no pueden ser obtenidas. Así, en el escenario planteado, las soluciones obtenidas por un algoritmo de búsqueda en el espacio $\mathcal{S}$ resultarán malas soluciones, debido a que justamente ningún elemento en el espacio $\mathcal{S}$ es una solución óptima (o una buena aproximación a la misma) [63, § 2]. *Es por esta razón que la decisión de cómo representar la estructura es crucial en un algoritmo de aprendizaje de estructuras.*

Para analizar como la representación afecta a un algoritmo de aprendizaje de estructuras, podemos utilizar el tradeoff *sesgo-varianza* (bias-variance tradeoff) [8]. Este tradeoff muestra que una representación introduce simultáneamente dos tipos de errores en el espacio de búsqueda: *errores de sesgo* y *errores de varianza*. Un error de sesgo implica que el espacio puede potencialmente no incluir buenas soluciones, o incluso, la solución óptima. Para establecer esto más precisamente, *si la estructura no es lo suficientemente flexible para codificar supuestos de independencia, entonces un algoritmo de aprendizaje está limitado en el tipo de soluciones que puede encontrar*. Este problema es también conocido como el *sesgo del lenguaje*, ver [64, p. 64].

Ingenuamente, uno puede pensar que una forma de abordar el sesgo del lenguaje es utilizando una representación más flexible. Sin embargo, una representación flexible introduce errores de varianza o sobreajuste. En otras palabras, al aumentar la flexibilidad, la extensión del espacio incrementa rápidamente (ver el Cuadro 2.1), aumentando la probabilidad de encontrar soluciones con baja exactitud. Adicionalmente, a medida que aumenta la flexibilidad de la representación, su simplicidad disminuye [41, § 2].

Por las razones previamente expuestas, los algoritmos de descubrimiento de conocimiento utilizan representaciones simples (por ejemplo un grafo), debido a que una representación simple facilita la interpretación de la estructura, reduciendo con-

[15]Si asumimos un espacio de estructuras donde sus elementos son conjuntos de independencias, entonces al utilizar una representación (e.g., un grafo) solamente podemos representar un subconjunto de todo este espacio. En base a esto, "el otro lado" de dicho subconjunto puede interpretarse como el complemento entre el espacio de las estructuras y el subconjunto asociado a la representación.

secuentemente el espacio de búsqueda. Por otro lado, los algoritmos de estimación de densidad utilizan representaciones flexibles (por ejemplo características), debido a que pueden obtener estructuras complejas, las cuales resultan útiles al momento de estimar distribuciones. Sin embargo, esta flexibilidad produce estructuras complejas que no son útiles para ser interpretadas (por ejemplo, por un experto humano).

En consecuencia, los algoritmos de descubrimiento de conocimiento son propensos a errores de sesgo del lenguaje, mientras que los algoritmos de estimación de densidad son propensos a errores de sobreajuste. Por ejemplo, para reducir errores de sobreajuste, los algoritmos de estimación de densidad comúnmente utilizan dos "sesgos" [18, 89, 56]: el uso de características positivas[16] y el uso de términos de regularización durante la estimación de parámetros.

2.4. Algoritmos basados en restricciones

Esta sección presenta una descripción detallada de cómo trabajan los algoritmos basados en restricciones, i.e., cómo estos algoritmos construyen un grafo solución desde un conjunto de datos muestreados desde una distribución. Para tal fin, esta sección se encuentra dividida en dos grandes partes.

La primera parte involucra las Secciones 2.4.1 y 2.4.2. En la Sección 2.4.1 describiremos, en términos generales, cómo funcionan los algoritmos basados en restricciones, es decir, cómo hace un algoritmo para construir un grafo desde un conjunto de datos. La Sección 2.4.2 se abocará a uno de los componentes claves en el diseño de cualquier algoritmo basado en restricciones: la prueba estadística de independencia.

La segunda parte involucra las Secciones 2.4.3 y 2.4.4. En estas dos últimas secciones, revisaremos en detalle dos algoritmos ilustrativos del enfoque basado en restricciones: el algoritmo PC y el algoritmo GS, respectivamente. Estos algoritmos son de sumo interés para nuestro trabajo debido a que, en el Capítulo 4, los utilizaremos como base para mostrar nuestro enfoque para aprender independencias específicas del

[16] Una característica $f_C(x)$ es positiva, si $\bigwedge_{a \in C}(x_a = 1)$. Esto reduce el espacio de estructuras de 2^{3^n} a 2^{2^n}.

contexto.

2.4.1. Construcción del grafo solución

Para construir un grafo solución desde un conjunto de datos, un algoritmo basado en restricciones procede de la siguiente manera. Primero propone un grafo inicial, el cual es modificado iterativamente hasta que eventualmente es encontrado el grafo solución. Las modificaciones efectuadas sobre un grafo pueden reducirse a dos operaciones elementales: *remover aristas*, lo cual implica agregar supuestos de independencia; y *agregar aristas*, lo cual implica remover supuestos de independencia.

Lo anterior es una manera muy simple de explicar el funcionamiento de cualquier algoritmo basado en restricciones. Pero, profundicemos esta idea. Primero y principal, ¿qué hace a un grafo, un grafo solución? En principio diremos que un grafo es solución, si dicho grafo es un I-map de $p(X)$. Como hemos comentado, un grafo G es un I-map de $p(X)$, si cada independencia en el conjunto $\mathcal{I}(G)$ se cumple en $p(X)$. Que un grafo G sea un I-map de $p(X)$, nos garantiza que G no codifica independencias falsas (i.e., el grafo no presenta errores del tipo I). Como resultado, la distribución $p(X)$ puede ser factorizada utilizando G. Y, como ya hemos mostrado, factorizar una distribución presenta varias ventajas en términos prácticos.

Sin embargo, una distribución puede tener más de un grafo I-map. Por ejemplo, un grafo completamente conectado es conocido como el I-map trivial para cualquier distribución $p(X)$, debido a que un grafo completamente conectado no codifica ninguna independencia condicional y, por lo tanto, satisface trivialmente la definición de I-map.

De este modo, diremos que un grafo es solución si el grafo es un *I-map mínimo* de $p(X)$.

Definición 2.6. Un grafo G es un *I-map mínimo* de $p(X)$, si al remover cualquier arista de G, G cesa de ser un I-map de $p(X)$.

Según nuestra discusión en la Sección 2.3, un grafo I-map mínimo puede pensarse como *aquel grafo que codifica el máximo número de independencias condicionales*

presentes en $p(X)$, es decir, buscamos un grafo G que maximice $|\mathcal{I}(G)|$ (lo cual implica una ausencia de errores del tipo II).

Como se muestra en [72, Teorema 3], toda distribución $p(X)$ positiva tiene un **único** mínimo I-map. En base a esto, el siguiente teorema muestra cómo construir el grafo mínimo I-map de una distribución $p(X)$ positiva.

Teorema 2.1. (c.f. [72, Teorema 3]). Sea $p(X)$ una distribución positiva. El grafo mínimo I-map de $p(X)$, $G_0 = (V, E_0)$, puede ser construido removiendo desde un grafo completamente conectado toda arista (a, b) tal que

$$(a, b) \notin E_0 \iff I(X_a \perp X_b \mid X_{V \setminus \{a,b\}}). \tag{2.20}$$

Demostración. Una demostración rigurosa de este teorema puede ser encontrada en [72]. En términos sencillos, la demostración de este teorema está basada en la Ecuación (2.16). Esta ecuación muestra que $\mathcal{I}(G) \iff \mathcal{I}^{\ell}(G)$, en palabras, el conjunto de independencias por parejas determina el conjunto de independencias globales (y viceversa). Precisamente, la Ecuación (2.20) determina todas las posibles independencias que pueden pertenecer al conjunto $\mathcal{I}^{\ell}(G)$, lo cual implica el conjunto $\mathcal{I}(G)$ con el mayor número de independencias. □

Este teorema muestra un método sencillo para construir un grafo solución utilizando el grafo completamente conectado[17] como grafo inicial. En palabras, para construir el grafo solución, un algoritmo basado en restricciones debe simplemente decidir cuáles de las aristas de un grafo completo remover. Dado que un grafo completo tiene $\binom{n}{2}$ aristas, la construcción del mínimo I-map involucra tomar al menos $O(n^2)$ decisiones, una por cada arista presente en un grafo completamente conectado. Como veremos en la Sección 2.4.3, el algoritmo PC se basa en el Teorema 2.1.

Por otro lado, como consecuencia del Teorema 2.1, se puede mostrar que cualquier variable $X_a \in X$ tiene una *mínima manta de Markov*, donde *la mínima manta de*

[17] Vale remarcar, que el Teorema 2.1 puede ser generalizado utilizando como grafo inicial *cualquier* grafo I-map, no necesariamente un grafo completamente conectado.

Markov de X_a se corresponde con las adyacencias del nodo a en el grafo mínimo I-map de $p(X)$ [72, Teorema 4]. Este último hecho sugiere una forma alternativa, a la mostrada en el Teorema 2.1, para construir un mínimo I-map.

Corolario 2.1. (c.f. [72, Corolario 1]). El grafo G_0 de cualquier distribución positiva $p(X)$ puede ser construido conectando cada nodo a con cada nodo b perteneciente a la mínima manta de Markov.

De este modo, identificando la mínima manta de Markov de cada variable $X_a \in X$, el Corolario 2.1 puede ser utilizado para construir G_0. Como mostraremos en la Sección 2.4.4, este es el método que sigue el algoritmo GS para construir un grafo solución.

Lo interesante del Corolario 2.1 es que el problema de construcción del grafo solución puede ser dividido en un *conjunto de subproblemas más simples y locales*. Asumiendo un ordenamiento arbitrario pero fijo de los nodos V, la construcción del grafo solución puede dividirse en $|V| = n$ subproblemas, uno por cada nodo $a \in V$. Cada subproblema consiste en identificar la mínima manta de Markov para una variable en particular.

Para identificar la mínima manta de una variable X_a, un método consistiría en proponer una manta inicial $G(a)$ a partir de la cual se agregará o removerá elementos hasta eventualmente obtener $G_0(a)$, la mínima manta de Markov. Este método hace surgir la siguiente pregunta, una vez agregado o removido un elemento desde la manta inicial $G(a)$, ¿cómo sabemos si $G(a)$ sigue siendo una manta o no? Para responder esta pregunta, podemos usar la Definición 2.5, puntualmente, la Ecuación (2.13). De este modo, en base a esto, podemos distinguir dos casos en el método para encontrar la mínima manta:

1. La manta inicial $G(a)$ satisface la Ecuación (2.13), i.e., $G(a) \supseteq G_0(a)$.

2. La manta inicial $G(a)$ no satisface la Ecuación (2.13), i.e., $G(a) \subset G_0(a)$.

En el primer caso, la identificación de la manta se reduce a remover nodos de $G(a)$ hasta que $G(a) = G_0(a)$. En términos generales, la idea aquí consiste en tomar un

nodo arbitrario $b \in G(a)$ y definir una nueva manta $G'(a) = G(a) \setminus \{ b \}$. Si $G'(a)$ satisface la Ecuación (2.13), entonces $G(a) \leftarrow G'(a)$. En caso de no satisfacer la ecuación, entonces no removemos el nodo b de $G(a)$. Repitiendo este proceso sobre cada elemento en $G(a)$, eventualmente obtendremos la mínima manta.

Sin embargo, el segundo caso no es tan sencillo como el primero. Sea V' el subconjuntos de nodos $V \setminus (G(a) \cup \{ a \})$, la identificación de la manta implica agregar uno o más nodos de V' a la manta $G(a)$ hasta que $G(a) = G_0(a)$. La idea consiste en tomar un nodo $b \in V'$ y definir una nueva manta $G'(a) = G(a) \cup \{ b \}$. Si $G'(a)$ no satisface la Ecuación (2.13), entonces $G(a) \leftarrow G'(a)$, y continuamos dicho proceso sobre los restantes nodos en V'. Por otro lado, si $G'(a)$ satisface la Ecuación (2.13), entonces $G'(a)$ es una manta de Markov. Sin embargo, no tenemos garantías de si $G(a)$ es mínimo o no. Para encontrar la mínimo manta, aplicamos el procedimiento del primer caso, utilizando como manta inicial la manta encontrada en el segundo caso.

Independientemente del método elegido para construir el grafo solución, es necesario determinar el valor de verdad de una independencia condicional. Por ejemplo, en el Teorema 2.1 es necesario determinar el valor de verdad de una independencia condicional con la forma $I(X_a \perp X_b \mid X_{V \setminus \{a,b\}})$. Por otro lado, en el Corolario 2.1, es necesario determinar el valor de verdad de una independencia condicional de la forma $I(X_a \perp X_{V \setminus (S \cup \{a\})} \mid X_S)$. Como veremos en la próxima sección, una prueba estadística de independencia puede utilizarse para determinar el valor de verdad de cualquier independencia condicional.

2.4.2. Pruebas de independencias

Como mencionamos en el capítulo anterior, los algoritmos basados en restricciones se caracterizan por utilizar una prueba estadística de independencia como mecanismo para decidir si una determinada independencia condicional se cumple en un conjunto de datos. Según el resultado obtenido al utilizar una prueba estadística, un algoritmo basado en restricciones modifica la topología de un grafo inicial con el objetivo de encontrar un grafo solución.

Un ejemplo muy bien conocido de prueba estadística de independencia es la famosa prueba χ^2 [74], ampliamente utilizada para problemas de pruebas de hipótesis. En términos generales, una prueba de independencia puede ser descripta de la siguiente manera. Dado un conjunto de datos D como evidencia, una prueba estadística de independencia, la cual denotaremos como T, es una función que mapea cualquier supuesto de independencia condicional, e.g. $I(X_a \perp X_b \mid X_C)$, a un valor de verdad (verdadero/falso) en base al conjunto D [2].

Intuitivamente, para obtener el valor de verdad de un supuesto de independencia, una prueba de independencia utiliza la definición de independencia condicional, ver la Definición 2.1. En otras palabras,

$$\mathrm{T}(I(X_a \perp X_b \mid X_C), D) = \textit{verdadero} \iff \hat{p}(X_a, X_b|X_C) \approx \hat{p}(X_a|X_C) \times \hat{p}(X_b|X_C),$$

donde $\hat{p}(\cdot)$ es una *distribución empírica* obtenida desde el conjunto D. Veamos un pequeño ejemplo de cómo obtener una distribución empírica desde un conjunto D. Asumamos que X es un conjunto de variables binarias y que queremos obtener la distribución empírica $\hat{p}(X_a = 0)$, $X_a \in X$. Sea D una muestra de $p(X)$, entonces $\hat{p}(X_a = 0)$ puede ser obtenida desde D del siguiente modo:

$$\hat{p}(X_a = 0) = \frac{1}{|D|} \sum_{x \in D} \mathbf{1}(x_a, 0), \tag{2.21}$$

donde $\mathbf{1}(\cdot, \cdot)$ es una función indicador que retorna 1 cuando sus dos argumentos son iguales, o cero en otro caso. De este modo, la distribución empírica $\hat{p}(X_a = 0)$ puede ser pensada como una aproximación de la distribución $p(X_a = 0)$. Se puede demostrar que a medida que el tamaño de D aumenta, la distribución empírica $\hat{p}(x_a)$ tiende a ser la verdadera distribución $p(x_a)$.

Una distribución empírica básicamente consiste en una *tabla de contingencia* nor-

malizada por $\frac{1}{|D|}$. Por ejemplo, la tabla de contingencia para $\hat{p}(X_a, X_b|X_C)$[18] puede descomponerse en $|val(C)| = 2^{|X_C|}$ subtablas, una por cada estado en $val(C)$, donde cada subtabla tiene $|val(a)|$ filas y $|val(b)|$ columnas, resultando en $2^{|val(\{a\}\cup\{b\}|}$ celdas. En una subtabla, sus celdas están asociadas a cada uno de los posibles estados $val(\{ a \} \cup \{ b \})$ y almacenan el número de veces (frecuencia) en que dicho estado aparece en D para un estado fijo de X_C.

Una desventaja de utilizar pruebas estadísticas de independencia es que la correctitud de sus resultados depende del número de datos, el cual resulta exponencial con respecto al número de variables involucradas en el supuesto de independencia [12, 20]. Por ejemplo, una regla común en la prueba χ^2 es que la frecuencia de cada celda de una tabla debe involucrar 5 o más observaciones en D para evitar resultados imprecisos.

2.4.3. El algoritmo PC

Este algoritmo fue primeramente presentado para redes de Bayes [87]. El nombre del algoritmo proviene de las primeras letras de los nombres de sus autores: Peter y Clark. Originalmente, el algoritmo PC construye un grafo dirigido (requerido por una red de Bayes) en dos etapas. Primero, el algoritmo construye un grafo no dirigido, formalmente conocido como "grafo esqueleto", y entonces obtiene un grafo dirigido, asignando una dirección a cada arista en el esqueleto. Según las direcciones agregadas, algunas aristas pueden ser removidas.

Afortunadamente, el algoritmo PC puede ser fácilmente adaptado para aprender la estructura de una red de Markov. Esto se debe a que el grafo esqueleto de una red de Bayes equivale a la estructura de una red de Markov [46, § 3.3.4]. De esta manera, debido al diseño modular del algoritmo PC, removiendo su segunda etapa, logramos que dicho algoritmo aprenda la estructura de una red de Markov. Por esta razón, denominaremos como "algoritmo PC" a la primera etapa del algoritmo PC original.

El algoritmo PC es formalmente mostrado en el Algoritmo 1. Dicho algoritmo tiene dos parámetros de entrada: una prueba estadística de independencia T y un conjunto de datos D. La salida del algoritmo es un grafo (solución) que codifica los

[18] Recordar que, a menos que se diga lo contrario, asumimos variables binarias.

Algoritmo 1 El algoritmo PC

procedure PC(T, D)
  $V \leftarrow (1, \ldots, n)$, donde n es el numero de columnas en D
  $G \leftarrow (V, E)$, $E = V \times V \setminus \{ (a, a) : a \in V \}$
  $k \leftarrow 0$
  repeat
    for (a, b) in E **do**
      for any $U \subseteq G(a) \setminus \{ b \}$ s.t. $|U| = k$ **do**
        if T($I(X_a \perp X_b \mid X_U)$, D) = verdadero **then**
          $E \leftarrow E \setminus \{ (a, b) \}$
    $k \leftarrow k + 1$
  until $|G(a)| \leq k$, for all $a \in V$
  return G

supuestos de independencias condicionales presentes en el conjunto D.

Para encontrar el grafo solución, el algoritmo PC utiliza el enfoque *top-down*. En consecuencia, el algoritmo PC define, como grafo inicial, el grafo completo. Luego, en base a este grafo inicial, el algoritmo PC remueve sucesivamente sus aristas hasta eventualmente obtener el grafo solución. Para decidir qué aristas remover, el algoritmo PC utiliza el Teorema 2.1. En otras palabras, el algoritmo PC intenta remover "dependencias falsas", o aristas espurias[19], en el grafo inicial.

Sin embargo, debido a que el grafo inicial es el grafo completo, determinar el valor de verdad para cualquier supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$ requiere un número exponencial de datos con respecto al número de variables involucradas en el supuesto. Esto se debe a que la manta de todo nodo $a \in V$ contiene todos los restantes nodos, i.e. $G(a) = V \setminus \{ a \}$, donde $|G(a)|$ es $|V| - 2$ inicialmente. Como resultado, la tabla de contingencia asociada al supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$ involucra $2^{|V|-2} \times 4$ celdas, requiriendo un número exponencial de datos.

Para evitar el problema anterior, el algoritmo PC usa un *enfoque incremental* para verificar cada arista en el grafo. En términos simples, esto consiste en tomar subconjuntos de tamaño incremental del conjunto condicionantes de cada supuesto de independencia. Para lograr esto se utiliza una variable umbral k, la cual determina

[19] Dado un grafo $G = (V, E)$, una arista $(a, b) \in E$ es espuria, si el supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$ es verdadero.

el tamaño máximo de todo conjunto condicionante. Por ejemplo, si $k = 4$, entonces, para todo supuesto de independencia $I(X_a \perp X_b \mid X_U)$, $|U| \leq 4$ donde $U \subseteq G(a)$. La variable umbral k es inicializada en 0 y es incrementada en 1 cada vez que todas las aristas fueron revisadas. El enfoque incremental termina (o alcanza la condición de terminación) cuando todas las mantas en el grafo tienen menos o igual que k nodos, i.e., $|G(a)| \leq k$ para todo $a \in V$.

Sin embargo, el enfoque incremental presenta una gran desventaja. Para decidir remover una arista, es necesario verificar más de un supuestos. Para ser más concreto, sin el enfoque incremental, el algoritmo PC obtiene el grafo solución tomando sólo $O(n^2)$ decisiones, es decir, un supuesto de independencia por arista. Sin embargo, al utilizar el enfoque incremental, cada arista tiene asociado un número exponencial de supuestos en el peor caso. Analicemos esto último. Sea $G(a)$ la manta del nodo $a \in V$. El peor caso sucede cuando $G(a) = V \setminus \{ a \}$. En tal caso, para cualquier $b \in G(a)$, el enfoque incremental produce, para la arista (a, b), un supuesto de independencia por cada subconjunto de $G(a)$. En otras palabras, hay un supuesto por cada elemento en el conjunto poder de $G(a)$. El número de elementos en el conjunto poder es igual a $\sum_{i=0}^{|G(a)|} \binom{|G(a)|}{i} = 2^{|G(a)|}$.

Una forma (sencilla) de superar este problema, consiste en modificar la condición de terminación del algoritmo para terminar antes su búsqueda. Por ejemplo, usando un valor máximo predefinido para la variable umbral k, denotado por $K_{\max}$. En otras palabras, la nueva condición de terminación sería: $|G(a)| \leq K_{\max} \vee |G(a)| \leq k$ para todo $a \in V$.

Sin embargo el uso de un valor máximo $K_{\max}$ puede ocasionar que el algoritmo PC obtenga un grafo con errores del tipo II. Para ilustrar esto, consideremos que la estructura subyacente es un grafo $G^\star$ tal que $k^\star$ es el grado máximo de $G^\star$. Ahora, supongamos que $K_{\max}$ toma un valor arbitrario entre 0 y $|V|$. Entonces existen tres escenarios de terminación para el algoritmo: (i) si $k^\star < K_{\max}$, entonces la condición $|G(a)| < k$ es alcanzada primero, sin ocasionar errores; (ii) si $k^\star = K_{\max}$, la condición $|G(a)| < k$ es alcanzada primero; y (iii) si $k^\star > K_{\max}$, el grafo solución puede contener errores del tipo II, porque no se verificaron todos los supuestos de una arista. A pesar

de esto, independientemente del valor elegido para $K_{\max}$, el algoritmo PC encontrará siempre un grafo I-map (sin errores del tipo I).

Para finalizar, analizaremos la complejidad temporal del algoritmo PC. Esta complejidad es proporcional al número de decisiones tomadas para obtener el grafo solución. Para este análisis asumiremos que el costo de cada decisión es uniforme. Este supuesto no cambia nuestra conclusiones sobre la complejidad temporal, debido a dicha complejidad está expresada en términos del número de decisiones tomadas por el algoritmo. Como dijimos anteriormente, sin el enfoque incremental, el algoritmo PC toma $\binom{n}{2} \approx n^2$ decisiones, mientras que, al usar el enfoque incremental y asumiendo que el grado máximo del grafo es $k^\star$, entonces por cada arista es necesario tomar $\sum_{i=0}^{k^\star} \binom{n-1}{i}$ decisiones en el peor caso. Para resumir, la complejidad temporal del algoritmo PC es

$$O\left(n^2 \sum_{i=0}^{k^\star} \binom{n-1}{i}\right). \tag{2.22}$$

Ejemplo 2.1. Veamos un ejemplo de cómo el algoritmo PC construye un grafo solución G desde un conjunto de datos. Sin pérdida de generalidad, asumiremos que dicho algoritmo utiliza, en vez de una prueba de independencia, un *oráculo* que "conoce" la estructura subyacente de la distribución que generó al conjunto de datos. Consultando al oráculo, se puede obtener el valor de verdad correcto para cualquier supuesto de independencia. Para este ejemplo, la estructura subyacente de la distribución es el grafo $G^\star$ mostrado en la Figura 2-3.

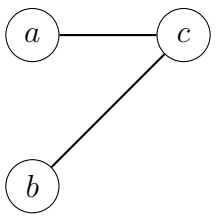
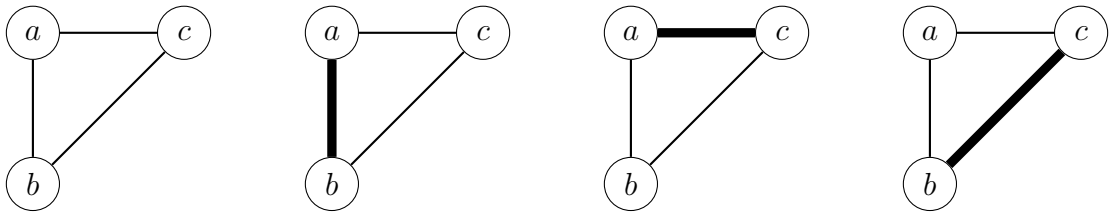


Figura 2-3: El grafo $G^\star$ de una distribución $p(X)$ desconocida.

Este grafo codifica un único supuesto de independencia: $I(X_a \perp X_b \mid X_c)$. Esto

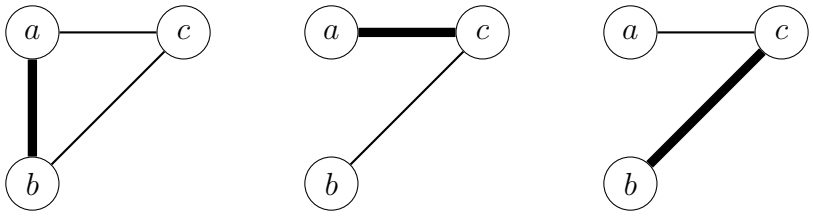
es porque tenemos que $\text{sep}_{G^\star}(a, b; c)$, es decir, no existe ningún camino del nodo a al nodo b sin pasar por el nodo c.



(a) Grafo inicial. (b) La arista (a, b) es revisada. (c) La arista (a, c) es revisada. (d) La arista (b, c) es revisada.

Figura 2-4: Pasos ejecutados por el algoritmo PC sobre un grafo inicial con un valor de k igual a 0. Una arista en negrita indica que dicha arista está siendo evaluada por el algoritmo PC.

Para construir el grafo G, el algoritmo PC identifica cuáles aristas remover en un grafo completo (Figura 2-4). Para esto, el algoritmo PC inicializa la variable k con cero y verifica cada arista en el grafo. Debido a que la variable $k = 0$, toda independencia condicional propuesta tendrá cero variables condicionales, i.e., todas las independencias serán independencias marginales. Por ejemplo, la arista (a, b) no puede ser removida porque la independencia marginal $I(X_a \perp X_b \mid \emptyset)$ es falsa debido a que $\text{sep}_{G^\star}(a, b; \emptyset)$ es falso. La arista (a, c) tampoco puede ser removida porque $I(X_a \perp X_c \mid \emptyset)$ es falsa debido a que $\text{sep}_{G^\star}(a, c; \emptyset)$ es falso. La última arista (b, c) tampoco puede ser removida porque $I(X_b \perp X_c \mid \emptyset)$ es falsa debido a que $\text{sep}_{G^\star}(b, c; \emptyset)$ es falso.



(a) La arista (a, b) es revisada. (b) La arista (a, c) es revisada. (c) La arista (b, c) es revisada.

Figura 2-5: Pasos ejecutados por el algoritmo PC sobre un grafo G con $k = 1$.

Una vez revisada todas las aristas del grafo, el algoritmo PC incrementa la variable k, la cual pasa a tomar el valor 1. Ahora, el algoritmo PC decide si continuar o

detenerse. Para esto evalúa la condición de parada, debido a que $|G(a)| = |G(b)| = |G(c)| = 2$ y 2 no es menor igual a 1, entonces PC decide continuar. En esta nueva iteración, toda independencia propuesta estará condicionada por otra variable. Esto es ilustrado en la Figura 2-5.

Por ejemplo, la arista (a, b) puede ser removida porque $I(X_a \perp X_b \mid X_c)$ es verdadera debido a que $\text{sep}_{G^\star}(a, b; c)$ es verdadero, i.e., el nodo a no puede alcanzar al nodo b sin pasar por el nodo c. Luego, son revisadas las aristas (a, c) y (b, c). Ninguna de estas aristas puede ser removida porque $I(X_a \perp X_c \mid \emptyset)$ y $I(X_b \perp X_c \mid \emptyset)$ son falsas.

Finalmente, la variable k es incrementada, y pasa a tomar el valor 2. Debido a que $|G(a)| = |G(b)| = 1$, lo cual es menor igual a 2, y $|G(c)| = 2$, lo cual es menor igual a 2, el algoritmo PC finaliza y retorna el grafo mostrado en la Figura 2-5c. Al comparar este grafo con el grafo mostrado en la Figura 2-3, podemos verificar que el algoritmo PC construyó el grafo subyacente.

□

2.4.4. El algoritmo GS

Al igual que el algoritmo PC, el algoritmo GS fue primeramente presentado para redes Bayesianas y su diseño también involucra dos etapas: la primera para obtener un esqueleto y la segunda para orientar sus aristas [61]. Dado nuestro interés en redes de Markov, solamente estamos interesados en el primer paso (el cual a partir de aquí llamaremos como "el algoritmo GS"). En la literatura, podemos encontrar una adaptación a redes de Markov del algoritmo GS denominado el *algoritmo GSMN* [13]. Comparado al algoritmo GS, el algoritmo GSMN presenta varias ideas interesantes para mejorar el rendimiento. Por ejemplo, una de ellas es un resultado teórico llamado el *teorema del triángulo*. Utilizando dicho teorema, el algoritmo GSMN puede inferir el valor de verdad de un supuesto de independencia utilizando los valores de verdad de otros supuestos, reduciendo el costo temporal para encontrar el grafo solución. Sin embargo, para los fines de este trabajo, sólo analizaremos el algoritmo GS básico, el cual es mostrado en el Algoritmo 2.

El algoritmo GS tiene dos parámetros de entrada: una prueba estadística T y un

Algoritmo 2 El algoritmo GS

```
1: procedure GS(T, D)
2:     V ← (1, ..., n), donde n es el numero de columnas en D
3:     G ← (V, E), E = ∅
4:     for a in V do
5:         G ← GROW(T, G, a, D)
6:         G ← SHRINK(T, G, a, D)
       return G
```

conjunto de datos D. La salida es un grafo (solución) que codifica independencias condicionales presentes en D. Para encontrar el grafo solución, el algoritmo GS sigue el enfoque *bottom-up*. En consecuencia, el grafo inicial es el grafo vacío[20] $G = (V = \{ 1, \ldots, n \}, E = \emptyset)$.

En términos generales, el Algoritmo 2 sigue el Corolario 2.1. Divide el problema de construir el grafo solución en n subproblemas, uno por cada *nodo objetivo* $a \in V$. Sea a un nodo objetivo en V, la manta $G(a)$ es identificada en dos etapas: la *fase de crecimiento* (grow phase) y la *fase de encogimiento* (shrink phase)[21].

En la primera etapa, intenta agregar nodos a la manta (la manta "crece"), lo cual implica agregar aristas en el grafo inicial. Un problema de intentar agregar aristas en un grafo con errores tipo I (i.e., las mantas del grafo inicial no satisfacen la Ecuación (2.13)) es que se puede incurrir en errores del tipo II, es decir, el algoritmo puede potencialmente agregar aristas espurias (para más detalles, ver Ejemplo 2.2). Como resultado, las mantas encontradas pueden potencialmente contener nodos "falsos" [61], es decir, la manta encontrada no es la mínima. En la segunda fase se remueven aristas (espurias), lo cual "encoge" la manta.

Algoritmo 3 La fase de crecimiento

```
1: procedure GROW(T, G, a, D)
2:     U ← { b ∈ V \ { a } : b ∉ G(a) }
3:     for b ∈ U do
4:         if T(I(X_a ⊥ X_b | X_{G(a)}), D) = falso then
5:             E ← E ∪ { (a, b) }
       return G
```

[20] Un grafo vacío $G = (V, E = \emptyset)$ implica que, para todo $a \in V$, $G(a) = \emptyset$.

[21] El "GS" del nombre del algoritmo justamente proviene de las letras iniciales de cada fase.

El Algoritmo 3 muestra la fase de crecimiento. Esta fase tiene como parámetros de entrada: una prueba estadística T, un grafo inicial G, un nodo objetivo a y un conjunto de datos D. Para agregar nodos a la manta $G(a)$, se define un conjunto U de nodos candidatos. Este conjunto está formado por todos aquellos nodos que no pertenecen a la manta $G(a)$. Un nodo candidato $b \in U$ es agregado a $G(a)$, si el supuesto $I(X_a \perp X_b \mid X_{G(a)})$ no se cumple en el conjunto D. En tal caso, la arista (a, b) es agregada en el grafo, con lo cual el nodo b es agregado a la manta $G(a)$ y el nodo a a la manta $G(b)$.

Algoritmo 4 La fase de encogimiento

```
1: procedure SHRINK(T, G, a, D)
2:     for b ∈ G(a) do
3:         if T(I(X_a ⊥ X_b | X_{G(a)\{b}}), D)=verdadero then
4:             E ← E \ { (a, b) }
       return G
```

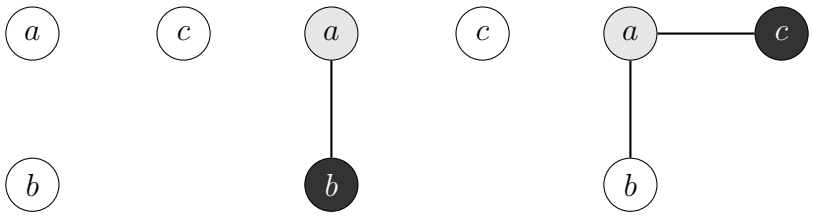
El Algoritmo 4 muestra la fase de encogimiento. Esta fase tiene como parámetros de entrada: una prueba estadística T, un grafo inicial G, un nodo objetivo a, y conjunto de datos D. Un nodo b en la manta $G(a)$ es removido, si el supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$ se cumple en el conjunto D. En tal caso, la arista (a, b) es removida del grafo, con lo cual el nodo b es removido de la manta $G(a)$ y el nodo a de la manta $G(b)$.

Para finalizar, analicemos la complejidad temporal del algoritmo GS. Sea n el número de nodos. El algoritmo GS tiene que identificar n mantas, por lo tanto, la complejidad temporal es proporcional a n. Para identificar una manta, se toman dos conjuntos de decisiones: las decisiones de la fase de crecimiento y las decisiones de la fase de encogimiento.

La fase de crecimiento toma $(n-1)$ decisiones en el peor caso, es decir, para la primer manta $G(a)$, hay $n - 1 = |U|$ nodos candidatos. Ahora, asumamos que la fase de crecimiento agrega como máximo $k^\star = |G(a)|$ nodos. Entonces, la fase de encogimiento toma $k^\star$ decisiones, es decir, un decisión para cada uno de los nodos en la manta. Resumiendo, la complejidad del algoritmo GS es

$$O(n((n-1)+k^{\star})). \tag{2.23}$$

Ejemplo 2.2. Veamos un ejemplo de cómo el algoritmo GS construye un grafo G. Al igual que el Ejemplo 2.1, sin pérdida de generalidad, asumiremos que el algoritmo GS utiliza un oráculo y que la estructura subyacente es el grafo $G^{\star}$ mostrado en la Figura 2-3. Para construir el grafo G, el algoritmo GS identifica la manta de cada nodo objetivo utilizando la fase de crecimiento y la fase de encogimiento. Asumamos que los nodos en V siguen un ordenamiento lexicográfico, en consecuencia, el nodo a es el primer nodo objetivo, el siguiente nodo objetivo es b y el nodo c es el último nodo objetivo. Dado el nodo a, los pasos de la fase de crecimiento son ilustrados en la Figura 2-6.

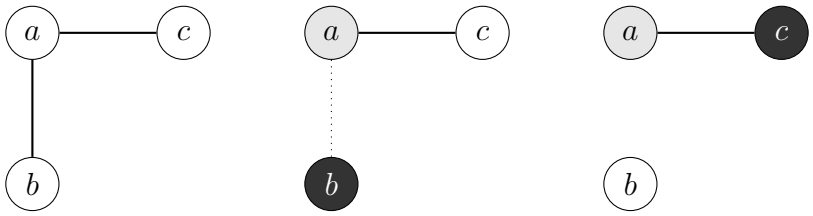


(a) Grafo inicial. (b) La arista (a, b) es agregada a G. (c) La arista (a, c) es agregada a G.

Figura 2-6: Pasos ejecutados por la fase de crecimiento sobre un grafo inicial. Un nodo en gris claro indica un nodo objetivo. Un nodo en gris oscuro indica nodo candidato.

Para agregar nodos a la manta $G(a)$, la fase de crecimiento define el conjunto U de nodos candidatos. Dado que el grafo inicial es el grafo vacío, entonces $G(a) = \emptyset$, y el conjunto $U = V \setminus G(a) \cup \{\ a\ \} = \{\ b, c\ \}$. Siguiendo el ordenamiento de nodos, primero se decide si el nodo $b \in U$ pertenece a la manta $G(a)$. Para tomar dicha decisión, la fase consulta al oráculo por el valor de verdad del supuesto $I(X_a \perp X_b \mid \emptyset)$. El oráculo responde que dicho supuesto es falso, porque $\text{sep}_{G^{\star}}(a, b; \emptyset)$ es falso, debido a la presencia del camino $a - c - b$. Como resultado, la arista (a, b) es agregada al grafo como muestra la Figura 2-6b. La nueva arista modifica la manta $G(a)$ tal que $G(a) = \{\ b\ \}$.

A continuación, la fase de crecimiento decide si el próximo nodo, $c \in U$, pertenece a la manta $G(a)$. Al consultar el valor de verdad del supuesto $I(X_a \perp X_c \mid X_b)$, el oráculo responde que es falso, porque $\text{sep}_{G^\star}(a, c; b)$ es falso, debido al camino $a - c$. En consecuencia, como muestra la Figura 2-6c, la arista (a, c) es agregada al grafo y la manta $G(a)$ toma el valor $\{\, b, c \,\}$. Al no existir un próximo nodo en U, la fase de crecimiento finaliza y retorna el grafo G mostrado en la Figura 2-6c. Al comparar el grafo G frente al grafo subyacente $G^\star$, podemos ver que la manta $G(a)$ es un superconjunto de la manta $G^\star(a)$, es decir, la arista (a, b) en G es espuria (no está en $G^\star$).



(a) Grafo inicial.

(b) La arista (a, b) es removida de G.

(c) La arista (a, c) no es removida de G.

Figura 2-7: Pasos ejecutados por la fase de encogimiento sobre un grafo inicial. Un nodo en gris claro indica un nodo objetivo. Un nodo en gris oscuro indica un nodo candidato.

Usando el grafo retornado por la fase de crecimiento, la fase de encogimiento ejecuta los pasos ilustrados en la Figura 2-7. Básicamente, la fase de encogimiento decide cuáles nodos en la manta $G(a)$ son falsos. Siguiendo el ordenamiento, primero se decide si el nodo $b \in G(a)$ realmente pertenece a $G(a)$. Para esto, es necesario determinar el valor de verdad del supuesto $I(X_a \perp X_b \mid X_c)$. Consultando al oráculo, el supuesto es verdadero porque $\text{sep}_{G^\star}(a, b; c)$ es verdadero. Como resultado, como muestra la Figura 2-7b, la arista (a, b) es removida del grafo G y la manta $G(a)$ pasa a tomar el valor $\{\, c \,\}$.

A continuación, se decide si el nodo $c \in G(a)$ pertenece a la manta. Al consultar el valor de verdad del supuesto $I(X_a \perp X_c \mid \emptyset)$, el oráculo responde que dicho supuesto es falso, porque $\text{sep}_{G^\star}(a, c; \emptyset)$ es falso, debido a la presencia del camino $a - c$. Al no existir un próximo nodo, la fase de encogimiento finaliza y retorna el grafo G mostrado en

la Figura 2-7c.

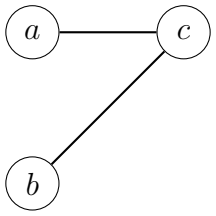


Figura 2-8: Grafo final obtenido por el algoritmo GS.

Finalizada las dos fases, el algoritmo GS toma el nodo b como próximo nodo objetivo. Repitiendo los pasos anteriores, la fase de crecimiento encontrará la arista (b, c) porque $\text{sep}_{G^\star}(b, c; \emptyset)$ es falso. Luego, durante la fase de encogimiento, la arista (b, c) no podrá ser removida porque $\text{sep}_{G^\star}(b, c; \emptyset)$ es falso, retornando el grafo G mostrado en la Figura 2-8.

Finalmente, el algoritmo GS tomará el nodo c como último nodo objetivo. En dicho nodo, ninguna de las fases realizará modificaciones adicionales sobre el grafo G. Así, el grafo de la Figura 2-8 es nuevamente retornado y el algoritmo GS finaliza. Tal como se puede apreciar, este grafo es un perfect map de $G^\star$.

□

2.5. Estimación de parámetros

Dada una estructura arbitraria pero fija (e.g., una obtenida por medio del aprendizaje de estructuras), la estimación de parámetros consiste en estimar el conjunto θ_S de parámetros utilizando el conjunto D, el cual asumimos está formado por un conjunto de estados en $val(V)$ totalmente observables, i.e., no hay estados perdidos (missing values).

Esta estimación puede ser realizada a través de la optimización de una función objetivo sobre el conjunto D de datos, donde dicha función objetivo mide, en base a D, qué tan bueno es un conjunto de parámetros con respecto a otro. Una función objetivo frecuentemente utilizada, y con sólidos fundamentos estadísticos, es el logaritmo de *la función de verosimilitud* (*log-likelihood*), la cual llamaremos como log-verosimilitud.

La log-verosimilitud se define formalmente como sigue. Sea $p(X)$ una distribución sobre X y sea D una muestra arbitraria tomada desde $p(X)$, la log-verosimilitud de θ dado D se define como:

$$ll(\theta\colon D) = \log \prod_{x \in D} p(x|\theta) = \sum_{x \in D} \log p(x|\theta).$$

Representando $p(X)$ con una red de Markov (G, θ), entonces la ecuación anterior puede ser expresada como:

$$ll(\theta\colon D) = \sum_{x \in D} \log \left\{ \frac{1}{Z} \prod_{i\colon \mathcal{C}(G)} \phi_i(x_C) \right\}. \tag{2.24}$$

Un conjunto de parámetros con un alta log-verosimilitud indica que el conjunto D tiene una alta probabilidad de haber sido generado por dicho conjunto de parámetros. Por lo tanto, podemos usar la log-verosimilitud como una criterio para seleccionar entre diferentes conjuntos de parámetros, seleccionando aquel conjunto que maximice la log-verosimilitud, donde dicho conjunto es llamado *estimador de máxima verosimilitud*.

Para encontrar el estimador de máxima verosimilitud, no se puede utilizar una expresión en forma cerrada de la Ecuación (2.24). En consecuencia, se debe recurrir a métodos iterativos de optimización. Un ejemplo representativo de tales métodos es el *gradiente ascendente* [93]. Tales métodos son una excelente opción para encontrar la máxima log-verosimilitud. Esto se debe a que la log-verosimilitud es una función convexa, lo cual *garantiza* la ausencia de máximos locales. En consecuencia, utilizando cualquier método iterativo, el máximo global de la log-verosimilitud, el estimador de máxima verosimilitud, puede ser encontrado en un número finito de pasos.

Sin embargo, cada paso de la optimización requiere ejecutar inferencia para evaluar la Ecuación (2.24) para un determinado conjunto de parámetros. Más concretamente, es necesario utilizar inferencia para computar la función de partición. Como veremos en la Sección 2.6, el cómputo de la función de partición involucra una sumatoria sobre

un número exponencial de términos, ocasionando que el proceso de optimización sea intratable, debido a que cada paso de la optimización iterativa requiere computar la función de partición. A pesar de eso, en la práctica podemos encontrar dos soluciones para enfrentar a este problema.

Una solución es utilizar métodos de inferencia aproximada. Desafortunadamente, el uso de inferencia aproximada durante la optimización, suele conducir a resultados imprecisos [48]. Otra solución consiste en utilizar otra función objetivo que sea tratable. Una de tales funciones es la *pseudo-likelihood* [7], el cual llamaremos pseudo-logverosimilitud. La pseudo-logverosimilitud está definido de manera de evitar la computación de la función de partición.

Formalmente, sea D un conjunto de $M = |D|$ observaciones, y (G, θ_G) una red de Markov arbitraria. La pseudo-logverosimilitud de (G, θ_G) se define como:

$$pll(\theta_G \colon D) = \frac{1}{M} \sum_{a \in V} \sum_{x \in D} \log p(x_a | x_{G(a)}, \theta_G). \tag{2.25}$$

La pseudo-logverosimilitud se diferencia de la log-verosimilitud en que la primera está definida desde distribuciones condicionales, no sobre una distribución conjunta. Esto último es lo que permite no computar la función de partición. Por ejemplo, dada cualquier distribución condicional $p(x_a|x_{G(a)})$ de la Ecuación (2.25), tenemos que la función Z no es necesaria computarla:

$$p(x_a|x_{G(a)}) = \frac{p(x_a, x_{G(a)})}{p(x_{G(a)})} = \frac{Z^{-1}\ \tilde{p}(x_a, x_{G(a)})}{Z^{-1}\ \tilde{p}(x_{G(a)})} = \frac{\tilde{p}(x_a, x_{G(a)})}{\tilde{p}(x_{G(a)})},$$

donde $\tilde{p}(\cdot)$ representa el producto de funciones paramétricas de una red de Markov sin estar normalizadas por Z. En otras palabras, se puede pensar que la distribución $\tilde{p}(x_{G(a)})$ es la función de partición para la distribución $p(x_a|x_{G(a)})$.

Una desventaja de utilizar la pseudo-verosimilitud para estimar el conjunto de parámetros es que, al utilizar posteriormente una red de Markov con dichos parámetros estimados, la red tiende a ser sólo precisa para consultas de inferencias que involucran un alto número de variables de evidencia [54].

Por otro lado, tanto la pseudo-logverosimilitud como la log-verosimilitud son pro-

pensas a *problemas de sobreajuste*, es decir, cuando los parámetros no sólo se ajustan a la distribución subyacente, sino también al ruido que puede estar presente en el conjunto D. Este ruido suele ser introducido durante el proceso de recolección del conjunto de datos.

Una forma de reducir los errores por sobreajuste es agregando, en la función objetivo, una función adicional sobre los parámetros θ_S, denominada "término de regularización". Básicamente, el término de regularización *suaviza* los parámetros, evitando que se ajusten estrictamente al conjunto de datos. El término de regularización puede ser caracterizado como una distribución a priori sobre θ_S, i.e. $p(\theta_S|\alpha)$, cuyos parámetros α son llamados *hiperparámetros*.

Por lo tanto, la pseudo-logverosimilitud regularizado se define como la suma de dos funciones:

$$pll(\theta_S : D) + \log p(\theta_S|\alpha).$$

En la práctica, dos términos de regularización ampliamente utilizados son las normas L_1 y L_2. L_1 asume una norma Laplaciana sobre θ_S y se define como:

$$p(\theta|\alpha) = \frac{1}{\alpha} \exp\left\{-\frac{|\theta|}{\alpha}\right\},$$

mientras L_2 asume una norma Gaussiana y se define como:

$$p(\theta|\alpha) = \prod_{i=1}^{k} \frac{1}{\sqrt{2\pi}\alpha} \exp\left\{-\frac{\theta_i^2}{2\alpha^2}\right\}.$$

Al estimar parámetros, el uso de L_1 tiende a remover parámetros (asignándoles un parámetro 0) a aquellos parámetros con un bajo soporte en los datos; mientras que, en el caso de L_2, los parámetros tienden a ser reducidos (pero no removidos). Por esta razón, muchos algoritmos de aprendizaje de estructuras utilizan esta característica de L_1 para seleccionar redes de Markov "ralas" (con pocos parámetros) [51, 38, 78, 88, 56].

La intensidad del término de regularización es controlada por los valores de los hiperparámetros α. Una selección inadecuada de hiperparámetros puede producir una

estimación incorrecta de parámetros. Por ejemplo, en el caso de L_1, hiperparámetros muy altos pueden provocar que muchos parámetros útiles sean removidos, resultando en una red de Markov imprecisa.

De esta manera, una manera para seleccionar los hiperparámetros consiste en utilizar técnicas de validación cruzada. La validación cruzada consiste en dividir el conjunto de datos en dos particiones. Una de las particiones es utilizada para aprender el conjunto de parámetros bajo diferentes configuraciones de hiperparámetros. La segunda partición es utilizada para seleccionar la mejor configuración de hiperparámetros. Como consecuencia, en algunos casos, la complejidad computacional de la estimación de parámetros tiende a aumentar considerablemente [15].

2.6. Inferencia

En términos generales, la tarea de inferencia consiste en utilizar una distribución $p(X)$ (o red de Markov) para responder consultas de interés. Dado un conjunto de variables como evidencia X_E, quizás la consulta más frecuente es determinar la distribución condicional $p(X_Q|X_E)$ desde la distribución $p(X)$[22]. En el caso general, se puede mostrar que la tarea de inferencia en redes de Markov es #P-complete [79].

Para ilustrar esto último, consideremos una distribución $p(X)$ donde $W = V \setminus (Q \cup E)$. Utilizando la regla de la cadena, $p(X)$ puede ser factorizada como

$$p(X) = p(X_Q|X_{E\cup W})p(X_{E\cup W}).$$

Ahora, para obtener la distribución condicional $p(X_Q|X_E)$, utilizamos la ley de la probabilidad total para remover la influencia de las variables X_W, i.e.,

$$p(X_Q|X_E) = \sum_{x_W \in val(W)} p(X_Q|X_E, x_W)p(X_E, x_W).$$

[22]En base a la distribución condicional $p(X_Q|X_E)$, una consulta frecuentemente realizada consiste en encontrar la asignación con máxima probabilidad dado una observación x_E, i.e., $\underset{x_Q \in val(Q)}{\arg\max}\ p(x_Q|X_E)$.

Si las distribuciones son representadas como tablas de parámetros, entonces obtener la distribución $p(X_Q|X_E)$ implica realizar un número exponencial de operaciones. En el ejemplo anterior, $p(X_Q|X_E)$ involucra $2^{|W|+2|E|+|Q|}$ operaciones: $2^{|W|}$ sumas y $2^{|Q|+|E|} \times 2^{|E|}$ multiplicaciones en cada suma.

Otro caso donde la intratabilidad de la inferencia resulta clara es al intentar computar la función de partición Z de una distribución de Boltzmann. Utilizando la Ecuación (2.17), tenemos que:

$$Z = \sum_{x \in val(X)} \prod_{i:\ C \in \mathcal{C}(G)} \phi_i(X_C),$$

donde $|val(X)| = 2^n$.

Uno de los usos de la estructura de una red de Markov es para reducir el costo computacional de la inferencia. En términos sencillos, la factorización de una distribución (a través de su estructura) reduce el número de parámetros, reduciendo el número de operaciones involucradas durante la inferencia [46, Parte II]. Por ejemplo, las *redes suma-producto* (sum-product networks), un caso especial de redes de Markov donde la estructura puede ser descompuesta en "subestructuras", el costo de ejecutar inferencia es lineal en el número de aristas [36].

Sin embargo, en el caso general, la inferencia en redes de Markov sigue siendo intratable (e.g., en el cómputo de la función de partición). Por esta razón, se suele recurrir a técnicas aproximadas para realizar inferencia, es decir, técnicas cuyos resultados no necesariamente son exactos. Una de las técnicas más utilizadas para inferencia aproximada es el *muestreo de Gibbs*, un caso especial de *Monte Carlo Cadena de Markov* [37].

El muestreo de Gibbs es un método que toma un conjunto D de muestras desde una distribución $p(X)$ (e.g., una red de Markov). Luego, usando el conjunto D de muestras, es posible responder cualquier pregunta de inferencia a través del cómputo de frecuencias (cuyo costo es lineal en $|D|$, es decir, el número de muestras, ver la Ecuación (2.21))..

A continuación introduciremos formalmente el muestreo de Gibbs para ilustrar

varios conceptos útiles. El Algoritmo 5 muestra una versión simple del muestreo de Gibbs basada en [46, Algoritmo 12.4]. Los parámetros de entrada para el muestreo de Gibbs son: un conjunto V de índices, un estado inicial $x \in val(V)$[23], una distribución $p(X)$, y un número entero positivo T. La salida es un conjunto D que contiene $|D| = T$ muestras.

Algoritmo 5 Muestreo de Gibbs

```
procedure GIBBS-SAMPLER(V, x, p(X), T)
    D ← ∅
    for t = 1, ..., T do
        D ← D ∪ { x }
        for each a ∈ V do
            x'_a ← tomar una muestra desde p(X_a | x_{V\{a}})
            x'_{-a} ← x_{V\{a}}
            x ← x'_{-a} ∧ x'_a
     return D
```

Básicamente, para generar el conjunto de muestras, el muestreo de Gibbs "perturba" cada variable del estado inicial x iterativamente, es decir, el estado de cada variable $X_a \in X$ es modificado sucesivamente. Una vez modificadas todas las variables, el estado perturbado x es almacenado en D. Usando el estado perturbado como nuevo estado inicial, se puede obtener otra nueva muestra repitiendo el proceso anterior.

El punto clave del Algoritmo 5 es cómo el estado de cada variable X_a es perturbado. Sea x un estado inicial, para perturbar una variable $X_a \in X$ en x, se toma una muestra, denotada por x'_a, desde la distribución condicional $p(X_a|x_{V\setminus\{a\}})$. Luego, el estado x'_a es utilizado para reemplazar el estado de X_a en x. Durante este proceso, intervienen dos elementos que fuertemente afectan la distribución condicional $p(X_a|x_{V\setminus\{a\}})$.

Primero, la distribución condicional está en función de un contexto: $x_{V\setminus\{a\}}$, de esta manera, la presencia de independencias específicas del contexto puede simplificar dicha distribución. Segundo, supongamos que G es la estructura de $p(X)$, entonces

[23]Como se mencionó en la sección de notación, un estado x consiste en una conjunción de estados marginales, i.e., $x \equiv \bigwedge_{a \in V} x_a$.

por el principio de localidad, sabemos que $p(X_a|x_{V\setminus\{a\}}) = p(X_a|x_{G(a)})$. Usando Ecuación (2.17), podemos expresar la distribución condicional como el siguiente producto de potenciales:

$$p(x_a|x_{V\setminus\{a\}}) = p(x_a|x_{G(a)}) \propto \prod_{i:\, C\in\mathcal{C}'(G)} \phi_i(x_C), \tag{2.26}$$

donde $\mathcal{C}'(G) = \{\, C \in \mathcal{C}(G) \colon C \subseteq G(a) \cup \{\, a \,\} \,\}$.

Lo importante de esta factorización es que la distribución condicional únicamente involucra a un subconjunto de los factores de la distribución $p(X)$. Este reducido número de potenciales implica un reducido número de parámetros, lo cual reduce el número de operaciones y simultáneamente incrementa la precisión del muestreo. De este modo, cualquier error del tipo II tiende a: (i) aumentar el alcance de los factores; y (ii) incrementar el número de potenciales. Este aumento en los alcances y en el número de factores inevitablemente incrementa el número de parámetros. Como resultado, la presencia de errores del tipo II en la estructura impactan sobre el rendimiento de la inferencia.

De este modo, la *eficiencia* así como la *eficacia* del muestreo de Gibbs se encuentran íntimamente relacionadas con la *topología de la estructura* de una red de Markov. En otras palabras, el rendimiento de la inferencia *está en función de la topología de la estructura*. Por ejemplo, como vimos anteriormente, si la estructura presenta errores del tipo II, la factorización produce un mayor número de parámetros (parámetros espurios). Estos parámetros adicionales impactan en el rendimiento de la inferencia: incrementando el número de operaciones y *decreciendo la precisión de las respuestas* [46, Teorema 12.6].

A la inversa, mientras más "exacta" es una estructura (i.e., la estructura tiende a ser un perfect map del conjunto de supuestos de independencias de la distribución subyacente), mejores niveles de eficiencia y eficacia pueden ser alcanzados. Por ejemplo, tal como se muestra en algunos experimentos [10, 76, 30], la codificación de independencias específicas del contexto en la estructura produce que la inferencia

obtenga mejores resultados en un menor tiempo.

Esta última observación es clave para la experimentación que presentaremos en el Capítulo 5, porque nos brinda **una forma de *medir* (o cuantificar) los beneficios de codificar independencias específicas del contexto en la estructura de una red de Markov.**

2.7. Resumen

En este capítulo introducimos varios conceptos elementales sobre redes de Markov. Mostramos que una red de Markov surge del hecho de factorizar una distribución de Boltzmann[24]. En consecuencia, una red de Markov ofrece una manera de representar eficientemente y eficazmente un gran espectro de distribuciones de probabilidad. Por otro lado, hicimos énfasis en los tres conceptos principales relacionados a una red de Markov, los cuales son: representación, aprendizaje e inferencia.

En la representación de redes de Markov, mostramos que una red de Markov puede ser representada por un modelo paramétrico, el cual se define por una estructura y un conjunto de parámetros. Definimos formalmente una estructura como un conjunto de independencias. Estudiamos los diferentes tipos de independencias y sus relaciones. Más específicamente, mostramos que el tipo más elemental de independencia (al menos para este trabajo) son las independencias instanciadas. A partir de las independencias instanciadas se pueden definir las independencias específicas del contexto así como también las independencias condicionales.

Presentamos dos representaciones de la estructura de una red de Markov: un grafo no dirigido y un conjunto de características. Discutimos las diferentes ventajas y desventajas que presentan cada una de estas representaciones. Puntualmente, mostramos que un grafo no dirigido, a diferencia de un conjunto de características, únicamente puede codificar independencias condicionales. Adicionalmente, mostramos que un grafo puede ser expresado como un conjunto de características y viceversa.

[24]Es importante remarcar que la factorización de una distribución Boltzmann incluye un término, llamado la función de partición, el cual involucra una sumatoria sobre un número exponencial de elementos.

En el aprendizaje de redes de Markov, mostramos que dicho tarea puede dividirse en dos subtareas: el aprendizaje de la estructura y la estimación de parámetros. En el aprendizaje de estructura, estudiamos principalmente los algoritmos basados en restricciones, los cuales representan la estructura de una red de Markov a través de un grafo no dirigido. Describimos en forma general cómo trabajan dichos algoritmos para construir un grafo desde un conjunto de datos. Por otro lado, ilustramos dichos algoritmos a través del estudio en particular de dos algoritmos basados en restricciones: el algoritmo PC y el algoritmo GS.

En la estimación de parámetros, mostramos que la optimizando de la verosimilitud es una de las formas para encontrar el conjunto de parámetros de una red de Markov. Sin embargo, la optimización de la verosimilitud tiene como subtarea el cómputo de la función de partición, lo cual vuelve intratable la optimización de la verosimilitud. En consecuencia, mostramos una manera alternativa de realizar la optimización, denominada pseudo-verosimilitud, la cual no requiere el cómputo de la función de partición.

Otro punto importante, con respecto a la estimación de parámetros, es el hecho de que la optimización de la (pseudo-)verosimilitud puede potencialmente producir errores de sobreajuste en los parámetros. Mostramos que una manera de enfrentar este problema es a través de la regularización de la (pseudo-)verosimilitud, la cual intenta "suavizar" los errores de sobreajuste. En la regularización, la suavización es controlada por la utilización de parámetros adicionales, los cuales son denominados hiperparámetros. Como desventaja, la selección de dichos hiperparámetros (a través de validación cruzada) produce un aumento inevitable del costo temporal de la estimación de parámetros.

Finalmente, mostramos que la inferencia es una manera de extraer información desde una red de Markov, por ejemplo, la computación de cualquier distribución condicional desde una red de Markov. En el caso general, la inferencia sobre una red de Markov es intratable. Este hecho hace surgir la necesidad de métodos aproximados para realizar inferencia. Estudiamos en detalle uno de los métodos más populares para realizar inferencia aproximada: el muestreo de Gibbs. Al estudiar el muestreo de

Gibbs, remarcamos la fuerte dependencia que existe entre la eficacia y eficiencia de dicho método y la exactitud de la estructura de una red de Markov. En otras palabras, mientras menos errores presente la estructura de una red de Markov, el muestreo de Gibbs producirá resultados más precisos.

Capítulo 3

Modelos canónicos

Un conjunto es un Muchos que permite en sí ser pensado como un Uno.

Georg Cantor.

Una distribución de probabilidad sobre un conjunto de variables está definida en base a una estructura. Dicha estructura consiste de un conjunto de dependencias e independencias entre las variables de la distribución. Al representar una distribución, su estructura es usualmente representada a través de alguna estructura de datos especializada. Por ejemplo, una de las representaciones más utilizada es un grafo no dirigido.

Al utilizar un grafo, las *independencias específicas del contexto*, un tipo particular de independencia entre variables, no pueden ser codificadas. Esto se debe a que un grafo únicamente puede codificar independencias condicionales. Por lo tanto, cuando una estructura presenta independencias específicas del contexto, su representación con un grafo produce *errores del tipo II* con respecto a la estructura subyacente.

Un error del tipo II implica asumir una dependencia como verdadera siendo que dicha dependencia es falsa en la estructura subyacente. O, en otros términos, un error del tipo II consiste en asumir una dependencia entre variables, siendo que dichas variables no son dependientes.

Como resultado, al utilizar un algoritmo de aprendizaje de estructuras para construir una estructura desde un conjunto de datos, si la estructura es representada por

un grafo, tal como sucede con los algoritmos basados en restricciones, entonces tal algoritmo *está sujeto a aprender estructuras que no codifican independencias específicas del contexto*.

Desde el punto de vista del aprendizaje de estructuras, una estructura errónea va en detrimento de los objetivos de estimación de densidad y descubrimiento de conocimiento. Analicemos esto último en más detalle.

En términos de la teoría de la información, utilizar un grafo para codificar una estructura que presenta independencias específicas del contexto produce una pérdida de información. En otras palabras, el grafo oculta independencias específicas del contexto, las cuales podrían aportar información relevante a un experto humano [35, 32]. Esta situación va en detrimento del objetivo de descubrimiento de conocimiento.

Por otro lado, debido a que una estructura es utilizada para representar eficientemente una distribución mediante su factorización[1], una factorización realizada con un grafo con errores del tipo II produce parámetros espurios. En situaciones de ausencia de datos, la presencia de parámetros espurios resulta en una distribución con *baja precisión predictiva* [76], lo cual va en detrimento del objetivo de estimación de densidad.

En la práctica existen otras representaciones de la estructura que permiten codificar independencias específicas del contexto [32, 70, 43, 44]. Para los fines de nuestro trabajo, la más relevante de estas representaciones son los *modelos CSI* [43]. La estructura de un modelo CSI presenta varias ventajas. Una de ellas es que puede ser representada a través de un conjunto de grafos llamados grafos instanciados.

Sin embargo, los modelos CSI son principalmente una propuesta teórica. En contraste, los *modelos divididos* son una propuesta práctica basada en los modelos CSI [44]. Hablando de forma aproximada, la estructura de un modelo dividido es llamada *grafo dividido* y está formada por un árbol cuyos nodos contienen grafos instanciados u otros grafos divididos. Una desventaja de un grafo dividido es que es una estructura sumamente compleja. Como resultado, no existe un algoritmo de aprendizaje de estructuras que pueda aprender un grafo dividido.

[1] Ver por ejemplo la Ecuación (2.1).

En este capítulo introduciremos un aporte teórico para atacar el problema recientemente discutido. Nuestra propuesta se basa en definir un nuevo modelo denominado *modelo canónico*, el cual se basa en los modelos CSI. La idea fundamental detrás de los modelos canónicos es que su estructura puede ser representada por un conjunto especial de grafos instanciados, los cuales hemos llamado *grafos canónicos*.

Representar la estructura utilizando un conjunto de grafos canónicos tiene aparejado dos interesantes características. En primer lugar, un conjunto de grafos canónicos puede codificar independencias específicas del contexto como también así independencias condicionales. En segundo lugar, proponemos un método para aprender un conjunto de grafos canónicos desde un conjunto datos utilizando *cualquier algoritmo basado en restricciones*.

Esta última característica destaca los modelos canónicos frente a los diferentes enfoques propuestos en la literatura. Por ejemplo, al revisar la literatura, los enfoques se caracterizan por utilizar una nueva representación de la estructura, la cual puede únicamente ser utilizada por *un nuevo método de aprendizaje de estructuras*. Por ejemplo, este es el caso de los modelos divididos.

Intuitivamente, esta necesidad de definir nuevos métodos de aprendizaje es una consecuencia del hecho que cualquier método de aprendizaje de estructuras está *íntimamente entrelazado* con la representación de la estructura. Esto es porque la representación de la estructura define el espacio de búsqueda, sobre el cual es diseñado un método de aprendizaje. En consecuencia, cualquier cambio en la representación de la estructura potencialmente afecta la forma en cómo una estructura debe ser aprendida.

Precisamente, en nuestra propuesta, la estructura de un modelo canónico es definida de tal manera que permite la reutilización de los algoritmos basados en restricciones como método de aprendizaje. Como resultado, uno de los aportes principales de los modelos canónicos es que su estructura puede ser aprendida por los algoritmos basados en restricciones.

Los contenidos de este capítulo se encuentran estructurados de la siguiente manera. En la Sección 3.1, presentaremos una visión general sobre los inconvenientes que surgen al representar y aprender estructuras que codifiquen independencias específi-

cas del contexto. Luego, en la Sección 3.2, presentaremos una revisión bibliográfica sobre los distintos enfoques para enfrentar dichos inconvenientes. En la Sección 3.3 mostraremos cómo una red de Markov puede ser representada utilizando los un modelo CSI. En la Sección 3.4, discutiremos a fondo cómo los modelos CSI representan la estructura de una red de Markov. Para ejemplificar los modelos CSI, la Sección 3.5 presenta los así llamados *modelos divididos*, un tipo particular de modelo CSI. En base a los modelos CSI y a los modelos divididos, en la Sección 3.6, presentaremos formalmente nuestra propuesta para representar una red de Markov: los *modelos canónicos*, cuya estructura puede ser representada a través de un conjunto de especial de grafos no dirigidos (los *grafos canónicos*). Finalmente, la Sección 3.7 presenta formalmente qué tipo de estructuras pueden ser codificadas por un conjunto de grafos canónicos. Dejaremos para el próximo capítulo los detalles de cómo aprender la estructura de un modelo canónico desde un conjunto de datos utilizando cualquier algoritmo basado en restricciones.

3.1. Inconvenientes de las independencias específicas del contexto

Como vimos formalmente en la Sección 2.1, la diferencia clave entre las independencias condicionales y las independencias específicas del contexto es que las últimas sólo se cumplen cuando un determinado conjunto de variables toma un estado particular, el cual es conocido como el contexto de la independencia. En otras palabras, una independencia condicional no tiene un contexto asociado. Ilustremos estos hechos con un ejemplo.

Ejemplo 3.1. Sea V el conjunto $\{a, b, c\}$ y sea $p(X)$ una distribución sobre un conjunto X de variables binarias indexadas por V, es decir, $X = \{ X_a, X_b, X_c \}$. Consideremos que la estructura S de $p(X)$ codifica el siguiente conjunto de independencias:

$$\mathcal{I}_S = \{ I(X_a \perp X_b \mid X_c = 1) \}.$$

Es decir, la estructura de $p(X)$ codifica una única independencia, a saber, la variable X_a es independiente de la variable X_b dado el contexto $X_c = 1$. Esta independencia es específica del contexto debido a que se encuentra asociada al contexto $X_c = 1$.

Decimos que la independencia $I(X_a \perp X_b \mid X_c = 1)$ se encuentra asociada al contexto $X_c = 1$ debido a la siguiente observación. Si el contexto fuese diferente a $X_c = 1$, entonces la independencia $I(X_a \perp X_b \mid X_c = 1)$ no se cumple en $p(X)$. Por ejemplo, la independencia $I(X_a \perp X_b \mid X_c = 0)$ no se cumple en $p(X)$, debido a que no pertenece a $\mathcal{I}_S$.

Por otro lado, en base a la Ecuación (2.8), podemos determinar que la independencia condicional $I(X_a \perp X_b \mid X_c)$ no se cumple en $p(X)$. Esto se debe a que el valor de verdad del supuesto $I(X_a \perp X_b \mid X_c)$ está en función de la conjunción de supuestos: $I(X_a \perp X_b \mid X_c = 0) \wedge I(X_a \perp X_b \mid X_c = 1)$, donde $I(X_a \perp X_b \mid X_c = 0)$ es falso; volviendo toda la conjunción falsa.

□

Es interesante notar que cuando una estructura presenta independencias específicas del contexto, la forma de factorizar una distribución de Boltzmann se encuentra condicionada según el contexto. En otras palabras, según el estado que tomen ciertas variables (e.g., $X_c = 1$ en el ejemplo anterior) puede hacer surgir ciertas independencias (e.g., $I(X_a \perp X_b \mid X_c = 1)$). Así, según el contexto, las independencias que surjan desde dicho contexto producirán una factorización particular en una distribución de Boltzmann. En contraste, cuando una estructura sólo presenta independencias condicionales, las cuales son *invariantes* al contexto[2], una distribución Boltzmann factoriza de igual manera independientemente del contexto.

El siguiente ejemplo ilustra como las independencias específicas del contexto producen diferentes factorizaciones de una distribución según el contexto.

Ejemplo 3.2. Como mencionamos en el Ejemplo 3.1, el conjunto $\mathcal{I}_S$ está formado por una única independencia, la cual sólo se cumple cuando $X_c = 1$. En consecuencia, al factorizar $p(X)$ sobre la estructura S, obtenemos diferentes factorizaciones en función del estado que tome la variable X_c.

[2]Esto se desprende de cómo es definida una independencia condicional, ver Ecuación (2.8).

Para mostrar esto, utilizaremos algunos conceptos básicos introducidos en la Sección 2.1. Primero, utilizando la regla de la cadena, la distribución $p(X)$ puede ser factorizada en el siguiente producto de distribuciones:

$$p(X) = p(X_c) \cdot p(X_a, X_b|X_c).$$

Dentro de este producto de distribuciones, encontramos la distribución $p(X_a, X_b|X_c)$, la cual se encuentra condicionada por la variable X_c. Según el Lema 2.1, sabemos que cualquier independencia específica del contexto es equivalente a una independencia condicional en una distribución condicionada por el contexto de la independencia.

Esto último implica lo siguiente. Por ejemplo, la independencia $I(X_a \perp X_b \mid X_c = 1)$ en $p(X)$ equivale a la independencia condicional $I(X_a \perp X_b \mid \emptyset)$ en la distribución $p(X_a, X_b|X_c = 1)$. Sin embargo, debido a que la independencia $I(X_a \perp X_b \mid X_c = 1)$ sólo está presente en el contexto $X_c = 1$, la distribución condicional $p(X_a, X_b|X_c = 0)$ *no presenta independencias.*

Por lo tanto, utilizando la Definición 2.2, la distribución $p(X_a, X_b|X_c)$ factoriza de distinta manera según el contexto. Cuando $X_c = 1$, la distribución $p(X_a, X_b|X_c = 1)$ factoriza como

$$p(X_a, X_b|X_c = 1) = p(X_a|X_c = 1) \times p(X_b|X_c = 1).$$

Sin embargo, cuando $X_c = 0$, la distribución $p(X_a, X_b|X_c = 0)$ no factoriza, es decir,

$$p(X_a, X_b|X_c = 0) \neq p(X_a|X_c = 0) \times p(X_b|X_c = 0).$$

□

Según [46, Capítulo 5], la estructura de una distribución $p(X)$ es usualmente definida como una combinación de dos tipos de estructuras: una *estructura global* y un conjunto de *estructuras locales*. La estructura global representa el conjunto de independencias condicionales presentes en $p(X)$, mientras que cada estructura local

representa un conjunto de independencias específicas *para un contexto fijo*.

Para ilustrar esta distinción en la estructura de una distribución, la distribución $p(X)$ del Ejemplo 3.1 presenta una estructura global que no codifica independencias condicionales y una única estructura local, la cual está asociada al contexto $X_c = 1$.

En base a lo anterior, cuando un grafo G es utilizado como representación de la estructura de una distribución, *el grafo solamente logra codificar la estructura global, no las estructuras locales* de una distribución. De esta manera, si la estructura de una distribución presenta estructuras locales, al representarla con un grafo, sus estructuras locales no son codificadas. Esto es ilustrado en el siguiente ejemplo.

Ejemplo 3.3. Continuando con la distribución $p(X)$ introducida en el Ejemplo 3.1. La estructura S de $p(X)$ puede ser vista como la combinación de dos estructuras: una *estructura global* y una *estructura local*. Dado que $p(X)$ no codifica independencias condicionales, la estructura global no codifica independencias. Por otro lado, la estructura local de $p(X)$ codifica una única independencia: $I(X_a \perp X_b \mid X_c = 1)$.

Al utilizar un grafo para representar la estructura de $p(X)$, este codifica su estructura global. El grafo que representa la estructura global de $p(X)$ es mostrado en la Figura 3-1.

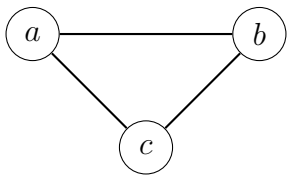


Figura 3-1: Representación de la estructura global de $p(X)$.

Para verificar que dicho grafo codifica la estructura global de $p(X)$, podemos usar separabilidad entre los nodos en G para certificar que ninguna independencias condicional es codificada por G. Debido a que no hay ningún nodo separado en G, entonces G no codifica ninguna independencia condicional.

□

Por lo tanto, al usar un grafo como representación de la estructura, el grafo presenta el sesgo de asumir que la distribución tiene una estructura global pero no estructuras locales. De este modo, para representar ambos tipos de estructuras de una

distribución, es necesario utilizar una representación que asuma ambos tipos. En otras palabras, es necesario una *representación que tenga la suficientemente flexibilidad para codificar simultáneamente las independencias condicionales y específicas del contexto* de una distribución. Una representación que presenta tal flexibilidad es un conjunto de características (ver la Sección 2.2.1).

Como ilustra el siguiente ejemplo, al utilizar un conjunto de características como representación de la estructura, podemos codificar simultáneamente la estructura global y las estructuras locales de una distribución.

Ejemplo 3.4. Siguiendo con la distribución $p(X)$ introducida en el Ejemplo 3.1. La estructura global y local de $p(X)$ puede ser representada utilizando el siguiente conjunto $\mathcal{F}$ de características:

1. $(X_a = 0 \wedge X_b = 0 \wedge X_c = 0)$;
2. $(X_a = 1 \wedge X_b = 0 \wedge X_c = 0)$;
3. $(X_a = 0 \wedge X_b = 1 \wedge X_c = 0)$;
4. $(X_a = 1 \wedge X_b = 1 \wedge X_c = 0)$;
5. $(X_a = 0 \wedge X_c = 1)$;
6. $(X_a = 1 \wedge X_c = 1)$;
7. $(X_b = 0 \wedge X_c = 1)$;
8. $(X_b = 1 \wedge X_c = 1)$.

Para verificar que el conjunto $\mathcal{F}$ codifica la estructura global y local de $p(X)$, utilizamos el procedimiento descripto en la Sección 2.2.1. Como comentamos, este procedimiento consiste en dos pasos. En el primer paso, un grafo es inducido desde el conjunto $\mathcal{F}$. En el segundo paso, utilizando el concepto de separabilidad entre nodos, el grafo inducido es examinado para verificar si presenta las independencias deseadas.

En base a lo anterior, usando todas las características en $\mathcal{F}$ (desde la número 1 a la número 8), el grafo inducido desde $\mathcal{F}$ luciría exactamente igual al grafo mostrado en la Figura 3-1. Tal como vimos en el Ejemplo 3.3, el grafo en la Figura 3-1 codifica la estructura global de $p(X)$, debido a que ningún nodo está separado.

Sin embargo, el conjunto $\mathcal{F}$ también codifica la estructura local de $p(X)$, la cual, según lo visto en el Ejemplo 3.1, sabemos que está asociada al contexto $X_c = 1$. Para ver esto, necesitamos solamente tener en cuenta el subconjunto de características en $\mathcal{F}$

que satisfacen el contexto $X_c = 1$. Así, las características número 5, 6, 7, y 8 satisfacen el contexto $X_c = 1$ (para más detalles, consultar la Ecuación 2.19). Denotaremos este subconjunto de características como $\mathcal{F}[X_c = 1] \subseteq \mathcal{F}$.

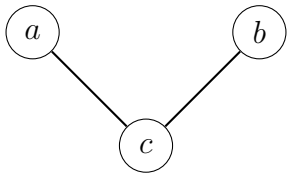


Figura 3-2: Grafo inducido desde $\mathcal{F}[X_c = 1] \subseteq \mathcal{F}$ que muestra la estructura local de $p(X)$.

Ahora, podemos usar el conjunto $\mathcal{F}[X_c = 1]$ para verificar independencias. El grafo inducido desde $\mathcal{F}[X_c = 1]$ es mostrado en la Figura 3-2. Dado que cada característica en $\mathcal{F}[X_c = 1]$ está asociada al contexto $X_c = 1$, decimos que el grafo inducido de la Figura 3-2 está "*instanciado*" (o contextualizado) por el contexto $X_c = 1$. En otras palabras, el nodo c del grafo está etiquetado/asociado a un *estado específico*, $X_c = 1$, en vez de una variable, X_c.

Como veremos en la Sección 3.3.2, un grafo como el mostrado en la Figura 3-2, donde algunos de sus nodos están asociados a estados en vez a variables, es formalmente conocido como *grafo instanciado*. Un grafo instanciado permite codificar independencias específicas del contexto. Para ver esto, en el grafo de la Figura 3-2, cualquier supuesto de independencia codificado por el grafo que involucre al nodo c, hace referencia al contexto $X_c = 1$ y no a la variable X_c. De este modo, usando separabilidad sobre el grafo de la Figura 3-2, tenemos que $\text{sep}_G(a, b; c)$, lo cual implica el supuesto $I(X_a \perp X_b \mid X_c = 1)$ en vez del supuesto $I(X_a \perp X_b \mid X_c)$. Como resultado, el conjunto $\mathcal{F}$ codifica tanto la estructura global como la estructura local de $p(X)$.

□

A pesar que un conjunto de características nos permite codificar simultáneamente la estructura global y las estructuras locales de una distribución, un conjunto de características presenta dos desventajas en comparación a un grafo no dirigido.

La primer desventaja es que un conjunto de características tiene baja interpretabilidad en comparación a un grafo. Por baja interpretabilidad nos referimos a que

recuperar cualquier (in)dependencia desde un conjunto características es más complejo que recuperarla desde un grafo. Para recuperar una (in)dependencia desde un conjunto de características, debemos revisar cada una de sus características, mientras que en un grafo sólo implica revisar cada uno de los nodos que intervienen en la (in)dependencia en cuestión.

Por ejemplo, el conjunto de características del Ejemplo 3.4 codifica las independencias presentes en la estructura global y local de la distribución $p(X)$ introducida en el Ejemplo 3.1. Sin embargo, dichas independencias se encuentran "ocultas" en una lista verbosa de características. Decimos "ocultas" debido a que, como se mostró en el Ejemplo 3.4, para determinar si $\mathcal{F}$ codifica las independencias de $p(X)$, es necesario primero inducir al menos un grafo desde $\mathcal{F}$. Para inducir un grafo desde $\mathcal{F}$, es necesario examinar cada una de sus características. Una vez inducido un grafo, el concepto de separabilidad entre nodos puede ser utilizado para "visualizar" las independencias codificadas por $\mathcal{F}$.

La segunda desventaja que presenta un conjunto de características es que, para visualizar una estructura local, debemos conocer a priori el contexto de dicha estructura. Más concretamente, en el Ejemplo 3.4, el grafo de la Figura 3-2, inducido desde el conjunto de características, representa la estructura local de $p(X)$. Sin embargo, para inducir dicho grafo, es necesario *seleccionar un subconjunto particular de características en* $\mathcal{F}$, es decir, un subconjunto no arbitrario de características en $\mathcal{F}$. En consecuencia, si dicho subconjunto de características es seleccionado incorrectamente, entonces el grafo inducido desde tal subconjunto podría no visualizar una estructura local. Para ilustrar esto último, supongamos que inducimos un grafo desde el subconjunto $\mathcal{F}[X_c = 0]$, entonces el grafo inducido desde dicho subconjunto luciría exactamente igual al grafo mostrado en la Figura 3-1. Este último grafo no muestra la estructura local asociada a $X_c = 1$.

De esta manera, si tenemos una estructura representada como un conjunto de características pero no sabemos a priori cuáles son los contextos de las estructuras locales, ¿cómo determinamos cuál es el subconjunto de características correcto para visualizar alguna de las estructuras locales de una distribución?

Una respuesta sencilla para esto sería inducir un grafo por cada posible subconjunto de características. Sin embargo, esta opción resultaría intratable debido a que hay $2^{|\mathcal{F}|}$ subconjuntos posibles en $\mathcal{F}$. Adicionalmente, este problema se ve exacerbado porque, en general[3], el número de características suele ser mayor al número de variables, i.e., $|\mathcal{F}| \gg n = |V|$.

Por lo tanto, si nuestro objetivo es descubrir conocimiento, al utilizar un conjuntos de características para representar la estructura tenemos un beneficio acotado. Por un lado, un conjunto de características nos permite codificar la estructura global y las estructuras locales de una distribución. Pero, por otro lado, no es una tarea trivial visualizar las estructuras locales codificadas en un conjunto de características.

En base a lo recientemente discutido, si una distribución de probabilidad presenta independencias específicas del contexto, una representación para tales estructuras debe presentar las siguientes propiedades:

1. Debido a que una estructura puede descomponerse en una estructura global (la cual codifica independencias condicionales) y un conjunto de estructuras locales (las cuales codifican independencias específicas del contexto), entonces la representación de una estructura debe permitir codificar ambos tipos de estructuras.

2. Debido a que una estructura local se encuentra asociada a un contexto (un estado tomado por un conjunto de variables), entonces la representación de una estructura debe permitir codificar explícitamente tales asociaciones.

En la siguiente sección, presentamos una revisión bibliográfica sobre los diferentes enfoques propuestos para representar estructuras con independencias específicas del contexto. Adicionalmente, mostraremos que dichos enfoques, al proponer una nueva forma de representar la estructura, se encuentran acompañados por nuevos métodos para aprender una estructura desde un conjunto de datos. Como hemos comentado, una desventaja de proponer nuevos métodos de aprendizaje es que no permite la reutilización de otros algoritmos de aprendizaje de estructuras ya diseñados, tal como los algoritmos basados en restricciones.

[3]En nuestros experimentos, en el mejor caso, $|\mathcal{F}|$ es en promedio un orden de magnitud mayor a n.

3.2. Trabajos relacionados

Desde una perspectiva amplia, los distintos enfoques para representar y aprender independencias específicas del contexto pueden ser agrupados en enfoques para redes de Bayes y enfoques para redes de Markov. Las redes de Bayes se encuentran fuera del alcance de este trabajo. Sin embargo, los enfoques para redes de Bayes son especialmente útiles porque sirven de base para entender mejor los enfoques de redes de Markov. Por ejemplo, muchas de las ideas utilizadas por los enfoques de redes de Markov se basan en ideas utilizadas por los enfoques en redes de Bayes. Por lo tanto, esta sección se estructura en dos partes: la primera presenta los enfoques para redes de Bayes y la segunda los enfoques para redes de Markov.

3.2.1. Enfoques para redes de Bayes

Los enfoques para redes de Bayes están principalmente basados en el hecho de que una red de Bayes factoriza una distribución de Boltzmann como un producto de *distribuciones condicionales* [10, 16, 33]. Más concretamente, sea $p(X)$ una distribución positiva, $p(X)$ factoriza sobre una red de Bayes si

$$p(X) = \prod_{a \in V} \prod_{S \models C_a} p(X_a | X_{C_a}), \tag{3.1}$$

donde $C_a \subseteq V$ y $S \models C_a$ denota que, para una estructura S dada, existe un conjunto X_{C_a} de variables tal que una variable X_a es condicionalmente independiente de las restantes variables $X_{V \setminus C_a}$ dado X_{C_a}. En otras palabras, la estructura codifica independencias condicionales tal que, para todo $a \in V$, existe un conjunto C_a tal que $p(X_a | X_{V \setminus \{a\}}) = p(X_a | X_{C_a})$. En una distribución $p(X_a | X_{C_a})$ decimos que *la variable X_a está en función de los estados de las variables X_{C_a}*.

A diferencia de una red de Markov, los factores en la Ecuación (3.1) son *distribuciones condicionales univariadas*. Estas distribuciones condicionales univariadas presentan dos interesantes características.

Primero, las variables de su alcance se encuentran agrupadas en dos conjuntos disjuntos. Por ejemplo, en una distribución arbitraria, e.g. $p(X_a|X_{C_a})$, tenemos: la *variable de respuesta* X_a y las *variables explicativas* X_{C_a}.

Segundo, cualquier distribución condicional univariada se encuentra normalizada (i.e., sus parámetros suman 1). Como resultado, en la factorización mostrada en la Ecuación (3.1), la función de partición *desaparece*. (Estrictamente hablando, en una red de Bayes, la función de partición es igual a 1.)

Una consecuencia clave de la segunda característica es que las distribuciones condicionales univariadas mostradas en la Ecuación (3.1) son *independientes entre sí*. Esto es porque cualquier cambio en los parámetros de una distribución condicional univariada en la Ecuación (3.1) no tiene impacto sobre las restantes distribuciones condicionales univariadas, debido a que sus parámetros no se encuentran acoplados por la función de partición [46, Capítulo 3]. En otras palabras, los parámetros de una distribución condicional univariada solamente afectan a los restantes parámetros involucrados en dicha distribución, debido a que la suma de los parámetros de una distribución condicional univariada deben sumar 1.

Por estas razones, los factores de una red de Bayes pueden verse como un conjunto de modelos independientes, donde cada modelo, uno por cada variable $X_a \in X$, representa dependencias asociadas a X_a [10]. Como consecuencia de esto, la estructura de una red de Bayes puede ser fácilmente representada utilizando dos estructuras de datos distintas [10]:

- un grafo (dirigido acíclico) que representa la estructura global, donde la estructura global determina un conjunto de $n = |X|$ distribuciones condicionales univariadas independientes entre sí; y

- una representación especial para la estructura local presente en cada distribución condicional univariada.

En la práctica, podemos encontrar dos estructuras de datos para representar una estructura local: los *árboles de decisión* [33] y los *grafos de decisión* [16]. Los grafos

de decisión son una generalización de los árboles de decisión [16]. De esta manera, nos focalizaremos en los árboles de decisión.

En términos sencillos, un árbol de decisión es una estructura de datos para representar eficientemente los parámetros asociados a una distribución condicional univariada. Trivialmente, una distribución condicional univariada puede ser representada como una tabla de parámetros, sin embargo, como veremos próximamente, esta representación puede producir parámetros espurios.

Más formalmente, un árbol de decisión es un grafo dirigido con exactamente un nodo raíz. Todos los restantes nodos en el árbol tienen exactamente un nodo padre (excepto el nodo raíz). Un nodo puede tener uno o más nodos descendientes. Un nodo es denominado "nodo hoja", si no tiene descendientes. Desde cualquier nodo hoja, existe un único camino que conecta dicho nodo con el nodo raíz.

Dada una distribución $p(X_a|X_{C_a})$, la variable X_a es asociada la raíz de un árbol de decisión, mientras que las variables X_{X_a} son asociadas a todos los restantes nodos no hojas. En otras palabras, cada nodo no hoja está etiquetado con una de las variables explicativas (i.e., cualquier variable en X_{C_a}). Dado un nodo no hoja asociado a una variable explicativa, e.g. $X_b \in X_{C_a}$, cada una de las aristas que conectan el nodo no hoja con sus descendientes están etiquetadas con cada uno de los estados en $val(b)$. En base a esto, cualquier camino desde la raíz a un nodo hoja está asociado a un contexto específico de las variables explicativas. En consecuencia, cada nodo hoja representa la distribución univariada de X_a condicionada por un contexto específico. Así, cada nodo hoja está asociado a una tabla de parámetros.

El siguiente ejemplo ilustrar cómo utilizar un árbol de decisión para representar la estructura local de una distribución condicional univariada.

Ejemplo 3.5. Usando la distribución $p(X)$ del Ejemplo 3.1, asumamos que la estructura global de una red de Bayes descompone $p(X)$ de la siguiente manera:

$$p(X) = p(X_a|X_b, X_c) \cdot p(X_b|X_c) \cdot p(X_c).$$

Dado que cada distribución condicional univariada es independiente de las restantes,

tomemos una arbitraria, e.g. $p(X_a|X_b, X_c)$, y representemos su estructura local con un árbol de decisión.

Antes de representar dicha distribución, es importante notar que $p(X)$ satisface la independencia $I(X_a \perp X_b \mid X_c = 1)$. Por lo tanto, por la Definición 2.2, la distribución $p(X_a|X_b, X_c)$ debe satisfacer la siguiente igualdad:

$$p(X_a|X_b, X_c = 1) = p(X_a|X_c = 1).$$

Como veremos a continuación, esta independencia específica del contexto puede ser codificada por un árbol de decisión.

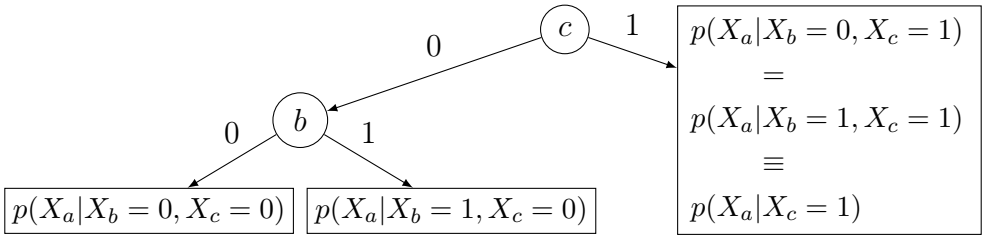


Figura 3-3: Un árbol de decisión representando la estructura local de la distribución $p(X_a|X_b, X_c)$.

La Figura 3-3 muestra un árbol de decisión que utiliza la estructura local presente en la distribución $p(X_a|X_b, X_c)$. Este árbol tiene X_c como la raíz del árbol. Por otro lado, tenemos las variables X_c y X_b como nodos no hojas. Finalmente, el árbol presenta tres nodos hojas, cada uno asociado a diferentes distribuciones condicionales univariadas sobre la variable de respuesta X_a.

Al analizar el árbol de la Figura 3-3, tres caminos (o contextos) desde la raíz a cada hoja pueden ser identificados. De izquierda a derecha, estos caminos/contextos son:

- $X_b = 0 \wedge X_c = 0$;
- $X_b = 1 \wedge X_c = 0$; y
- $X_c = 1$.

Cada uno de estos contextos condiciona las diferentes distribuciones asociadas a cada nodo hoja. Por ejemplo, cuando el contexto es $X_c = 1$, X_a es independiente de X_b, entonces la distribución en el nodo hoja es $p(X_a|X_c = 1)$, mientras que, cuando $X_c = 0$, la variable X_a es dependiente de X_b.

□

Debido a que la estructura de una red de Bayes puede representarse por dos estructuras de datos distintas, el aprendizaje de la estructura de una red de Bayes puede dividirse en dos subproblemas. El primero es el aprendizaje de la estructura global. Para aprender la estructura global, puede ser utilizado cualquier algoritmo básico de aprendizaje de grafos dirigidos.

Una vez obtenida la estructura global, el segundo problema es el aprendizaje de las estructuras locales. Este último problema puede ser dividido en n subproblemas: el aprendizaje de la estructura local de cada distribución condicional univariada. Cada una de estas estructuras locales puede ser aprendida utilizando algún método especializado para inducir un árbol/grafo de decisión desde datos [16, 33, 5]. Para garantizar que un árbol/grafo de decisión codifique la estructura local de una distribución $p(X_a|X_{C_a})$, el árbol de decisión tiene que asociar las variables explicativas X_{C_a} a los nodos no hojas del árbol, y asociar distribuciones univariadas sobre la variable de respuesta X_a a los nodos hojas [10].

3.2.2. Enfoques para redes de Markov

En general, los enfoques para representar estructuras locales en redes de Bayes no pueden ser directamente adaptados para redes de Markov. Esto se debe principalmente a dos razones.

Primero, a diferencia de una red de Bayes, los factores de una red de Markov no son necesariamente distribuciones. Por ejemplo, en una red de Markov, los parámetros de un factor no necesariamente están limitados a sumar 1. Por lo tanto, una red de Markov define la función de partición, la cual normaliza el producto de todos los factores. En consecuencia, los factores de una red de Markov se encuentran acoplados

entre sí, es decir, no son independientes entre sí. Por ejemplo, si tomamos un factor arbitrario y modificamos el valor de uno de sus parámetros, entonces el valor de la función de partición es modificado, lo cual termina perturbando los parámetros de todos los restantes factores. Como resultado, a diferencia de lo que sucede en una red de Bayes, las estructuras locales en una red de Markov no necesariamente se encuentran asociadas a cada uno de sus factores.

Segundo, los factores en una red de Markov son funciones arbitrarias. Por lo tanto, un factor no necesariamente es una distribución condicional univariada. En consecuencia, las variables en el alcance de cualquier factor no se encuentra necesariamente divididas. Por ejemplo, en una distribución condicional univariada, las variables de su alcance pueden ser divididas en dos grupos: variables de respuesta y variables explicativas. Como resultado, la identificación de contextos en una red de Markov es un problema complejo.

Por lo tanto, en la literatura podemos encontrar dos tipos de enfoques en redes de Markov. En el primer tipo de enfoques, se asume supuestos adicionales sobre la forma funcional de una red de Markov. Por ejemplo, al asumir tales supuestos, la estructura de una red de Markov presenta una topología especial [28, 29, 68, 69, 70]. En el otro tipo de enfoques, no se asumen supuestos adicionales sobre la forma funcional de una red de Markov [43, 44, 40]. Analicemos más en detalle cada uno de estos enfoques.

Por lo general, en el primer tipo de enfoques, se asume que la distribución subyacente es un *modelo descomponible* (decomposable model). Teóricamente hablando, un modelo descomponible puede ser representado tanto por una red de Markov como por una red de Bayes [72, § 3.3.3]. Como consecuencia, la estructura de un modelo descomponible puede ser representada por un *grafo cordal* (*chordal graph*). Asumiendo un grafo cordal, una distribución puede ser descompuesta en un conjunto de factores independientes entre sí. Por ejemplo, en los trabajos [68, 69], representan un modelo llamado *stratified graphical model*, el cual es, en términos simples, una red de Markov cuya estructura es un grafo cordal. Por lo tanto, las ideas comentadas en la Sección 3.2.1 pueden ser adaptadas de forma sencilla para modelos descomponibles.

En contraste, el segundo tipo de enfoques no asumen un modelo descomponible,

ocasionando que el problema de representar y aprender las estructuras locales de una red de Markov sea mucho más desafiante. En general, estos enfoques se caracterizan por proponer nuevas representaciones de una red de Markov, las cuales permiten codificar tanto la estructura global como las estructuras locales de una distribución. Adicionalmente, junto a estas nuevas representaciones, estos enfoques también proponen nuevos métodos de aprendizaje de estructuras, los cuales están especialmente diseñados para explotar las representaciones de la estructura propuestas.

Dentro de estos últimos enfoques, podemos resaltar dos propuestas para representar una red de Markov: los *modelos de Markov mixtos* (*mixed Markov models*) [32] y los *modelos CSI*[4] [44]. Dedicaremos el resto de esta sección para revisar los modelos de Markov mixtos. Sin embargo, debido a que nuestra propuesta está basada en los modelos CSI, dedicaremos la próxima sección para revisar en profundidad dichos modelos.

Retomando nuestra discusión, los modelos de Markov mixtos son un tipo especial de red de Markov donde sus variables están agrupadas en dos clases: *variables regulares* y *variables punteros* (*address variables*). Las primeras son las típicas variables presentes en una red de Markov, mientras que las segundas son variables cuyos estados hacen referencia a las variables regulares. Por ejemplo, una variable puntero puede verse como un puntero hacia cualquier otra variable regular. En consecuencia, cuando una variable puntero toma un estado en particular, dicha variable se comporta como la variable regular a la cual apunta.

Según se muestra en [32], los modelos de Markov mixtos pueden codificar estructuras locales a través de un grafo especial. En términos sencillos, los nodos de este grafo especial son de dos tipos. Un tipo de nodo es aquel asociado a una variable regular, mientras que el otro tipo de nodo es aquel asociado a una variable puntero. Como consecuencia, según el tipo de nodo, una arista en este grafo tienen un significado particular. Por ejemplo, aristas entre nodos regulares son interpretadas de igual manera que las aristas en un grafo no dirigido. Sin embargo, aristas entre nodos de diferentes tipos (puntero-regular) tienen asociado interpretaciones mucho

[4]Las siglas CSI provienen del nombre en inglés: Context Specific Interaction models.

más compleja. Esto se debe a que, según el estado tomado por un nodo puntero, éste se comporta de distintas formas. Como resultado, la topología en un grafo de un modelo de Markov mixto permite codificar complejas independencias, entre ellas las independencias específicas del contexto.

Una desventaja de los modelos de Markov mixtos es que el aprendizaje de su estructura es un problema complejo. Esto se debe principalmente a la distinción que existe entre las variables (puntero-regular). Por ejemplo, en el trabajo [32], la distinción entre variables punteros y regulares se asume como un conocimiento a priori. En consecuencia, no existe (en el mejor de nuestros conocimientos) un criterio para definir una variable como puntero o regular.

Para ilustrar esto último, en el trabajo [6] utilizan un modelo de Markov mixto para modelar los cambios que ocurren entre pares de imágenes aéreas tomadas en una misma coordenada espacial pero en diferentes períodos de tiempo. En términos sencillos, dada una región espacial en particular, este modelo se utiliza para sus píxeles como: constante/fondo o cambiado. En base a esto, se pueden detectar automáticamente cambios en dicha región a través del tiempo.

La estructura de este modelo de Markov mixto consiste en tres grafos diferentes. Cada uno de estos grafos tiene aristas que conectan sus propios nodos. Pero, los nodos de diferentes grafos pueden estar conectados a través de variables punteros. Esta estructura, sumamente compleja, permite resolver con exactitud el problema de clasificación de píxeles. Sin embargo, dicha estructura fue definida utilizando conocimiento de expertos sobre el dominio del problema. En otras palabras, la estructura no fue aprendida automáticamente por un algoritmo de aprendizaje de estructuras.

3.3. Modelos CSI

Recordando lo discutido en la Sección 3.2.2, los modelos CSI se diferencian de otras representaciones alternativas en el hecho que no utilizan supuestos adicionales sobre la forma funcional de una red de Markov. Por ejemplo, un modelo CSI no asume que la distribución subyacente es descomponible.

Un modelo CSI es una representación alternativa de una red de Markov [44]. En términos generales, un modelo CSI puede pensarse como una generalización de los modelos log-linear y los modelos probabilísticos gráficos. Por un lado, la estructura de un modelo CSI tiene la flexibilidad de un conjunto de características. Por otro lado, la estructura de un modelo CSI también tiene la interpretabilidad de un grafo.

Como veremos en esta sección, la estructura de un modelo CSI permite superar las dificultades que discutimos en la Sección 3.1. Por ejemplo, la estructura de un modelo CSI permite codificar tanto la estructura global como también las estructuras locales de una distribución. Por lo tanto, la estructura de un modelo CSI puede codificar independencias específicas del contexto.

Esta sección presenta formalmente los modelos CSI. Para esto, la sección se encuentra estructurada de la siguiente manera. La Sección 3.3.1 introduce formalmente los diferentes conceptos involucrados en un modelo CSI, entre ellos, la estructura de un modelo CSI. La Sección 3.3.2 discute cómo representar gráficamente la estructura de un modelo CSI. Finalmente, la Sección 3.3.3 detalla algunos de los inconvenientes que surgen al representar gráficamente la estructura de un modelo CSI.

Como veremos más adelante, los modelos CSI son la base sobre la que se definen los modelos divididos, los cuales son presentados en la Sección 3.5, y nuestra propuesta, los modelos canónicos, presentados en la Sección 3.6.

3.3.1. Definiciones y notaciones

En esta sección primero definiremos la estructura de un modelos CSI y luego definiremos un modelo CSI.

La estructura de un modelo CSI está definida, por así decirlo, por un conjunto de elementos muy simples, los cuales son formalmente llamados *generadores*. En términos generales, un *generador* puede interpretarse como un clique, como una característica, o como una combinación de estos dos.

Como vimos en el Capítulo 2, un clique es un subconjunto de nodos en un grafo, donde cada nodo en un clique se encuentra totalmente conectado con los restantes. En una red de Markov (ver la Ecuación (2.17)), cada uno de los cliques en un grafo

se corresponden con las variables en los alcances de los factores. Como resultado, un clique (así como un generador) puede pensarse como un conjunto de variables que interaccionan mutuamente entre sí.

Sin embargo, un clique, a diferencia de una característica, no se encuentra asociado a ningún contexto en particular. En contraste, una característica es un clique asociado a un contexto. Pero el contexto de una característica no es arbitrario, únicamente involucra un estado para cada una de las variables de la característica. Por lo tanto, una característica (así como un generador) es un conjunto de variables que interaccionan mutuamente entre sí cuando cada una de éstas tiene asignado un estado particular.

Por otro lado, un generador puede ser interpretado como una combinación de un clique y una característica. Por ejemplo, un generador representaría un conjunto de variables donde algunas de ellas tienen (o no) un estado asignado. En este caso, un generador se diferencia de un clique en que las variables se encuentran asociadas a un contexto. Por otro lado, el generador se diferencia de una característica en el hecho de que no todas las variables tienen un estado asignado. Por lo tanto, un generador puede representar interacciones que no pueden ser representadas por un clique o por una característica.

Formalmente, un generadora se define de la siguiente manera.

Definición 3.1. (c.f. [44]). Sea X un conjunto de variables aleatorias, y sean A y B dos conjuntos disjuntos en V. Sea x_B^k un estado particular en $val(B)$. Un *generador* sobre X es un par $[x_B^k; A]$.

Dado un generador $[x_B^k; A]$ arbitrario sobre X, si $B = \emptyset$ entonces el generador es denotado simplemente como $[A]$. Por ejemplo, un generador $[A]$ se corresponde con un clique (un conjunto de nodos). Similarmente, dado un generador $[x_B^k; A]$, si $A = \emptyset$, entonces el generador es denotado simplemente como $[x_B^k]$. Por ejemplo, un generador $[x_B^k]$ se corresponde con una característica (conjunción de estados utilizada por una función indicador).

Un generador tiene la propiedad de satisfacer o no un determinado estado o contex-

to. Más formalmente, sea $[x_B^k; A]$ un generador sobre X. Sea x_W un estado arbitrario en $val(W)$. El generador $[x_B^k; A]$ *satisface* el estado x_W, si

$$x_{B \cap W} = x_{B \cap W}^k.$$

Aunque un generador únicamente puede representar una interacción en particular, un conjunto de generadores permiten representar diferentes interacciones simultáneamente. Formalmente, un conjunto de generadores es llamado *clase generadora*. Usando una clase generadora, es posible codificar la estructura global como las estructuras locales de una distribución de probabilidad.

Definición 3.2. (c.f. [44]). Sea X un conjunto de variables aleatorias. Una *clase generadora* $\mathcal{C}$ sobre X es un conjunto finito de generadores sobre X, i.e.,

$$\mathcal{C} = \{\ [x_B; A], [x_W; B], \dots\ \}.$$

Hay una importante operación sobre una clase generadora, la cual usaremos reiteradas veces. Dada una clase generadora $\mathcal{C}$ sobre X, el subconjunto de generadores en $\mathcal{C}$ que *satisfacen un estado* $x_W \in val(W)$, con $W \subseteq V$, lo denotaremos como $\mathcal{C}(x_W)$.

Ejemplo 3.6. En este ejemplo mostraremos como utilizar una clase generadora $\mathcal{C}$ para representar la estructura de la distribución $p(X)$ introducida en el Ejemplo 3.1. Como mostramos en el Ejemplo 3.4, la estructura de $p(X)$ puede representarse como un conjunto de características $\mathcal{F}$, donde el conjunto $\mathcal{F}$ representa la estructura global de $p(X)$ y el subconjunto $\mathcal{F}[X_c = 1] \subseteq \mathcal{F}$ representa la estructura local de $p(X)$. La estructura global no codifica independencias condicionales, mientras que la estructura local, asociada al contexto $X_c = 1$, codifica el supuesto $I(X_a \perp X_b \mid X_c = 1)$. La siguiente clase generadora representa la estructura global y local de $p(X)$:

$$\mathcal{C} = \{[X_c = 0; \{\ a, b\ \}], [X_c = 1; \{\ a\ \}], [X_c = 1; \{\ b\ \}]\}.$$

Intuitivamente, siguiendo la interpretación de que un generador es como un clique, la clase generadora $\mathcal{C}$ puede verse como un conjunto de tres cliques: $\{\ a, b, c\ \}$, $\{\ a, c\ \}$,

$\{ b, c \}$; uno por cada generador en $\mathcal{C}$. Estos cliques se corresponden con los cliques del grafo inducido desde $\mathcal{F}$, mostrado en la Figura 3-1, el cual representa la estructura global de $p(X)$. Por otro lado, la clase generadora $\mathcal{C}$ puede dividirse en los siguientes dos subconjuntos:

$$\mathcal{C}(X_c = 0) = \{[X_c = 0; \{ a, b \}]\},$$

y

$$\mathcal{C}(X_c = 1) = \{[X_c = 1; \{ a \}], [X_c = 1; \{ b \}]\}.$$

La clase $\mathcal{C}(X_c = 1)$ está compuesta por dos generadores, ambos asociados al contexto $X_c = 1$, los cuales puede pensarse como dos cliques: $\{ a, c \}$ y $\{ b, c \}$. Justamente estos son los cliques del grafo inducido desde $\mathcal{F}[X_c = 1]$, mostrado en la Figura 3-2, el cual codifica la estructura local de $p(X)$.

□

Utilizando una clase generadora, la siguiente definición muestra cómo factorizar una distribución $p(X)$.

Definición 3.3. (c.f. [44, § 2]). Sea $p(X)$ una distribución positiva sobre X, y $\mathcal{C}$ una clase generadora sobre X. Decimos que $p(X)$ *factoriza* sobre $\mathcal{C}$ si

$$p(x) = \frac{1}{Z} \prod_{j\colon [x_B^k; A] \in \mathcal{C}} \phi_j(x_{A \cup B})^{f_B^k(x)}. \tag{3.2}$$

Una distribución $p(X)$ factoriza sobre una clase generadora $\mathcal{C}$ si para todo generador $[x_B^k; A] \in \mathcal{C}$ existe un factor $\phi(\cdot)$ cuyo alcance es $A \cup B$. El factor asociado a un generador $[x_B^k; A]$ se encuentra elevado por una función indicador, f_B^k, la cual está defina sobre el estado x_B^k del generador. Dado que el rango de una función indicador es $\{ 0, 1 \}$, el rol de la función indicador es "remover la influencia" de aquellos parámetros en el factor que no satisfacen el contexto x_B^k. En otras palabras, los parámetros que no satisfacen un contexto son una constante igual a uno, no afectando el producto

del resto de los parámetros.

En base a la factorización mostrada en la Ecuación (3.3), un *modelo CSI* consiste simplemente en el conjunto de distribuciones que factorizan sobre una determinada clase generadora.

Definición 3.4. (c.f. [44]). Sea $\mathcal{C}$ una clase generadora sobre X. Un *modelo CSI* con respecto a $\mathcal{C}$ es la familia de distribuciones que factoriza sobre $\mathcal{C}$.

Por lo tanto, una clase generadora es la estructura de un modelo CSI. Para mostrar las ventajas de la estructura de un modelo CSI, ilustraremos en el Ejemplo 3.7 cómo una clase generadora puede representar tanto un conjunto de cliques (lo cual define un grafo) como también un conjunto de características en el Ejemplo 3.8.

Ejemplo 3.7. Sea $p(X)$ una distribución positiva, y $\mathcal{C}$ una clase generadora sobre X, donde todo generador en $\mathcal{C}$ tiene la forma $[A]$. Una distribución $p(X)$ factoriza sobre $\mathcal{C}$ si

$$p(x) = \frac{1}{Z} \prod_{j\colon [A]\in\mathcal{C}} \phi_j(x_A), \tag{3.3}$$

donde todas las funciones indicador devuelven trivialmente 1.

Comparando la factorización de la Ecuación (3.3) con la factorización de la Ecuación (2.17) (donde una distribución es factorizada en base a un grafo no dirigido), cada generador en $\mathcal{C}$ equivale a un clique máximo en un grafo no dirigido.

□

Ejemplo 3.8. Sea $p(X)$ una distribución positiva, y $\mathcal{C}$ una clase generadora sobre X, donde todo generador en $\mathcal{C}$ tiene la forma $[x_B^k]$. Una distribución $p(X)$ factoriza sobre $\mathcal{C}$ si

$$p(x) = \frac{1}{Z} \prod_{j\colon [x_B^k]\in\mathcal{C}} \phi_j(x)^{f_B^k(x)}.$$

Ahora, como $p(X)$ es positiva, podemos reexpresar esta factorización aplicando la identidad $p(X) = \exp(\log(p(X)))$ como sigue:

$$p(x) = \frac{1}{Z} \exp \left\{ \sum_{j \colon [x_B^k] \in \mathcal{C}} f_B^k(x) w_j \right\},$$

donde $\log \phi_j(x) = w_j$.

Comparando la factorización de la arriba con la factorización de la Ecuación (2.18) (donde una distribución es factorizada en base a un conjunto de características), cada generador en $\mathcal{C}$ equivale a una característica.

□

En base a los ejemplos anteriores, ciertas clases generadoras puede ser representadas como un grafo no dirigido o como un conjunto de características. Sin embargo, en el caso general, ¿cómo se representa una clase generadora? Como veremos en lo que resta de este capítulo, una clase generadora puede representarse de varias formas distintas. Sin embargo, todas estas formas se basan en el hecho de que una clase generadora puede ser asociada a un conjunto especial de grafos. En la próxima sección, introduciremos este tipo especial de grafo, llamado grafo instanciado, y mostraremos que un grafo instanciado puede representar gráficamente a un conjunto de generadores.

3.3.2. Interpretación gráfica de una clase generadora

Una clase generadora puede ser representada por un grafo no dirigido. Esto es formalizado en la siguiente definición.

Definición 3.5. Sea X un conjunto de variables. Sea $\mathcal{C}$ una clase generadora arbitraria sobre X. Un grafo $G = (V, E)$ es una representación gráfica de $\mathcal{C}$, si el conjunto E de aristas en G es

$$E = \{ \, (a, b) \colon [x_B; A] \in \mathcal{C} \wedge a, b \in A \cup B \, \}.$$

El siguiente ejemplo ilustra cómo representar una clase generadora con un grafo.

Ejemplo 3.9. Usando la Definición 3.5, la clase generadora $\mathcal{C}$ introducida en el Ejemplo 3.6 tendría asociado un grafo G exactamente igual al mostrado en la Figura 3-1. □

Sin embargo, usando un grafo para representar una clase generadora ocultamos posibles estructuras locales. Por ejemplo, como hemos visto, un grafo no puede codificar independencias específicas del contexto. Para evitar este problema, podemos utilizar la siguiente observación. Dada una distribución $p(X)$ positiva y un estado $x_W \in val(W)$ arbitrario con $W \subseteq V$, si $p(X)$ factoriza sobre $\mathcal{C}$, entonces la distribución condicional $p(X_{V \setminus W}|x_W)$ factoriza sobre $\mathcal{C}(x_W) \subseteq \mathcal{C}$ [44]. En consecuencia, la clase generadora $\mathcal{C}(x_W)$ representa la estructura de la distribución condicional $p(X_{V \setminus W}|x_W)$. En otras palabras, la clase $\mathcal{C}(x_W)$ es *la estructura local para el contexto* x_W *en* $p(X)$.

Por lo tanto, la Definición 3.5 puede ser usada con la clase $\mathcal{C}(x_W)$ para obtener un grafo G, el cual representa la estructura local asociada al contexto x_W. Sin embargo, aunque el grafo G está representado una estructura local, el grafo G no puede asociar explícitamente dicha estructura local con un contexto. Para solucionar este problema, podemos utilizar un tipo especial de grafo, el cual asocia un grafo no dirigido con un contexto.

Definición 3.6. Sea $G = (V, E)$ un grafo no dirigido. Sea x un estado en $val(V)$. Sea W un subconjunto de V. Un grafo instanciado por x_W, denotado por el par (G, x_W), es un grafo $G = (V, E)$ donde cada nodo $w \in W$ *está etiquetado* por el estado x_w[5].

A diferencia de un grafo común, un grafo instanciado (G, x_W) permite asociar cliques con contextos. Por ejemplo, en un grafo instanciado (G, x_W), cualquier clique que involucre algún nodo en W estará asociado a uno de los estados en x_W. Así, un grafo instanciado puede codificar interacciones e independencias asociadas a un contexto. Como muestra el siguiente resultado, un grafo instanciado puede codificar independencias específicas del contexto.

[5]En base a esta definición, un grafo G no dirigido puede ser visto como un grafo instanciado por el contexto vacío, es decir, $G \equiv (G, \emptyset)$.

Teorema 3.1. (Teorema 1 [43]). Sea (G, x_W) un grafo instanciado por x_W. El grafo (G, x_W) codifica el supuesto de independencia $I(X_A \perp X_B \mid X_U, x_W)$ siempre que en (G, x_W) el conjunto de nodos A y el conjunto B estén separados por los nodos $U \cup W$.

El siguiente ejemplo ilustra cómo utilizar del Teorema 3.1 en un grafo instanciado.

Ejemplo 3.10. Sea $V = \{ a, b, c \}$ y sea X un conjunto de variables binarias. Supongamos un grafo instanciado como el mostrado por la Figura 3-4.

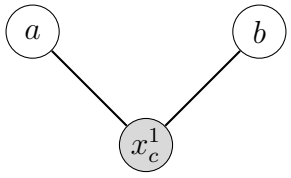


Figura 3-4: Grafo instanciado $(G, X_c = 1)$.

El grafo en la Figura 3-4 tiene el nodo c coloreado con gris, indicando que dicho nodo está etiquetado con el estado $X_c = 1$, y no con la variable X_c.

Usando separabilidad entre nodos en el grafo $(G, X_c = 1)$, tenemos que $\text{sep}_G(a, b; c)$, lo cual implica que $I(X_a \perp X_b \mid X_c)$. Sin embargo, debido a que el nodo c está etiquetado por el estado $X_c = 1$, entonces la presencia de $\text{sep}_G(a, b; c)$ implica la independencia $I(X_a \perp X_b \mid X_c = 1)$.

□

Como resultado, dada una clase generadora $\mathcal{C}$ arbitraria, un grafo instanciado (G, x_W) puede utilizarse para representar gráficamente la clase $\mathcal{C}(x_W) \subseteq \mathcal{C}$. Como hemos dicho, la clase $\mathcal{C}(x_W)$ representa la estructura de la distribución condicional $p(X_{V \setminus W} | x_W)$. Por lo tanto, el grafo (G, x_W) es una representación gráfica de la estructura de la distribución $p(X_{V \setminus W} | x_W)$.

Todo lo recientemente discutido puede ser utilizado para diferentes subconjuntos de una clase generadora. En consecuencia, una clase generadora puede ser representada por más de un grafo instanciado. El siguiente ejemplo muestra cómo utilizar más de un grafo instanciado para representar gráficamente una clase generadora.

Ejemplo 3.11. En el Ejemplo 3.6 introducimos la siguiente clase generadora:

$$\mathcal{C} = \{[X_c = 0; \{\ a, b\ \}], [X_c = 1; \{\ a\ \}], [X_c = 1; \{\ b\ \}]\}.$$

Esta clase generadora puede verse como la unión de dos clases generadoras. Por un lado,

$$\mathcal{C}(X_c = 0) = \{[X_c = 0; \{\ a, b\ \}]\},$$

y, por otro lado,

$$\mathcal{C}(X_c = 1) = \{[X_c = 1; \{\ a\ \}], [X_c = 1; \{\ b\ \}]\}.$$

La clase $\mathcal{C}(X_c = 0)$ representa la estructura de la distribución $p(X_{V\setminus\{c\}}|X_c = 0)$, mientras que la clase $\mathcal{C}(X_c = 1)$ representa la estructura de la distribución $p(X_{V\setminus\{c\}}|X_c = 1)$.

Utilizando la Definición 3.5, cada una de las clases generadoras presentadas arriba puede ser representada gráficamente. En tal caso, la clase $\mathcal{C}(X_c = 0)$ tendría un grafo que luciría exactamente igual a aquel mostrado en la Figura 3-1, mientras que el grafo para $\mathcal{C}(X_c = 1)$ luciría como aquel mostrado en la Figura 3-2.

Por lo tanto, la clase generadora $\mathcal{C}$ puede ser representada gráficamente a través de dos grafos instanciados.

□

Por lo tanto, una clase generadora $\mathcal{C}$ puede ser representada gráficamente a través de un conjunto de grafos instanciados. Sin embargo, una pregunta que surge de esta representación es ¿qué conjunto de grafos instanciados hay que utilizar para representar una clase generadora? Como veremos en la próxima sección, no existe un único conjunto de grafos instanciados para representar una clase generadora.

3.3.3. Inconveniente de una clase generadora

Como se señala en [44], **no existe un único conjunto de grafos instanciados para representar** una clase generadora. Para ilustrar esto, basta con pensar cómo

es representada graficamente una clase generadora. Según lo comentado en la Sección 3.3.2, cualquier subconjunto de generadores en una clase generadora puede ser representado gráficamente utilizando un grafo. Por lo tanto, para representar gráficamente una clase generadora, necesitamos que al menos cada uno de sus generadores esté representado por un grafo.

En base a lo comentado, una clase generadora $\mathcal{C}$ puede ser arbitrariamente dividida en un conjunto de k subconjuntos disjuntos (i.e., particiones) de generadores, e.g. $\mathcal{C}'_1, \mathcal{C}'_2, \ldots, \mathcal{C}'_k$, tal que

$$\mathcal{C} = \bigcup_{i=1}^{k} \mathcal{C}'_i,$$

entonces, cada subconjunto $\mathcal{C}'_i$ puede ser representado por un grafo. Como resultado, todos los generadores en $\mathcal{C}$ son al menos representados por un grafo.

Sin embargo, *dependiendo de cómo son particionados los generadores de una clase generadora*, obtendremos diferentes conjuntos de grafos. Por lo tanto, una clase generadora puede ser representada por tantos conjuntos de grafos como número de particiones existan en dicha clase generadora.

Un problema con la presencia de diferentes conjuntos de grafos para representar una misma clase generadora es que diferentes independencias serán obtenidas desde cada conjunto, al utilizar separación entre nodos sobre cada grafo en un conjunto. Consecuentemente, algunos conjuntos de grafos codificaran más (o menos) independencias con respecto a otros conjuntos. En otras palabras, dependiendo del conjunto de grafos con el cual es representado una clase generadora, ciertas estructuras podrían ser no visibles. Ilustremos esto último a través de un ejemplo.

Supongamos una distribución $p(X)$ cuya estructura presenta independencias específicas del contexto. Supongamos también que esta estructura es representada por una clase generadora $\mathcal{C}$. Entre todas las posibles particiones de $\mathcal{C}$, una de ellas es $\mathcal{C}(\emptyset) \equiv \mathcal{C}$. Esta partición puede ser representada por un grafo instanciado $(G, \emptyset)$, donde su contexto es $\emptyset$ (i.e., un grafo no dirigido). Sin embargo, un grafo no dirigido no puede codificar independencias específicas del contexto, ocultando las estructuras

locales en $\mathcal{C}$.

Una forma de asociar un único conjunto de grafos con una clase generadora consiste en definir un criterio para particionar una clase generadora. La próxima sección tiene como objetivo describir un criterio para particionar una clase generadora. En base a este criterio, la Sección 3.4.1 introduce una estructura de datos llamada *árbol dividido*, la cual permite representar gráficamente las particiones de una clase generadora.

3.4. Representación de una clase generadora

Dada una clase generadora $\mathcal{C}$ sobre X, una posible representación gráfica de $\mathcal{C} \equiv \mathcal{C}(\emptyset)$ es utilizar la Definición 3.5 para obtener un grafo instanciado $(G_0, \emptyset)$, donde $(G_0, \emptyset)$ es simplemente un grafo no dirigido. Sin embargo, como vimos anteriormente, dicha representación podría ocultar potenciales estructuras locales. Para enfrentar esto, podríamos utilizar más de un grafo para representar $\mathcal{C}$.

Una manera sencilla, para asociar más de un grafo instanciado a una clase generadora, es tomando una variable arbitraria, digamos $X_a \in X$, y particionar la clase generadora en dos subconjuntos tal que:

$$\mathcal{C} = \mathcal{C}(X_a = 0) \cup \mathcal{C}(X_a = 1),$$

donde cada subconjunto contiene aquellos generadores en $\mathcal{C}$ que definen las estructuras asociadas a los contextos $X_a = 0$ y $X_a = 1$, respectivamente. Más concretamente, las estructuras locales asociadas a las distribuciones $p(X_{V\setminus\{a\}}|X_a = 0)$ y $p(X_{V\setminus\{a\}}|X_a = 1)$.

Debido a que $\mathcal{C}(X_a = 0)$ es un conjunto disjunto con respecto a $\mathcal{C}(X_a = 1)$, entonces ambas clases de generadores pueden ser representadas independientemente entre sí. Usando la Definición 3.5 sobre cada partición, dos grafos instanciados pueden ser construidos, i.e., $(G_1, X_a = 0)$ y $(G_2, X_a = 1)$. A diferencia del grafo $(G_0, \emptyset)$, cada uno de estos nuevos grafos instanciados pueden representar estructuras locales asociadas a los contextos $X_a = 0$ y $X_a = 1$, respectivamente.

Sin embargo, como ocurrió inicialmente con $(G_0, \emptyset)$, los grafos instanciados $(G_1, X_a = 0)$ y $(G_2, X_a = 1)$ podrían potencialmente estar ocultando estructuras locales asociadas a otros contextos, es decir, contextos que no sólo involucran la variable X_a. En tal caso, podríamos aplicar nuevamente la estrategia anterior. En otras palabras, podríamos particionar cada uno de los subconjuntos $\mathcal{C}(X_a = 0)$ y $\mathcal{C}(X_a = 1)$, y luego asociar a cada nueva partición un grafo instanciado.

Por ejemplo, seleccionando una nueva variable, e.g. $X_b \in X_{V \setminus \{a\}}$, la clase $\mathcal{C}(X_a = 0)$ puede ser particionada de la siguiente manera:

$$\mathcal{C}(X_a = 0) = \mathcal{C}(X_a = 0, X_b = 0) \cup \mathcal{C}(X_a = 0, X_b = 1).$$

Ahora, cada una de estas dos nuevas particiones puede ser representada por los grafos instanciados $(G_3, X_a = 0, X_b = 0)$ y $(G_4, X_a = 0, X_b = 1)$, respectivamente.

Resumiendo, a través del procedimiento descripto, la clase $\mathcal{C}$ puede ser representada con el siguiente conjunto de grafos instanciados:

$$\{\ (G_0, \emptyset), (G_1, X_a = 0), (G_1, X_a = 1), (G_1, X_a = 0, X_b = 0), (G_1, X_a = 0, X_b = 1)\ \}. \tag{3.4}$$

Cada uno de estos grafos instanciados fue construido desde una partición de una clase generadora. En consecuencia, si los grafos instanciados generados no representa todas las independencias codificadas por $\mathcal{C}$, entonces nuevas particiones pueden ser generadas, las cuales pueden ser representadas por nuevos grafos instanciados.

Como veremos en lo que sigue, esta idea de representar una clase generadora a través de particiones, donde cada partición es representada por un grafo instanciado, es utilizada por una estructura de datos llamado *árbol dividido*. La Sección 3.4.1 introduce formalmente dicha estructura de datos.

3.4.1. Árbol dividido

En base al conjunto de grafos instanciados mostrados en la Ecuación (3.4), algo vale remarcar sobre ellos. Dichos grafos se encuentran organizados jerárquicamente. Más específicamente, este conjunto de grafos puede ser organizado en un árbol tal como se muestra en la Figura 3-5.

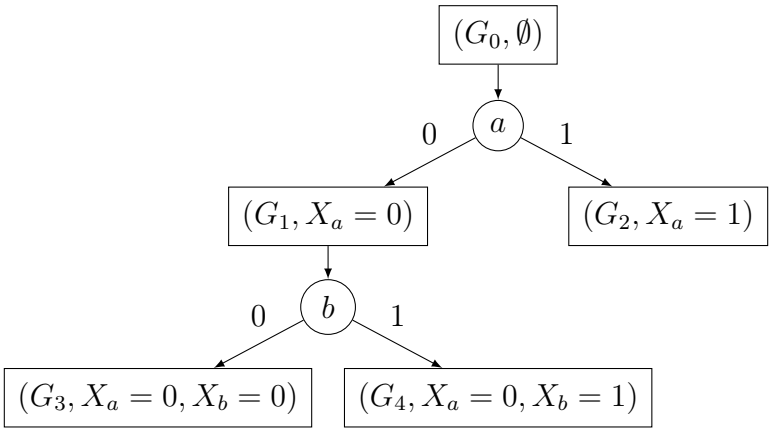


Figura 3-5: Un clase generadora $\mathcal{C}$ representada por un grafo dividido.

El árbol $\mathcal{T}$ de la Figura 3-5 es llamado *árbol dividido* (*split tree*) [43]. En un árbol dividido, hay dos tipos de nodos, los *nodos decisión* y los *nodos no decisión*. Un nodo decisión (representado por un círculo) está asociado a alguna variable, mientras que un nodo no decisión (representado por un rectángulo) está asociado a un grafo instanciado.

La raíz de un árbol dividido es siempre un nodo no decisión. Por lo tanto, la raíz de un árbol dividido siempre contiene un grafo, el cual está definido sobre un conjunto V de nodos. Dado que el conjunto V de nodos está asociado a un conjunto de variables, cualquier nodo decisión en el árbol $\mathcal{T}$ puede estar únicamente asociado a una variable en el conjunto X_V.

Todo nodo no decisión, diferente al nodo raíz, puede estar asociado a un subgrafo[6] del grafo del nodo raíz. En otras palabras, *los diferentes grafos en el árbol dividido pueden ser definidos sobre cualquier subconjunto de los nodos del grafo en el nodo raíz*. Por ejemplo, si el grafo G_0 es definido sobre un conjunto V de nodos, entonces

[6]En el Capítulo 2 (sección Notación) fue definido el concepto de subgrafo.

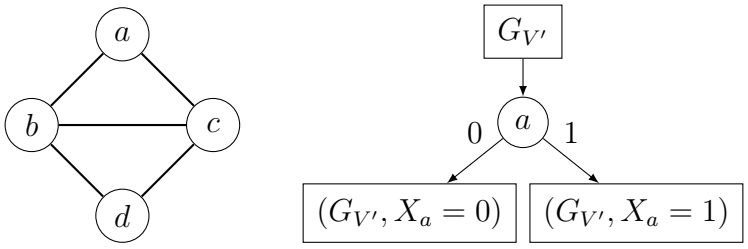
los nodos en el grafo G_1 pueden ser cualquier subconjunto en V.

Todo nodo decisión tiene como descendiente uno o más nodos no decisión, mientras que un nodo no decisión tiene como descendiente un único nodo decisión. Un nodo decisión tiene un descendiente por cada posible estado de su variable asociada. Por ejemplo, si las variables son binarias, cada nodo decisión puede tener dos descendientes. Por otro lado, un nodo no decisión únicamente puede tener como descendiente un nodo decisión, el cual puede ser asociado a alguna variable no asociada por ningún nodo decisión ancestro. Por ejemplo, en el árbol de la Figura 3-5, el nodo $(G_1, X_a = 0)$ puede tener como descendiente un nodo decisión cuya variable esté en el conjunto $X_{V\setminus\{a\}}$, mientras que, en el nodo $(G_3, X_a = 0, X_b = 0)$, su nodo decisión puede ser asociado a una variable en $X_{V\setminus\{a,b\}}$.

Un camino desde la raíz del árbol hasta cualquier nodo no decisión tiene asociado un contexto, el cual es determinado por las aristas de los diferentes nodos decisión involucrados en dicho camino. En consecuencia, al grafo asociado a cualquier nodo no decisión tiene inevitablemente un contexto asociado. Por ejemplo, en la Figura 3-5, el nodo raíz tiene asociado el contexto $\emptyset$, pero el grafo G_3 tiene asociado el contexto $X_a = 0, X_b = 0$. Por lo tanto, decimos que un nodo decisión *expande* un árbol dividido porque agrega nuevos contextos, los cuales pueden ser luego asociados a nuevos grafos (instanciados).

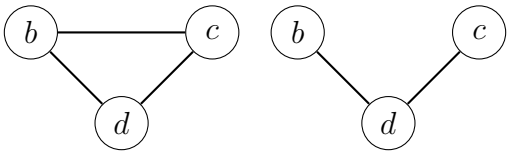
A continuación presentaremos dos ejemplos muy concretos sobre árboles divididos con el objetivo de fortalecer el entendimiento de las definiciones previas. Las Figuras 3-6 y 3-7 ilustran dos árboles divididos, los cuales denotaremos como $\mathcal{T}_0$ y $\mathcal{T}_1$, respectivamente. Cada árbol dividido tiene como raíz un grafos diferente. En el caso del árbol $\mathcal{T}_0$, el grafo raíz es el grafo $G_{V'}$, donde $V' = \{ a, b, c, d \}$. El grafo $G_{V'}$ es mostrado en la Figura 3-6a. En el caso del árbol $\mathcal{T}_0$, el grafo raíz es $G_{V''}$, donde $V'' = \{ e, f, g \}$. El grafo $G_{V''}$ es mostrado en la Figura 3-7a. Como veremos a continuación, cada uno de estos grafos pueden presentar estructuras locales, las cuales pueden ser representadas a través del árbol dividido.

Supongamos que la estructura representada por grafo $G_{V'}$ presenta la independencia $I(X_b \perp X_c \mid X_a = 1, X_d)$, la cual no puede ser codificada en el grafo $G_{V'}$.



(a) Grafo inicial $G_{V'} = (V', E)$ con $V' = \{ a, b, c, d \}$.

(b) Árbol dividido $\mathcal{T}_0$.



(c) Grafo instanciado $(G_{V'}, X_a = 0)$.

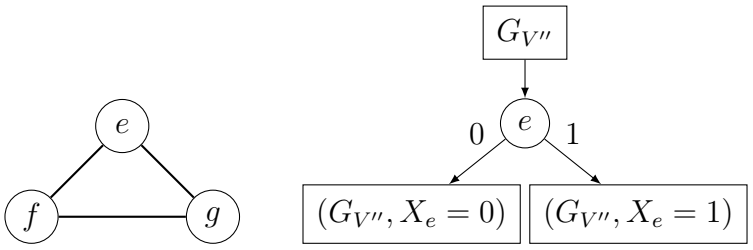
(d) Grafo instanciado $(G_{V'}, X_a = 1)$.

Figura 3-6: Árbol dividido $\mathcal{T}_0$ definido sobre el grafo $G_{V'}$, donde las hojas de $\mathcal{T}_0$ son dos grafos instanciados: $(G_{V'}, X_a = 0)$ y $(G_{V'}, X_a = 1)$.

Utilizando un árbol dividido, podemos definir un nodo decisión con la variable X_a con el objetivo de representar dicha independencia. Así, usando los estados de X_a, tenemos dos nuevos contextos que pueden ser asociados a nuevos grafos, los cuales denotaremos como $(G_{V'}, X_a = 0)$ y $(G_{V'}, X_a = 1)$, ilustrados en las Figuras 3-6c y 3-6d, respectivamente. Estos nuevos grafos instanciados nos permiten representar posibles estructuras locales no visibles en el grafo $G_{V'}$. Por ejemplo, el grafo $(G_{V'}, X_a = 1)$ permite representar la independencia $I(X_b \perp X_c \mid X_a = 1, X_d)$.

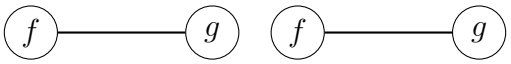
Por otro lado, supongamos que no conocemos si la estructura representada por $G_{V''}$ presenta o no estructuras locales. De este modo, utilicemos un variable aleatoria, e.g. X_e, como nodo decisión. De este modo, el árbol $\mathcal{T}_1$ asigna estados al nodo e. A diferencia del árbol dividido de la Figura 3-6, los grafos instanciados del árbol $\mathcal{T}_1$ no muestran estructuras locales. Esto puede deberse a dos causas: (1) el grafo $G_{V''}$ no presenta estructuras locales; o (2) las estructuras locales en $G_{V''}$ están asociadas a contextos que no involucran la variable X_e.

En base a todo lo discutido, un árbol dividido puede ser interpretado de dos formas diferentes pero complementarias. Una forma es únicamente pensando que cada nodo



(a) Grafo raíz $G_{V''}$, con $V'' = \{ e, f, g \}$.

(b) Árbol dividido $\mathcal{T}_1$.



(c) Grafo instanciado $(G_{V''}, X_e = 0)$.

(d) Grafo instanciado $(G_{V''}, X_e = 1)$.

Figura 3-7: Árbol dividido $\mathcal{T}_1$ definido sobre el grafo $G_{V''}$, donde las hojas de $\mathcal{T}_1$ son dos grafos instanciados: $(G_{V''}, X_e = 0)$ y $(G_{V''}, X_e = 1)$.

no decisión en el árbol dividido involucra generadores. La otra forma consiste en pensar que cada nodo no decisión involucra grafos. Veamos en detalle cada una de estas interpretaciones.

En el primer punto de vista, un árbol dividido ofrece un mecanismo para particionar una clase generadora, asociando a cada nueva partición un grafo instanciado. Si los grafos instanciados en un árbol de decisión no representan todas las independencias codificadas por $\mathcal{C}$, entonces un nuevo nodo decisión puede ser agregado. Al agregar un nuevo nodo decisión, el árbol de decisión se expande, permitiendo codificar posibles nuevas independencias. Por lo tanto, en un número finito (pero no necesariamente pequeño) de expansiones, todas las independencias presentes en $\mathcal{C}$ eventualmente serán codificadas por algún grafo instanciado en el árbol de decisión.

En el segundo punto de vista, un árbol dividido puede ser pensado como una forma de "escudriñar" un grafo con el objetivo de "visualizar" estructuras locales ocultas en dicho grafo. De este modo, la raíz del árbol sería el grafo a escudriñar. Mediante los nodos decisión en el árbol, podemos definir contextos que nos permitan instanciar el grafo raíz con un contexto en particular. Al instanciar el grafo raíz nos abre la posibilidad de hacer visible ciertas estructuras. Por ejemplo, este es el caso mostrado en la Figura 3-6, donde el grafo $(G_{V'}, X_a = 1)$ permitió visualizar una estructura local no visible en el grafo $G_{V'}$.

Un árbol dividido es utilizado por los modelos divididos, los cuales son introducidos en la Sección 3.5. La estructura de un modelo dividido puede ser representada por varios árboles divididos. Como mostraremos en la Sección 3.5.2, la construcción de la estructura de un modelo dividido presenta varias dificultades. Como una alternativa para evitar estas dificultades, proponemos los *modelos canónicos*, los cuales serán presentados en la Sección 3.6. Los modelos canónicos no utilizan un árbol de decisión para representar una clase generadora, simplificando considerablemente su construcción desde un conjunto de datos.

3.5. Modelos divididos

Un modelo dividido es un tipo particular de modelo CSI. En otras palabras, un modelo dividido es una familia de distribuciones de probabilidad (ver la Definición 3.4), donde cada una factoriza sobre una determinada clase generadora. La clase generadora de un modelo dividido, es decir su estructura, es representada utilizando una estructura de datos llamada *grafo divido* (*split graph*). En términos sencillos, un grafo dividido es una colección de árboles divididos.

Esta sección tiene como objetivo introducir el concepto de grafo dividido y presentar las dificultades de construir un grafo dividido desde un conjunto de datos. Por lo tanto, la sección está dividida en dos partes. La primera parte, la Sección 3.4.1, presenta formalmente un grafo dividido, es decir, la estructura de un modelo dividido. La segunda parte, la Sección 3.5.2 presenta algunas de las dificultades de utilizar un modelo dividido como representación de la estructura de una distribución. Estas dificultades motivarán nuestra propia propuesta para representar un modelo CSI, los modelos canónicos, los cuales serán presentados en la Sección 3.6.

3.5.1. Grafo dividido

Formalmente un grafo dividido, denotado por $\mathcal{SG} = (G_0, \{ \mathcal{T}_0, \mathcal{T}_1, \dots \})$, es un par formado por un grafo instanciado G_0 y una lista de árboles divididos $\{ \mathcal{T}_0, \mathcal{T}_1, \dots \}$, donde cada árbol dividido en la lista tiene como grafo raíz algún subgrafo de G_0.

Veamos un ejemplo concreto de un grafo dividido.

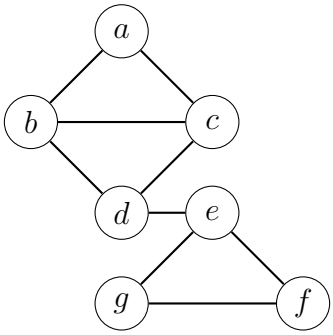


Figura 3-8: Un grafo $G = (V, E)$, donde $V = \{\ a, b, c, d, e, f, g\ \}$.

La Figura 3-8 muestra un grafo G que usaremos para definir un grafo dividido $\mathcal{SG}_0$. En base al grafo G, tenemos que definir una lista de árboles divididos, asociando a algún subgrafo en G un árbol dividido. Consideremos dos subgrafos de G. El subgrafo que involucra el conjunto $V' = \{\ a, b, c, d\ \} \subset V$, el cual denotaremos como $G_{V'}$. Y, el subgrafo que involucra el conjunto $V'' = \{\ e, f, g\ \} \subset V$, el cual denotaremos como $G_{V''}$.

Usando como raíz cada uno de estos grafos, definiremos dos árboles divididos, los cuales denotaremos como $\mathcal{T}_0$ y $\mathcal{T}_1$, respectivamente. Cada uno de estos árboles divididos podrían lucir exactamente igual a aquellos mostrados en la Figura 3-6 y la Figura 3-7, respectivamente. Como vimos en la Sección 3.4.1, el árbol dividido $\mathcal{T}_0$ involucra dos grafos instanciados, el grafo $(G_{V'}, X_a = 0)$ y $(G_{V'}, X_a = 1)$, mientras que el árbol dividido $\mathcal{T}_1$ involucra los grafos $(G_{V''}, X_e = 0)$ y $(G_{V''}, X_e = 1)$.

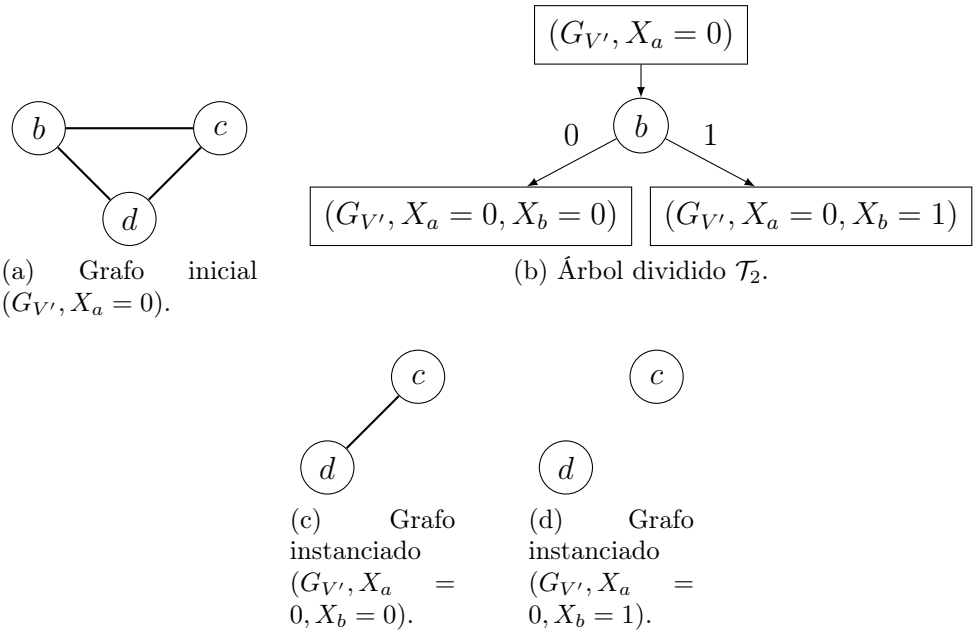
De esta manera, el grafo dividido $\mathcal{SG}_0$ puede ser formalmente definido como:

$$
\begin{aligned}
\mathcal{SG}_0 &= (G, \{\ \mathcal{T}_0, \mathcal{T}_1\ \}), \\
\mathcal{T}_0 &= \{\ (G_{V'}, X_a = 0), (G_{V'}, X_a = 1)\ \}, \\
\mathcal{T}_1 &= \{\ (G_{V''}, X_e = 0), (G_{V''}, X_e = 1)\ \},
\end{aligned}
$$

donde cada árbol dividido es denotado como una lista conformada por sus respectivos grafos instanciados.

Adicionalmente, como se muestra en [43], la definición de un grafo dividido puede ser generalizada para permitir recursión. La ventaja de esta generalización es que un modelo dividido puede representar estructuras mucho más complejas. La generalización consiste en la siguiente observación. Un grafo dividido con una lista vacía de árboles divididos es simplemente un grafo instanciado. Por otro lado, un grafo instanciado con un contexto vacío es simplemente un grafo no dirigido.

En base a la última observación, un grafo dividido puede ser definido a partir de otro grafo dividido base (al fin y al cabo un grafo dividido es un grafo). En consecuencia, la lista de árboles divididos del nuevo grafo dividido pueden tener como raíz cualquier subgrafo de los grafos instanciados en el grafo dividido base. Veamos un ejemplo.



(a) Grafo inicial $(G_{V'}, X_a = 0)$.

(b) Árbol dividido $\mathcal{T}_2$.

(c) Grafo instanciado $(G_{V'}, X_a = 0, X_b = 0)$.

(d) Grafo instanciado $(G_{V'}, X_a = 0, X_b = 1)$.

Figura 3-9: Árbol dividido $\mathcal{T}_2$ definido sobre el grafo $(G_{V'}, X_a = 0)$, donde las hojas de $\mathcal{T}_2$ son dos grafos instanciados: $(G_{V'}, X_a = 0, X_b = 0)$ y $(G_{V'}, X_a = 1, X_b = 0)$.

Para ilustrar la generalización en la definición de un grafo dividido, primero supongamos lo siguiente. Supongamos que el grafo instanciado $(G_{V'}, X_a = 0))$ en el árbol $\mathcal{T}_0$ mostrado en la Figura 3-6 representa una estructura local aún no representada. Además supongamos que esta estructura local contiene la independencia $I(X_c \perp X_d \mid X_a = 0, X_b = 1)$.

Para representar esta estructura local, usando el grafo $(G_{V'}, X_a = 0))$ como raíz, podemos definir un árbol con un nodo decisión con la variable X_b. Este árbol de decisión resultará en dos nuevos grafos instanciados, donde uno de ellos estará instanciado por el contexto $X_a = 0, X_b = 1$. De este modo, la independencia $I(X_c \perp X_d \mid X_a = 0, X_b = 1)$ puede ser codificada en uno de los nuevos grafos instanciados. La Figura 3-9 muestra este nuevo árbol dividido el cual denotaremos como $\mathcal{T}_2$.

Utilizando el grafo $(G_{V'}, X_a = 0))$ y el árbol dividido $\mathcal{T}_2$ podemos definir un nuevo grafo dividido como:

$$\begin{aligned}\mathcal{SG}_1 &= ((G_{V'}, X_a = 0), \{\ \mathcal{T}_2\ \}),\\ \mathcal{T}_2 &= \{\ (G_{V'}, X_a = 0, X_b = 0), (G_{V'}, X_a = 0, X_b = 1)\ \}.\end{aligned}$$

Usando un grafo dividido generalizado, podemos combinar el grafo dividido $\mathcal{SG}_0$ y el grafo dividido $\mathcal{SG}_1$ en un único grafo dividido definido sobre G como sigue:

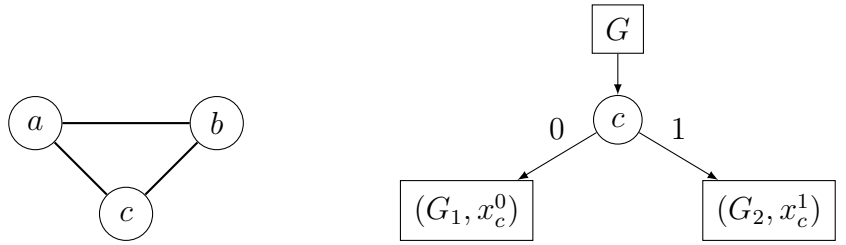
$$\begin{aligned}\mathcal{SG}_0 &= (G, \{\ \mathcal{T}_0, \mathcal{T}_1\ \}),\\ \mathcal{T}_0 &= \{\ \mathcal{SG}_1, (G_{V'}, X_a = 1)\ \},\\ \mathcal{T}_1 &= \{\ (G_{V''}, X_e = 0), (G_{V''}, X_e = 1)\ \}.\end{aligned}$$

Para finalizar, mostraremos cómo utilizar un grafo dividido para representar tanto la estructura global como las estructuras locales de una distribución.

Ejemplo 3.12. La estructura de la distribución $p(X)$ del Ejemplo 3.1 puede ser representada utilizando un grafo dividido $\mathcal{SG} = (G, \{\ \mathcal{T}\ \})$, donde el grafo G es la estructura global de $p(X)$ y el árbol dividido $\mathcal{T}$ permitirá representar la estructura local de $p(X)$. Como hemos visto, el grafo G no puede codificar la estructura local de

$p(X)$, la cual consiste en el supuesto $I(X_a \perp X_b \mid X_c = 1)$.

La Figura 3-10 muestra una representación gráfica de $\mathcal{SG}$, donde el grafo G es mostrado en la Figura 3-10a, mientras que el árbol dividido es mostrado en la Figura 3-10b.

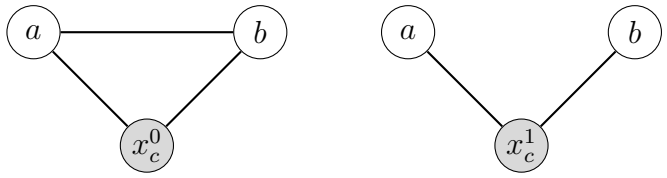


(a) Estructura global codificada por un grafo.

(b) Estructura local codificada por un árbol dividido.

Figura 3-10: El modelo dividido representando la estructura de una distribución.

El grafo dividido $\mathcal{T}$ tiene como raíz el grafo G y como nodo decisión la variable $X_c \in X$. Este nodo decisión produce dos grafos instanciados, uno instanciado por el contexto $X_c = 0$ y otro por el contexto $X_c = 1$. Cada uno de estos grafos instanciados codifican posibles estructuras locales para cada contexto de la variable X_c. La Figura 3-11 muestra en detalle los grafos instanciados en el árbol dividido $\mathcal{T}$.



(a) Grafo instanciado $(G_1, x_c = 0)$.

(b) Grafo instanciado $(G_2, X_c = 1)$.

Figura 3-11: La estructura local de una distribución representada con dos grafos instanciados. Los nodos de color gris denotan nodos instanciados.

Los grafos instanciados $(G_1, X_c = 0)$ y $(G_2, X_c = 1)$ son subgrafos de G, puntualmente G_1 y G_2 son subgrafos que involucran todos los nodos de G. El grafo instanciado $(G_1, X_c = 0)$ codifica la estructura de la distribución $p(X_{V\setminus\{c\}}|X_c = 0)$, mientras que el grafo $(G_2, X_c = 1)$ codifica la estructura de la distribución $p(X_{V\setminus\{c\}}|X_c = 1)$. Sabemos que la distribución $p(X_{V\setminus\{c\}}|X_c = 1)$ presenta la independencia $I(X_a \perp X_b \mid \emptyset)$

que se corresponde con el supuesto $I(X_a \perp X_b \mid X_c = 1)$.

Debido que no hay independencias en el contexto $X_c = 0$, entonces el grafo $(G_1, X_c = 0)$ no codifica supuestos de independencia, es decir, $G_1 = G$. Por otro lado, el contexto $X_c = 1$ presenta la independencia $I(X_a \perp X_b \mid X_c = 1)$, en consecuencia, dado que el grafo $(G_2, X_c = 1)$ cumple $\text{sep}_{G_2}(\{ a \}, \{ b \}; \{ c \})$, entonces el supuesto $I(X_a \perp X_b \mid X_c = 1)$ es codificado.

Como resultado, utilizando un modelo dividido como representación de la estructura, podemos codificar simultáneamente la estructura global y las estructuras locales de una distribución.

□

3.5.2. Construcción de un modelo dividido

Como hemos visto en la Sección 3.5.1, la estructura de un modelo dividido, un grafo dividido, permite codificar tanto la estructura global como las estructuras locales de una distribución. Sin embargo, la construcción de un grafo divido presenta grandes dificultes. El objetivo de esta sección es detallar cuáles son estas dificultes.

Para construir un grafo dividido, e.g. $\mathcal{SG} = (G_0, \{ \mathcal{T}_0, \mathcal{T}_1, \dots \})$, desde un conjunto de datos, el problema puede ser dividido en la construcción de un grafo G_0 y, por otro lado, la construcción de una lista de árboles divididos.

Si el grafo G_0 es un grafo no dirigido, como vimos en el Ejemplo 3.12, éste puede ser construido utilizando cualquier algoritmo de aprendizaje de estructuras (e.g., un algoritmo basado en restricciones). Pero, si el grafo es un grafo instanciado, existe un algoritmo especializado para construir un grafo instanciado, el cual es presentado en [43]. Por lo tanto, la construcción del grafo G_0 no presenta dificultades.

En comparación al grafo G_0, la construcción de una lista de árboles divididos es más compleja. Para mostrar esta complejidad, analizaremos dos cuestiones. Dado un grafo G_0, la primer cuestión es ¿cuántos árboles divididos asociar al grafo G_0? Por ejemplo, en el Ejemplo 3.12, solamente utilizamos un único árbol dividido. En contraste, en la Sección 3.5.1, el grafo dividido $\mathcal{SG}_0$ fue inicialmente definido en base a dos árboles divididos, cada uno asociado a diferentes subgrafos de G (el cual es

mostrado en la Figura 3-8).

Similarmente, la pregunta anterior puede ser reformulada como ¿cuál subgrafo del grafo G_0 se debería utilizar para construir los árboles divididos? Una respuesta sencilla sería asociar un árbol de decisión por cada posible subgrafo en G_0, pero esto es intratable. Dado un grafo G_0 con $|V| = n$ nodos, tenemos $\sum_{i=3}^{n} \binom{n}{i} \approx 2^n$ posibles subgrafos de interés que podrían ser asociados a un árbol dividido.

La segunda cuestión es la decisión de cuál variable asociar a un nodo decisión en un árbol dividido. Tal como se comenta en [44], dicha decisión impone fuertes restricciones en la codificación de las estructuras locales. Por ejemplo, en el Ejemplo 3.12, si la variable X_c no fuese seleccionada como nodo decisión en el árbol dividido, entonces la estructura local mostrada en la Figura 3-11 no podría haber sido visible. Esto se debe a que dicha estructura local codifica una independencia específica cuyo contexto es $X_c = 1$. En consecuencia, para que dicha variable tome tal estado, la variable X_c debe estar asociada a un nodo decisión en el árbol dividido.

Estas cuestiones con respecto a la construcción de los árboles divididos, en el mejor de nuestro conocimiento, no han sido abordadas por la literatura. Por ejemplo, analizando los trabajos [43, 44], donde se proponen los modelos divididos, solamente se presenta un método para construir un grafo instanciado, el cual no puede ser utilizado para construir un grafo dividido y, por ende, tampoco una lista de árboles divididos.

Incluso al revisar otros trabajos que citan a [43, 44], encontramos los *hierarchical space models* [40], una generalización de los modelos divididos. Sin embargo, el trabajo [40] no presenta un método para construir un árbol dividido.

Nuestra perspectiva es que gran parte de los trabajos relacionados con el aprendizaje de estructuras locales en redes de Markov han abordado el problema sin la utilización de los modelos divididos. Como comentamos en la Sección 3.2.2, muchos trabajos asumen distribuciones descomponibles, lo cual simplifica considerablemente el aprendizaje de estructuras locales [68, 69, 70]. Por otro lado, otro gran número de trabajos utilizan un conjuntos de características como representación de la estructura [18, 89, 56].

En este trabajo no pretendemos resolver directamente los problemas para construir

un grafo dividido. En cambio, proponemos una forma alternativa de representar una clase generadora. Esta propuesta permite evitar los problemas que presenta un grafo dividido. Como resultado, representando una clase generadora a través de nuestra propuesta, una clase generadora puede ser aprendida por cualquier algoritmo basado en restricciones.

3.6. Modelos canónicos

En términos sencillos, los modelos canónicos son una clase especial de modelo CSI. Por ejemplo, la estructura de un modelo canónico es esencialmente una clase generadora. Debido a que un modelo dividido también es un modelo CSI, en cierto sentido, un modelo canónico es similar a un modelo dividido. Por ejemplo, en ambos modelos, una clase generadora es representada por un conjunto de grafos instanciados.

Sin embargo, la principal diferencia entre un modelo dividido y un modelo canónico es que un modelo canónico *no utiliza un grafo dividido* para representar una clase generadora. Por lo tanto, la estructura de un modelo canónico puede pensarse estrictamente como un *conjunto* de grafos instanciados, donde los grafos instanciados no se encuentran bajo ninguna organización en particular.

En esta sección introduciremos formalmente los modelos canónicos. Para esto, hemos decidido estructurar esta sección en tres partes. La Sección 3.6.1 presenta una descripción intuitiva de la estructura de un modelo canónico. La Sección 3.6.2 introduce formalmente los modelos canónicos y su estructura. La Sección 3.6.3 presenta cómo representar gráficamente la estructura de un modelo canónico.

En el próximo capítulo, presentaremos un algoritmo para aprender la estructura de un modelo canónico desde un conjunto de datos.

3.6.1. Introducción

Esta sección tiene como objetivo ofrecer una idea aproximada, no necesariamente precisa, sobre la estructura de un modelo canónico. Para esto, retomaremos nuestra discusión de la Sección 3.4, en la cual discutimos cómo representar gráficamente una

clase generadora.

Como vimos en dicha sección, dicha representación es obtenida a través del siguiente procedimiento. Seleccionado una variable del dominio, los estados de dicha variable pueden utilizarse para particionar una clase generadora. Luego, cada partición puede ser representada por un grafo instanciado. Usando como base alguna de las particiones obtenidas, y seleccionando una nueva variable del dominio, el procedimiento anterior puede repetirse para obtener nuevos grafos instanciados.

Un problema con este procedimiento es que, en base a las variables utilizadas para particionar una clase generadora, algunas estructuras locales podrían no resultar visibles en los grafos instanciados. Un ejemplo de esto fue ilustrado cuando introducimos los grafos divididos en la Sección 3.5.1. Por ejemplo, en la Figura 3-7, el árbol dividido $\mathcal{T}_0$ tiene como descendiente el grafo $(G_{V'}, X_a = 0)$. Este grafo no muestra ninguna estructura local asociada al contexto $X_a = 0$, porque todos sus nodos están conectados. Sin embargo, en la Figura 3-9, el árbol dividido $\mathcal{T}_2$ utiliza como raíz el grafo $(G_{V'}, X_a = 0)$ y utiliza la variable X_b como nodo decisión. Este árbol tiene como descendiente el grafo $(G_{V'}, X_a = 0, X_b = 1)$. Dicho grafo muestra una estructura local asociada al contexto $X_a = 0, X_b = 1$, la cual se encontraba oculta en el grafo $(G_{V'}, X_a = 0)$.

Una solución a este problema podría ser la utilización simultanea de todas las variables del dominio para particionar una clase generadora. Esto produciría un contexto donde todas las variables del dominio tienen un estado asignado. Formalmente, este estado es llamado estado canónico. En nuestro caso, se trata de un *contexto canónico*.

Dada n variables binarias, hay 2^n contextos canónicos. Usando este conjunto de contextos canónicos, podemos particionar una clase generadora de la siguiente manera. Tomando un contexto canónico, asociamos a dicho contexto aquellos generadores que lo satisfacen. Luego, usando los restantes generadores, podemos tomar otro contexto canónico, sobre el cual podemos asociar nuevos generadores. Este proceso puede repetirse hasta que eventualmente no queden más generadores disponibles.

Es importante notar que, a pesar que existe un número exponencial de contextos,

según cuáles contextos canónicos son usados para particionar una clase generadora, una clase generadora puede particionarse en un número no necesariamente exponencial de particiones.

Las particiones obtenidas en base a lo discutido previamente pueden ser representadas gráficamente a través de un conjunto de grafos instanciados. Este conjunto de grafos instanciados presenta dos características. Primero, los grafos instanciados no están organizados en ninguna manera en particular. Por ejemplo, en un árbol dividido, los grafos instanciados están organizados jerárquicamente. Segundo, todo grafo está instanciado por un contexto canónico. Llamamos *grafo canónico* a un grafo instanciado por un contexto canónico.

Informalmente, la estructura de un modelo canónico puede pensarse como un conjunto de grafos canónicos. Sin embargo, como veremos más adelante, la estructura de un modelo canónico es mucho más general que lo recientemente discutido. Por ejemplo, en el caso anterior, cada grafo canónico es definido a partir de una partición de una clase generadora. En consecuencia, cada grafo canónico es definido por un conjunto de generadores no presente en ningún otro grafo canónico. En contraste, la estructura de un modelo canónico permite la presencia de grafos canónicos cuyos generadores hayan sido usados para definir otros grafos canónicos.

3.6.2. Definiciones y notaciones

En esta sección introduciremos los modelos canónicos, la base de nuestra propuesta para representar una clase generadora de una forma alternativa. Para esto, empezaremos describiendo la estructura de un modelo canónico, la cual denominamos como *ensamble canónico*. Una vez introducido el concepto de ensamble canónico, presentaremos formalmente la definición de un modelo canónico.

En términos muy sencillos, un ensamble canónico es una clase generadora que puede ser representado gráficamente sin la necesidad de utilizar un grafo dividido. Por lo tanto, al construir un ensamble canónico desde un conjunto de datos, los problemas descriptos en la Sección 3.5.2 pueden ser evitados.

Siendo más específico, un ensamble canónico es una clase generadora cuyos genera-

dores tienen una característica bastante particular, involucran únicamente un estado de un subconjunto de variables. Este tipo de generadores son formalmente definidos como sigue:

Definición 3.7. Sea $[x_B^k; A]$ un generador sobre X. Si $A = \emptyset$ entonces $[x_B^k; A] \equiv [x_B^k]$ es un *generador específico*.

Diremos que una clase generadora sobre X es específica, denotada por $\widetilde{\mathcal{C}}$, si todos sus generadores son específicos.

Utilizar una clase generadora específica, en vez de una clase generadora no específica, permite resolver uno de los problemas planteados en la Sección 3.6.3, sobre cuál variable utilizar en un nodo decisión en un árbol dividido.

Para representar un generador no específico, es necesario un grafo instanciado donde algunos de sus nodos estén asociados a estados y otros no. Como consecuencia, al aprender un grafo instanciado, es necesario decidir cuáles de sus nodos asociar a un estado y cuáles no. Por ejemplo, usando un nodo decisión en un árbol dividido. Al considerar una clase generadora específica, este problema puede ser evitado, debido a que todos sus generadores son específicos, no hay variables asignadas y no asignadas.

Una clase generadora específica puede verse como un tipo especial de clase generadora. Sin embargo, *toda clase generadora no específica puede ser expresada como una clase generadora específica*. Para ver esto, cualquier generador no específico, e.g. $[x_B^k; A]$, puede ser expresado como un conjunto de generadores específicos, tal como se muestra en la siguiente ecuación:

$$[x_B^k; A] = \{\, [x_A^j \wedge x_B^k; \emptyset] \colon x_A^j \in val(A), j = 0, \ldots, |val(A)| - 1 \,\}\,. \tag{3.5}$$

En palabras, dado un generador $[x_B^k; A]$, este puede ser expresado como un conjunto de generadores específicos, uno por cada posible estado de las variables X_A.

Como consecuencia de la Ecuación 3.5, cualquier clase generadora $\mathcal{C}$ sobre X puede ser expresada como una nueva clase generadora $\widetilde{\mathcal{C}}$, donde todos sus generadores son específicos. Para ilustrar esto último, dada una clase generadora $\mathcal{C}$ arbitraria,

la Ecuación 3.5 puede ser aplicada a cada uno de sus generadores para obtener la siguiente clase generadora específica:

$$\widetilde{\mathcal{C}} = \{\ [x_A^0 \wedge x_B^k; \emptyset], [x_A^1 \wedge x_B^k; \emptyset], \ldots, [x_A^{|val(A)|-1} \wedge x_B^k; \emptyset] \colon [x_B^k; A] \in \mathcal{C}\ \}\,.$$

Para minimizar toda posible duda referente a los conceptos recientemente introducidos, presentamos un ejemplo ilustrativo.

Ejemplo 3.13. Este ejemplo ilustra los conceptos de generador específico, clase generadora específica y cómo una clase generadora arbitraria puede ser expresada como una clase generadora específica. Supongamos un dominio con las siguientes variables $X = \{\ X_a, X_b, X_c, X_f\ \}$ y una clase generadora sobre X definida como sigue:

$$\begin{aligned}\mathcal{C} = \{&[X_f = 0; \{\ a, b\ \}],\\ &[X_f = 0; \{\ a, c\ \}],\\ &[X_f = 0; \{\ b, c\ \}],\\ &[X_f = 1; \{\ a\ \}],\\ &[X_f = 1; \{\ b\ \}],\\ &[X_f = 1; \{\ c\ \}]\}.\end{aligned} \tag{3.6}$$

Utilizando la Ecuación 3.5 podemos expresar cada uno de los generadores en $\mathcal{C}$ como un conjunto de generadores específicos, resultando en la siguiente clase específica:

$$\begin{aligned}\widetilde{\mathcal{C}} = \{&[X_a = 0 \wedge X_b = 0 \wedge X_f = 0], [X_a = 0 \wedge X_b = 1 \wedge X_f = 0], \\ &[X_a = 1 \wedge X_b = 0 \wedge X_f = 0], [X_a = 1 \wedge X_b = 1 \wedge X_f = 0], \\ &[X_a = 0 \wedge X_c = 0 \wedge X_f = 0], [X_a = 0 \wedge X_c = 1 \wedge X_f = 0], \\ &[X_a = 1 \wedge X_c = 0 \wedge X_f = 0], [X_a = 1 \wedge X_c = 1 \wedge X_f = 0], \\ &[X_b = 0 \wedge X_c = 0 \wedge X_f = 0], [X_b = 0 \wedge X_c = 1 \wedge X_f = 0], \\ &[X_b = 1 \wedge X_c = 0 \wedge X_f = 0], [X_b = 1 \wedge X_c = 1 \wedge X_f = 0], \\ &[X_a = 0 \wedge X_f = 1], [X_a = 1 \wedge X_f = 1], \\ &[X_b = 0 \wedge X_f = 1], [X_b = 1 \wedge X_f = 1], \\ &[X_c = 0 \wedge X_f = 1], [X_c = 1 \wedge X_f = 1]\}.\end{aligned} \tag{3.7}$$

□

De esta manera, al tener una clase generadora específica, cada una de sus estructuras locales se encuentran asociadas a diferentes contextos canónicos. En consecuencia, dada una clase generadora específica $\widetilde{\mathcal{C}}$ y un contexto canónico $x^k \in val(V)$, el conjunto $\widetilde{\mathcal{C}}(x^k) \subseteq \widetilde{\mathcal{C}}$ es la estructura local asociada al contexto canónico x^k.

En base a esto, debido a que una clase generadora específica puede posiblemente presentar más de una estructura local, cada estructura local estará asociada a un contexto canónico diferente. De este modo, dada una clase $\widetilde{\mathcal{C}}$, existe un conjunto $\mathcal{X} \subseteq val(V)$ de contextos canónicos, tal que cada contexto $x^k \in \mathcal{X}$ tiene asociado una de las estructuras locales en $\widetilde{\mathcal{C}}$. Veamos un ejemplo sobre esto último.

Ejemplo 3.14. Para mostrar todas las estructuras locales asociadas a la clase generadora $\widetilde{\mathcal{C}}$ mostrada en la Ecuación (3.6) del Ejemplo 3.13. Utilizaremos el siguiente conjunto $\mathcal{X}$ de contextos canónicos:

$$\begin{aligned}\mathcal{X} = \{&x^0 \equiv (X_a = 0 \wedge X_b = 0 \wedge X_c = 0 \wedge X_f = 0),\\ &x^1 \equiv (X_a = 1 \wedge X_b = 1 \wedge X_c = 1 \wedge X_f = 0),\\ &x^2 \equiv (X_a = 0 \wedge X_b = 1 \wedge X_c = 1 \wedge X_f = 0),\\ &x^3 \equiv (X_a = 1 \wedge X_b = 0 \wedge X_c = 0 \wedge X_f = 0),\\ &x^4 \equiv (X_a = 0 \wedge X_b = 1 \wedge X_c = 0 \wedge X_f = 0),\\ &x^5 \equiv (X_a = 0 \wedge X_b = 0 \wedge X_c = 1 \wedge X_f = 0),\\ &x^6 \equiv (X_a = 0 \wedge X_b = 0 \wedge X_c = 0 \wedge X_f = 1),\\ &x^7 \equiv (X_a = 1 \wedge X_b = 1 \wedge X_c = 1 \wedge X_f = 1)\}.\end{aligned}$$

El conjunto $\mathcal{X}$ es un subconjunto del conjunto $val(\{\ a, b, c, f\ \})$, donde $|\mathcal{X}| = 8$ y $|val(\{\ a, b, c, f\ \})| = 2^4$. Usando este conjunto $\mathcal{X}$, la clase generadora $\widetilde{\mathcal{C}}$ presenta las siguientes estructuras locales:

$$\begin{aligned}
\widetilde{\mathcal{C}}(x^0) &= \{\ [X_a = 0, X_b = 0, X_f = 0], [X_a = 0, X_c = 0, X_f = 0], [X_b = 0, X_c = 0, X_f = 0]\ \},\\
\widetilde{\mathcal{C}}(x^1) &= \{\ [X_a = 1, X_b = 1, X_f = 0], [X_a = 1, X_c = 1, X_f = 0], [X_b = 1, X_c = 1, X_f = 0]\ \},\\
\widetilde{\mathcal{C}}(x^2) &= \{\ [X_a = 0, X_b = 1, X_f = 0], [X_a = 0, X_c = 1, X_f = 0], [X_b = 1, X_c = 1, X_f = 0]\ \},\\
\widetilde{\mathcal{C}}(x^3) &= \{\ [X_a = 1, X_b = 0, X_f = 0], [X_a = 1, X_c = 0, X_f = 0], [X_b = 0, X_c = 0, X_f = 0]\ \},\\
\widetilde{\mathcal{C}}(x^4) &= \{\ [X_a = 0, X_b = 1, X_f = 0], [X_a = 0, X_c = 0, X_f = 0], [X_b = 1, X_c = 0, X_f = 0]\ \},\\
\widetilde{\mathcal{C}}(x^5) &= \{\ [X_a = 0, X_b = 0, X_f = 0], [X_a = 0, X_c = 1, X_f = 0], [X_b = 0, X_c = 1, X_f = 0]\ \},\\
\widetilde{\mathcal{C}}(x^6) &= \{\ [X_a = 0, X_f = 1], [X_b = 0, X_f = 1], [X_c = 0, X_f = 1]\ \},\\
\widetilde{\mathcal{C}}(x^7) &= \{\ [X_a = 1, X_f = 1], [X_b = 1, X_f = 1], [X_c = 1, X_f = 1]\ \}.
\end{aligned}$$

□

Debido a que las estructuras locales de una clase generadora específica están aso-

ciadas a contextos canónicos, definimos la estructura de un modelo canónico en base a esta observación. Formalmente, la estructura de un modelo canónico la denominamos *ensamble canónico*.

Definición 3.8. Sea $\widetilde{\mathcal{C}}$ una clase generadora específica sobre X. Sea $\mathcal{X}$ un conjunto de estados canónicos, $\mathcal{X} \subseteq val(V)$. Un *ensamble canónico* es un par $\bar{\mathcal{C}} = (\widetilde{\mathcal{C}}, \mathcal{X})$.

En palabras, un ensamble canónico $\bar{\mathcal{C}}$ es un par $(\widetilde{\mathcal{C}}, \mathcal{X})$ que define un conjunto de contextos canónicos asociados a las estructuras locales en una clase $\widetilde{\mathcal{C}}$. Dado un ensamble canónico $\bar{\mathcal{C}} = (\widetilde{\mathcal{C}}, \mathcal{X})$, es posible definir una clase generadora $\mathcal{C}$ "ensamblando" cada una de las estructuras locales presentes en $\widetilde{\mathcal{C}}$, es decir,

$$\mathcal{C} = \bigcup_{x^k \in \mathcal{X}} \widetilde{\mathcal{C}}(x^k). \tag{3.8}$$

El resultado de la Ecuación 3.8 es sumamente importante porque muestra que un ensamble canónico tiene asociado una clase generadora. Como mostramos en la Sección 3.3, una clase generadora es la estructura de una distribución de probabilidad, en consecuencia, un ensamble canónico es también la estructura de una distribución de probabilidad.

Como muestra la Definición 3.3, una clase generadora $\mathcal{C}$ (e.g. aquella obtenida desde un ensamble canónico $\bar{\mathcal{C}}$) puede ser utilizada para factorizar una distribución $p(X)$. En base a este hecho, definimos formalmente un modelo canónico como sigue:

Definición 3.9. Sea $\bar{\mathcal{C}}$ un ensamble canónico sobre X. Sea $\mathcal{C}$ la clase generadora obtenida desde $\bar{\mathcal{C}}$ por la Ecuación (3.8). Un *modelo canónico* con respecto a $\bar{\mathcal{C}}$ es la familia de distribuciones que factoriza sobre la clase $\mathcal{C}$.

De esta manera, una distribución de probabilidad puede ser representada por un modelo canónico, cuya estructura es un ensamble canónico. Un ensamble canónico se caracteriza por asociar las estructuras locales de una distribución a un conjunto de contextos canónicos. Como veremos a continuación, cada una de estas estructu-

ras locales puede ser representada utilizando un conjunto muy particular de grafos instanciados.

3.6.3. Representación gráfica de un ensamble canónico

La estructura de un modelo canónico es un ensamble canónico. Un ensamble canónico $\bar{\mathcal{C}} = (\widetilde{\mathcal{C}}, \mathcal{X})$ está formado por una clase generadora específica $\widetilde{\mathcal{C}}$ y un conjunto $\mathcal{X}$ de contextos canónicos. Cada contexto canónico $x^k \in \mathcal{X}$ tiene asociado una estructura local definida por la clase $\widetilde{\mathcal{C}}(x^k)$. En consecuencia un ensamble canónico está formado por $|\mathcal{X}|$ estructuras locales, donde cada estructura local puede ser gráficamente representada por un grafo instanciado. Como resultado, *cada grafo instanciado está asociado a un contexto canónico*. Esto es esquemáticamente ilustrado en la Figura 3-12.

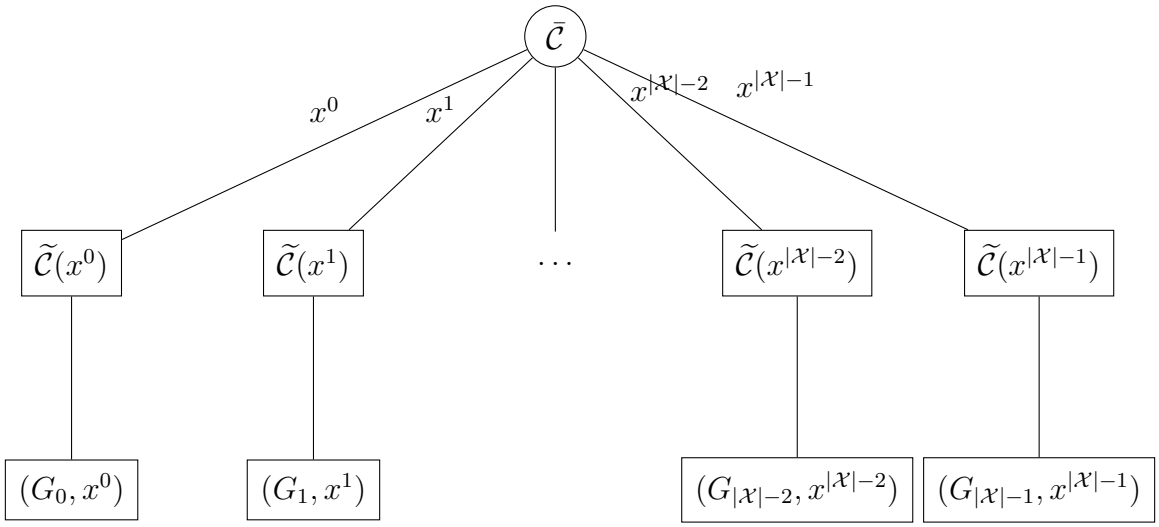


Figura 3-12: Representación esquemática de un ensamble canónico.

Los grafos instanciados utilizados para representar un ensamble canónico se caracterizan por tener todos sus nodos etiquetados. La siguiente definición formaliza este hecho:

Definición 3.10. Sea (G, x^k) un grafo instanciado. El grafo instanciado (G, x^k) es *canónico*, denotado por G^k, si todo nodo $a \in V$ *está etiquetado* con un estado en $val(a)$.

La Figura 3-13 ilustra un grafo canónico arbitrario.

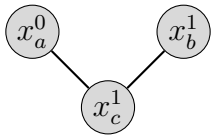


Figura 3-13: Un grafo canónico $(G, x_a^0 \wedge x_b^1 \wedge x_c^1)$ arbitrario.

Como consecuencia evidente del Teorema 3.1, un grafo canónico codifica el siguiente conjunto de *independencias instanciadas*[7]:

Corolario 3.1. Sea $G^k = (G, x^k)$ un grafo canónico. El grafo G^k codifica el supuesto de independencia $I(x_A^k \perp x_B^k \mid x_U^k)$ siempre que en G el conjunto de nodos A y el conjunto B estén separados por los nodos U, es decir, $\text{sep}_G(A, B; U)$.

Por ejemplo, el grafo canónico mostrado en la Figura 3-13 cumple $\text{sep}_G(a, b; c)$. Debido a que todos los nodos tienen un estado asignado en el contexto canónico $(x_a^0 \wedge x_b^1 \wedge x_c^1)$, entonces el grafo canónico $(G, x_a^0 \wedge x_b^1 \wedge x_c^1)$ codifica la independencia $I(x_a^0 \perp x_b^1 \mid x_c^1)$, la cual es una independencia instanciada.

De esta manera, en comparación a un modelo dividido, un ensamble canónico es representado como un conjunto de grafos canónicos, i.e., *un ensamble canónico no utiliza grafos divididos*. Como mencionamos en la Sección 3.5.2, gran parte de la complejidad de construir un modelo dividido surge al construir los árboles divididos de un grafo dividido (e.g., decidir cuál subgrafo asociar a un árbol dividido, decidir el nodo raíz del árbol, decidir cómo representar las hojas del árbol). En consecuencia, al construir un ensamble canónico, todas las complejidades involucradas al construir un grafo dividido son evitadas.

Principalmente, un conjunto de grafos canónicos evita los problemas de un grafo dividido por las siguientes razones:

I en un grafo canónico, todas las variables tienen un estado asignado, evitando el problema de decidir cuáles variables deben tener un estado asignado y cuáles no; y

[7]Las independencias instanciadas fueron formalmente definidas en la Definición 2.3.

II los grafos canónicos que representan un ensamble canónico no están relacionados jerárquicamente, facilitando la construcción de cada grafo canónico, debido a que no se encuentran relacionados entre sí.

Finalmente, como hemos comentado, la estructura de una distribución está formada por un estructura global, la cual codifica independencias condicionales, y un conjunto de estructuras locales, la cuales codifican independencias específicas del contexto. Sin embargo, un grafo canónico únicamente puede codificar independencias instanciadas, y, en consecuencia, un conjunto de grafos canónicos puede únicamente codificar un conjunto de independencias instanciadas. Como veremos en la próxima sección, cualquier independencia condicional o independencia específica del contexto puede ser representada a través de un conjunto de independencias instanciadas.

3.7. Representando estructuras con grafos canónicos

Como vimos en la sección anterior, un grafo canónico sólo puede codificar independencias instanciadas. En consecuencia, utilizando un conjunto de grafos canónicos, como representación de la estructura de una distribución, únicamente es posible codificar un conjunto de independencias instanciadas. Este hecho puede hacer surgir preguntas como ¿qué sucede si la estructura de una distribución presenta independencias que no son instanciadas? En dicho caso ¿un conjunto de grafos canónicos puede representar dicha estructura correctamente?

Como mencionamos en la Sección 2.1, cualquier independencia condicional e independencia específica del contexto puede ser expresada en términos de un conjunto de independencias instanciadas. De este modo, como se mostró en la Sección 2.1.1, la estructura de cualquier distribución puede ser definida como un conjunto de independencias instanciadas, ver la Ecuación (2.10). Debido a que la estructura de una distribución es un conjunto de independencias instanciadas, cualquier independencia condicional e independencia específica del contexto puede ser inferida a través de las Ecuaciones (2.8) y (2.9), respectivamente.

El siguiente ejemplo ilustra como el conjunto de independencias de una estructura puede ser expresado en términos de independencias instanciadas.

Ejemplo 3.15. En el Ejemplo 3.1 mostramos que la distribución $p(X)$ presenta la siguiente independencia:

$$\mathcal{I}_S = \{\ I(X_a \perp X_b \mid x_c^1)\ \}.$$

Utilizando la Ecuación (2.9), la independencia $I(X_a \perp X_b \mid x_c^1)$ puede ser expresada como un conjunto de independencias instanciadas. Así, el conjunto $\mathcal{I}_S$ puede ser reexpresado como un nuevo conjunto de independencias, denotado por $\mathcal{I}'_S$, el cual está formado por las siguientes independencias:

1. $I(x_a^0 \perp x_b^0 \mid x_c^1)$;
2. $I(x_a^1 \perp x_b^0 \mid x_c^1)$;
3. $I(x_a^0 \perp x_b^1 \mid x_c^1)$;
4. $I(x_a^1 \perp x_b^1 \mid x_c^1)$.

Según lo visto en Ejemplo 3.3, la estructura de $p(X)$ está formada por dos estructuras: una global y una local. Ambas estructuras también se encuentran expresadas en el nuevo conjunto $\mathcal{I}'_S$. La estructura global de $p(X)$ puede ser representada por un grafo no dirigido completo, es decir, $p(X)$ no codifica ninguna independencia condicional. Desde el conjunto $\mathcal{I}'_S$, ninguna independencia condicional puede ser inferida. Por ejemplo, según la Ecuación (2.8), la independencia $I(X_a \perp X_b \mid X_c)$ puede ser expresada como:

$$I(X_a \perp X_b \mid X_c) = I(X_a \perp X_b \mid x_c^0) \wedge I(X_a \perp X_b \mid x_c^1), \tag{3.9}$$

donde, por la Ecuación (2.9), cada independencia específica del contexto puede ser expresada como:

$$I(X_a \perp X_b \mid x_c^0) = I(x_a^0 \perp x_b^0 \mid x_c^0) \wedge I(x_a^1 \perp x_b^0 \mid x_c^0) \wedge I(x_a^0 \perp x_b^1 \mid x_c^0) \wedge I(x_a^1 \perp x_b^1 \mid x_c^0), \tag{3.10}$$

y

$$I(X_a \perp X_b \mid x_c^1) = I(x_a^0 \perp x_b^0 \mid x_c^1) \wedge I(x_a^1 \perp x_b^0 \mid x_c^1) \wedge I(x_a^0 \perp x_b^1 \mid x_c^1) \wedge I(x_a^1 \perp x_b^1 \mid x_c^1). \tag{3.11}$$

En base al conjunto $\mathcal{I}'_S$, la independencia $I(X_a \perp X_b \mid X_c)$ no se cumple debido a que ninguna de las independencias instanciadas mostradas en la Ecuación (3.10) está presente en $\mathcal{I}'_S$.

Por otro lado, la estructura local de $p(X)$ está asociada al contexto $X_c = 1$. Como se mostró en el Ejemplo 3.3, esta estructura codifica la independencia $I(X_a \perp X_b \mid x_c^1)$, y tal como muestra la Ecuación (3.11), dicha independencia se encuentra presente en el $\mathcal{I}'_S$.

En conclusión, la estructura global y la estructura local de $p(X)$ se encuentran presentes en el conjunto $\mathcal{I}'_S$, el cual, como mencionamos al comienzo, sólo contiene independencias instanciadas.

□

Por lo tanto, al utilizar un conjunto de grafos canónicos como representación, las independencias instanciadas de una distribución son codificadas. Este conjunto de independencias instanciadas está indirectamente codificando tanto la estructura global (independencias condicionales) como las estructuras locales (independencias específicas del contexto) de una distribución.

En esta sección presentaremos formalmente qué conjunto de independencias instanciadas es codificado por un conjunto de grafos canónicos. Con este fin, la sección está dividida en dos partes, las Secciones 3.7.1 y 3.7.2. En la Sección 3.7.1, mostraremos en detalle qué conjunto de independencias instanciadas puede ser codificado

por un grafo canónico. En base a esto, presentaremos el Teorema 3.3, el cual muestra formalmente cómo construir un grafo canónico para codificar independencias instanciadas presentes en una distribución de probabilidad.

Sin embargo, un grafo canónico está limitado en cuanto al conjunto de independencias que puede codificar, en términos simples, un grafo canónico únicamente puede codificar independencias instanciadas por un contexto canónico en particular. De esta manera, en la Sección 3.7.2, mostraremos cómo utilizar un conjunto de grafos canónicos para codificar un conjunto más amplio de independencias instanciadas. Como resultado, presentaremos el Corolario 3.2, el cual muestra formalmente cómo construir un conjunto de grafos canónicos para codificar la estructura de una distribución.

3.7.1. Independencias codificadas por un grafo canónico

Desde un punto de vista general, esta sección trata dos temas. El primer tema trata sobre qué conjunto de independencias instanciadas puede ser codificada por un grafo canónico. Más concretamente, mostraremos que un grafo canónico puede codificar tres conjuntos diferentes de independencias instanciadas: el *conjunto de independencias globales*, el *conjunto de independencias locales* y el *conjunto de independencias por parejas*.

Estos conjuntos se encuentran relacionados entre sí. Para distribuciones en general, el conjunto de independencias globales es estrictamente un superconjunto del conjunto de independencias locales, mientras que el conjunto de independencias locales es estrictamente un superconjunto del conjunto de independencias por parejas (Proposición 3.1). Sin embargo, como mostraremos en el Teorema 3.2, para distribuciones positivas, *estos tres conjuntos son equivalentes*.

En base a esta última equivalencia, el segundo tema de esta sección presenta el Teorema 3.3, el cual muestra cómo construir un grafo canónico para codificar independencias instanciadas presentes en una distribución.

Ofrecida esta visión general, profundicemos cada uno de los temas señalados.

Tal como vimos en la Sección 2.2, un grafo no dirigido puede ser asociado a diferentes conjuntos de independencias condicionales: las independencias globales, Defini-

ción 2.4, las independencias locales, Ecuación (2.14), y las independencias por parejas, Ecuación (2.15). A continuación mostraremos que un grafo canónico también puede ser asociado a estos mismos conjuntos de independencias. Sin embargo, a diferencia de un grafo no dirigido, los conjuntos de independencias asociados a un grafo canónico únicamente puede contener independencias instanciadas por un contexto canónico en particular.

Definición 3.11. Sea $G = (V, E)$ un grafo no dirigido. Sea $G^k = (G, x^k)$ un grafo canónico instanciado por un contexto canónico $x^k \in val(V)$. Definimos el conjunto de *independencias globales* asociadas a G^k, denotado por $\mathcal{I}(G^k)$, como

$$\mathcal{I}(G^k) = \{\ I(x_A^k \perp x_B^k \mid x_U^k)\ :\ \mathrm{sep}_G(A, B; U)\ \}\,. \tag{3.12}$$

Definición 3.12. Sea $G = (V, E)$ un grafo no dirigido. Sea $G^k = (G, x^k)$ un grafo canónico instanciado por un contexto canónico $x^k \in val(V)$. Definimos el conjunto de *independencias locales* asociadas a G^k, denotado por $\mathcal{I}^\ell(G^k)$, como

$$\mathcal{I}^\ell(G^k) = \{\ I(x_a^k \perp x_{V\setminus S}^k \mid x_S^k)\colon\ \mathrm{sep}_G(a, V \setminus S; S) \text{ para todo } a \in V\ \}\,. \tag{3.13}$$

Definición 3.13. Sea $G = (V, E)$ un grafo no dirigido. Sea $G^k = (G, x^k)$ un grafo canónico instanciado por un contexto canónico $x^k \in val(V)$. Definimos el conjunto de *independencias por parejas* asociadas a G^k, denotado por $\mathcal{I}^p(G^k)$, como

$$\mathcal{I}^p(G^k) = \{\ I(x_a^k \perp x_b^k \mid x_{V\setminus\{a,b\}}^k)\colon\ \mathrm{sep}_G(a, b; V \setminus \{\ a, b\ \}) \text{ para todo } a, b \in V\ \}\,. \tag{3.14}$$

Esta última definición afirma que si la arista (a, b) no está en G^k, entonces x_a^k y x_b^k deben ser independientes dado todos los restantes nodos, $x_{V\setminus\{a,b\}}^k$, *independientemente de cuales otras aristas están presentes en el grafo canónico.*

Tal como muestra el siguiente resultado, estos conjuntos de independencias asociados a un grafo canónico se encuentran relacionados entre sí. Concretamente, las independencias locales y las independencias por parejas son subconjuntos de las independencias globales.

Proposición 3.1. Sea G^k un grafo canónico y sea $p(X)$ cualquier distribución de probabilidad. Si G^k es un I-map de $p(X)$, entonces la distribución $p(X)$ cumple con

$$\mathcal{I}(G^k) \implies \mathcal{I}^\ell(G^k) \implies \mathcal{I}^p(G^k).$$

Demostración. Como mencionamos en el capítulo anterior, el predicado $I(\cdot \perp \cdot \mid \cdot)$ satisface las propiedades de Markov (ver Apéndice A). Estas propiedades son axiomas de un grafo, es decir, cualquier grafo satisface dicha propiedades. Debido a que un grafo canónico es simplemente un grafo, donde sus nodos están etiquetados con estados de variables, un grafo canónico también satisface las propiedades de Markov.

La primera implicación, $\mathcal{I}(G^k) \implies \mathcal{I}^\ell(G^k)$, sigue del hecho de que cualquier independencia local es un caso particular de independencia global. En otras palabras, podemos definir una independencia global $I(x_A^k \perp x_B^k \mid x_U^k)$ tal que $A = \{\, a \,\}$, $B = V \setminus S$, y $U = S$.

Para la segunda implicación, $\mathcal{I}^\ell(G^k) \implies \mathcal{I}^p(G^k)$, para cualquier independencia $I(x_a^k \perp x_{V\setminus S}^k \mid x_S^k)$ en $\mathcal{I}^\ell(G^k)$, tenemos que cualquier nodo $b \in V \setminus S$ es no adyacente al nodo a. Sea W el subconjunto $W = V \setminus S \setminus \{\, b \,\}$ en V. Entonces $I(x_a^k \perp x_{V\setminus S}^k \mid x_S^k)$ puede reescribirse como $I(x_a^k \perp x_{W\cup\{\, b\,\}}^k \mid x_S^k)$. Aplicando unión débil, tenemos que

$$I(x_a^k \perp x_b^k \mid x_{S\cup W}^k).$$

Reemplazando W, tenemos

$$I(x_a^k \perp x_b^k \mid x_{S\cup V\setminus S\setminus\{\, b\,\}}^k) = I(x_a^k \perp x_b^k \mid x_{V\setminus\{\, a,b\,\}}^k).$$

□

Interesántemente, cuando una distribución es positiva, *los tres conjuntos de in-*

dependencias asociados a un grafo canónico son equivalentes, tal como muestra el siguiente resultado.

Teorema 3.2. Sea G^k un grafo canónico arbitrario. Sea $p(X)$ una distribución positiva. Si G^k es I-map con respecto a $p(X)$, entonces la distribución $p(X)$ satisface

$$\mathcal{I}(G^k) \iff \mathcal{I}^\ell(G^k) \iff \mathcal{I}^p(G^k). \tag{3.15}$$

Demostración. 1. La primera dirección del teorema fue demostrada en la Proposición 3.1.

2. A diferencia de la Proposición 3.1, este teorema asume una distribución positiva, en consecuencia, las independencias en $p(X)$ satisface las propiedades de Markov: unión fuerte y transitividad. Para demostrar $\mathcal{I}^p(G^k) \implies \mathcal{I}(G^k)$, usaremos inducción hacia atrás en el número de elementos de un conjunto $S \subseteq V$, donde X_S son las variables que condicionan cualquier independencia instanciada por x^k en $p(X)$ (e.g., $I(x^k_\cdot \perp x^k_\cdot \mid x^k_S)$).

   - **Caso base** sucede cuando asumimos que cualquier independencia instanciada por x^k en $p(X)$ satisface $|S| = k = |V| - 2$. En dicho caso, cualquier independencia instanciada tiene la forma $I(x^k_a \perp x^k_b \mid x^k_{V\setminus\{a,b\}})$, entonces $\mathcal{I}(G^k) = \mathcal{I}^\ell(G^k) = \mathcal{I}^p(G^k)$.
   - **Paso de inducción**, por la hipótesis de inducción, para todo conjunto $S' \subseteq V$, cualquier independencia $I(x^k_\cdot \perp x^k_\cdot \mid x^k_{S'})$ con $|S'| = k$ se cumple en $p(X)$. Sea S cualquier subconjunto en V tal que $|S| = k - 1$. Cualquier independencia $I(x^k_A \perp x^k_B \mid x^k_S)$ en $p(X)$ puede implicar dos casos: $A \cup B \cup S = V$ y $A \cup B \cup S \subset V$.
     - Para el primer caso, tenemos que al menos uno de los conjuntos, A o B, tienen más de un elemento. Sin pérdida de generalidad, asumamos que A tiene más de un elemento y que $a \in A$. En tal caso, $p(X)$ satisface las siguientes independencias

$$I(x^k_{A\setminus\{\,a\,\}} \perp x^k_B \mid x^k_{S\cup\{\,a\,\}}) \wedge I(x^k_a \perp x^k_B \mid x^k_{S\cup A\setminus\{\,a\,\}}),$$

debido a la hipótesis de inducción, i.e., $|S \cup \{\ a\ \}| = |S \cup A \setminus \{\ a\ \}| > k-1$. Por transitividad, si $p(X)$ satisface las independencias anteriores, entonces también debe satisfacer la siguiente independencia:

$$I(x^k_A \perp x^k_B \mid x^k_S).$$

- Para el segundo caso, tenemos que $V \setminus (A \cup B \cup S) = U \neq \emptyset$, es decir, existe al menos un elemento $u \in U$ que no pertenece a $A \cup B \cup S$. Para cualquier elemento $u \in U$, $p(X)$ cumple la siguiente independencia

  $$I(x^k_A \perp x^k_B \mid x^k_{S\cup\{\,u\,\}}),$$

  y, alguna de las siguientes

  $$I(x^k_A \perp x^k_u \mid x^k_{S\cup B}) \vee I(x^k_u \perp x^k_B \mid x^k_{S\cup A}),$$

  o ambas. Las independencias anteriores se cumplen por la hipótesis de inducción, es decir, el tamaño de cualquier conjunto condicional es mayor a $k-1$. De esta manera, los siguientes conjuntos de independencias se cumplen en $p(X)$:

  $$I(x^k_A \perp x^k_B \mid x^k_{S\cup\{\,u\,\}}) \wedge I(x^k_A \perp x^k_u \mid x^k_{S\cup B}),$$

  o

  $$I(x^k_A \perp x^k_B \mid x^k_{S\cup\{\,u\,\}}) \wedge I(x^k_u \perp x^k_B \mid x^k_{S\cup A}).$$

  Sin pérdida de generalidad, tomemos el último, por intersection, la independencia $I(x^k_{A\cup\{\,u\,\}} \perp x^k_B \mid x^k_S)$ debe cumplirse en $p(X)$, y, por descomposición, $I(x^k_A \perp x^k_B \mid x^k_S)$ también debe cumplirse en $p(X)$.

□

A partir de este último teorema, una interesante observación puede ser extraída: cualquier independencia instanciada puede ser expresada como un conjunto de independencias por parejas. En consecuencia, en una distribución positiva, cualquier conjunto de independencias instanciadas, independencias condicionales o independencias específicas del contexto pueden ser uniformemente descriptas a través de un conjunto de independencias por parejas.

Por la Definición 3.13, cualquier independencia por parejas tiene una característica destacable, tal independencia involucra *todas las variables del dominio*. De este modo, cualquier independencia por parejas se encuentra asociada a un contexto canónico. Por ejemplo, dado un contexto canónico arbitrario, e.g. $x^k \in val(V)$, una distribución puede asociar una o más independencias por parejas a dicho contexto. El siguiente teorema muestra cómo codificar las independencias por parejas asociadas a un contexto canónico particular por medio de un grafo canónico.

Teorema 3.3. Sea $p(X)$ una distribución positiva. Sea x^k un contexto canónico en $val(V)$. Sea $\mathcal{I}^k$ todas las independencias instanciadas por x^k en $p(X)$. Si $G^k = (G, x^k)$ es un grafo canónico completo desde el cual

$$(a, b) \notin E \iff I(x_a^k \perp x_b^k \mid x_{V\setminus\{a,b\}}^k) \text{ se cumple en } p(X),$$

entonces $\mathcal{I}^k = \mathcal{I}(G^k)$.

Demostración. Por el Teorema 3.2, tenemos que $\mathcal{I}^p(G^k) \iff \mathcal{I}(G^k)$. Por la Definición 3.11, el conjunto $\mathcal{I}(G^k)$ es precisamente el conjunto de todas las independencias instanciadas por x^k. □

Finalmente, el siguiente ejemplo ilustra cómo utilizar el Teorema 3.3 para construir un grafo canónico.

Ejemplo 3.16. Como vimos en el Ejemplo 3.15, la distribución $p(X)$ presenta el conjunto $\mathcal{I}'_S$ de independencias instanciadas. Para esta distribución, podemos construir

un grafo canónico para representar un subconjunto de sus independencias instanciadas.

Por ejemplo, tomemos un contexto canónico arbitrario, $(x_a^0 \wedge x_b^0 \wedge x_c^1) \in val(V)$, el cual denotaremos como x^0. La distribución $p(X)$ presenta una única independencia instanciada por x^0, a saber, $I(x_a^0 \perp x_b^0 \mid x_c^1)$.

Según el Teorema 3.3, un grafo canónico $G(x^0)$ pueden ser usado para codificar las independencias instanciadas por x^0. Para esto, partiendo de un grafo canónico completo, removemos la arista (a, b) debido a que $I(x_a^0 \perp x_b^0 \mid x_c^1)$ se cumple en $p(X)$. El resto de las aristas del grafo $G(x^0)$ no pueden ser removidas debido a que el Teorema 3.3 no se cumple. El grafo resultante es mostrado en la Figura 3-14.

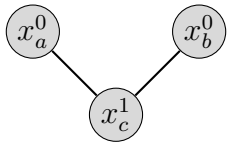


Figura 3-14: Grafo canónico que codifican la independencia $I(x_a^0 \perp x_b^0 \mid x_c^1)$.

Utilizando separabilidad entre nodos sobre el grafo canónico $G(x^0) = (G, x^0)$, tenemos que $\text{sep}_G(a, b; c)$, lo cual implica la independencia $I(x_a^0 \perp x_b^0 \mid x_c^1)$. De esta manera, $G(x^0)$ codifica todas las independencias instanciadas por x^0 en $p(X)$.

□

3.7.2. Independencias codificadas por un conjunto de grafos canónicos

Como vimos anteriormente, un grafo canónico G^k obtenido por el Teorema 3.3 solamente puede codificar independencias instanciadas por el contexto x^k. Sin embargo, una distribución de probabilidad puede presentar independencias instanciadas por distintos contextos canónicos. Por ejemplo, según la Ecuación (2.8), una independencia condicional equivale a una conjunción de independencias instanciadas sobre diferentes contextos. Así, si una distribución presenta una independencia condicional, un único grafo canónico no puede codificar dicha independencia.

Para hacer frente a dicha situación, podemos utilizar más de un grafo canónico. De esta manera, a través de un conjunto de grafos canónicos, denotado por $\mathcal{G}$, cada grafo canónico G^k en el conjunto $\mathcal{G}$ se encuentra asociado a un contexto diferente, codificando independencias instanciadas por diferentes contextos.

Para ver esto más formalmente, generalizaremos las Definiciones 3.11, 3.12 y 3.13 para ser aplicadas sobre un conjunto de grafos canónicos. Dado un conjunto $\mathcal{G}$ de grafos canónicos, las siguientes ecuaciones muestran los diferentes conjuntos de independencias asociados al conjunto $\mathcal{G}$:

$$\mathcal{I}(\mathcal{G}) = \bigcup_{G^k \in \mathcal{G}} \mathcal{I}(G^k), \tag{3.16}$$

$$\mathcal{I}^{\ell}(\mathcal{G}) = \bigcup_{G^k \in \mathcal{G}} \mathcal{I}^{\ell}(G^k), \tag{3.17}$$

$$\mathcal{I}^{p}(\mathcal{G}) = \bigcup_{G^k \in \mathcal{G}} \mathcal{I}^{p}(G^k). \tag{3.18}$$

En las ecuaciones previas, la generalización consiste en observar que las Definiciones 3.11, 3.12 y 3.13 son casos particulares para un conjunto $\mathcal{G}$ con un único grafo canónico.

Ahora, tal como muestra el siguiente resultado, un conjunto $\mathcal{G}$ de grafos canónicos puede ser construido utilizando las independencias instanciadas por diferentes contextos presentes en $p(X)$.

Corolario 3.2. Sea $p(X)$ una distribución positiva. Sea $\mathcal{X}$ un conjunto de contextos canónicos en $val(V)$. Sea $\mathcal{I}^{\mathcal{X}}$ todas las independencias instanciadas por cada $x^k \in \mathcal{X}$ en $p(X)$. Si el conjunto $\mathcal{G} = \{ G^k \colon x^k \in \mathcal{X} \}$ es definido tal que, para todo grafo canónico $G^k \in \mathcal{G}$, G^k es construido según el Teorema 3.3, entonces $\mathcal{I}^{\mathcal{X}} = \mathcal{I}(\mathcal{G})$.

Demostración. Este Corolario sigue directamente del Teorema 3.3. Básicamente, para cada contexto x^k en el conjunto $\mathcal{X}$, podemos definir un grafo completo canónico G^k en el cual codificar el conjunto $\mathcal{I}^k = \mathcal{I}(G^k)$ de independencias instanciadas presentes en $p(X)$ utilizando el Teorema 3.3. Una vez que cada contexto $x^k \in \mathcal{X}$ tiene un grafo

canónico G^k obtenido por el Teorema 3.3, entonces tenemos

$$\mathcal{I}^{\mathcal{X}} = \bigcup_{x^k \in \mathcal{X}} \mathcal{I}^k = \bigcup_{x^k \in \mathcal{X}} \mathcal{I}(G^k) = \mathcal{I}(\mathcal{G}).$$

□

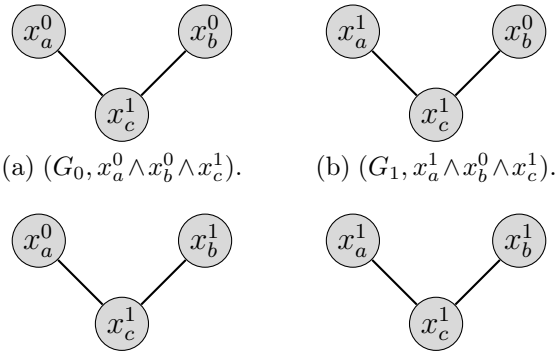
En el siguiente ejemplo, mostraremos cómo utilizar el conjunto $\mathcal{X}$ inducido en el ejemplo anterior para construir un conjunto $\mathcal{G}$ de grafos canónicos que codifiquen todas las independencias instanciadas de $p(X)$.

Ejemplo 3.17. En este ejemplo mostraremos como construir un conjunto $\mathcal{G}$ con el Corolario 3.2 desde un conjunto $\mathcal{X}$. Luego mostraremos que el conjunto $\mathcal{I}(\mathcal{G})$ codifica todas las independencias instanciadas en $p(X)$ del Ejemplo 3.1.

Primero, utilizaremos el siguiente conjunto de contextos canónicos:

$$\mathcal{X} = \{\ (x_a^0 \wedge x_b^0 \wedge x_c^1), (x_a^1 \wedge x_b^0 \wedge x_c^1), (x_a^0 \wedge x_b^1 \wedge x_c^1), (x_a^1 \wedge x_b^1 \wedge x_c^1)\ \}\,.$$

Utilizando cada contexto $x^k \in \mathcal{X}$, un grafo $G(x^k)$ puede ser construido con el Teorema 3.3. El conjunto $\mathcal{G}$ de grafos canónicos obtenido por este proceso es mostrado en la Figura 3-15.



(a) $(G_0, x_a^0 \wedge x_b^0 \wedge x_c^1)$. (b) $(G_1, x_a^1 \wedge x_b^0 \wedge x_c^1)$.

(c) $(G_2, x_a^0 \wedge x_b^1 \wedge x_c^1)$. (d) $(G_3, x_a^1 \wedge x_b^1 \wedge x_c^1)$.

Figura 3-15: Conjunto de grafos canónicos que codifican las independencias de $p(X)$.

Para verificar que estos grafos canónicos codifican todas las independencias de $p(X)$, utilizamos separación entre nodos. Los grafos canónico, de izquierda a derecha, codifican las siguientes independencias: $I(x_a^0 \perp x_b^0 \mid x_c^1)$, $I(x_a^1 \perp x_b^0 \mid x_c^1)$, $I(x_a^0 \perp x_b^1 \mid x_c^1)$,

y $I(x_a^1 \perp x_b^1 \mid x_c^1)$. Precisamente estas son las independencias pertenecientes al conjunto $\mathcal{I}'_S$ (Ejemplo 3.15), las independencias instanciadas de $p(X)$.

□

El Corolario 3.2 plantea una última pregunta fundamental que resta por responder con respecto a la construcción de un conjunto de grafos canónicos:

¿Cuáles contextos canónicos deben pertenecer al conjunto $\mathcal{X}$?

La importancia de esta pregunta radica en el hecho de que, al construir el conjunto $\mathcal{G}$, estamos buscando codificar el mayor número de independencias instanciadas presentes en una distribución en particular. Así, una selección no adecuada de contextos canónicos podría resultar en un conjunto de grafos canónicos en el cual algunas independencias instanciadas no sean codificadas.

Ilustremos esto último a través de un ejemplo. En el Ejemplo 3.15, cualquier grafo canónico obtenido por el Teorema 3.3 cuyo contexto canónico involucre el estado $X_c = 0$, resulta en un grafo canónico completo, el cual no codifica ninguna de las independencias instanciadas en $p(X)$. En consecuencia, si se define un conjunto $\mathcal{X}$ sólo formado por contextos canónicos con $X_c = 0$, el conjunto de grafos canónicos obtenidos por el Corolario 3.2 no codificarían ninguna de las independencias de $p(X)$.

Sin embargo, responder esta pregunta no es sencillo y, bajo cierta perspectiva, se encuentra más allá de los objetivos de este trabajo. A pesar de esto, como veremos en la Sección 4.3.1, propondremos una forma práctica para definir el conjunto $\mathcal{X}$. En términos sencillos, el conjunto $\mathcal{X}$ puede ser definido desde el conjunto D, lo cual garantiza que $|\mathcal{X}| \leq |D|$. Aunque esta forma de definir el conjunto $\mathcal{X}$ no garantice que los grafos canónicos construidos desde tal conjunto capturen todas las independencias de una distribución, nuestros resultados experimentales (presentados en el Capítulo 5) muestran claramente que el conjunto de grafos canónicos construido con este conjunto de contextos logra representar estructuras más exactas que aquellas obtenidas por otros algoritmos de aprendizaje de estructuras.

Finalmente, para cerrar este capítulo, mostraremos de dónde proviene la dificultad de definir el conjunto $\mathcal{X}$ de contextos. Hablando de forma aproximada, la dificultad

radica en la presencia de "redundancia" en la estructura de una distribución. En lo que sigue, introduciremos más formalmente el concepto de "redundancia" en la estructura de una distribución.

Antes que nada, asumiremos una distribución de probabilidad positiva y retomaremos la definición de estructura presentada en la Sección 2.1.1. Recordando, la estructura de una distribución puede ser definida de la siguiente manera. Sea $\mathcal{I}$ el conjunto de todas las posibles independencias instanciadas para un dominio, la estructura de una distribución consiste de una función indicador que asigna, a cada posible independencia instanciada en $\mathcal{I}$, un valor de verdad. En base a esto, la estructura de una distribución puede ser definida como el conjunto de independencias instanciadas en $\mathcal{I}$ cuyo valor de verdad es "verdadero".

Ahora, según se mostró en la Ecuación (2.10), el conjunto de todas las posibles independencias instanciadas de un dominio es definido como sigue:

$$\mathcal{I} = \bigcup_{A,B,U \subseteq V} \bigcup_{x \in val(A \cup B \cup U)} I(x_A \perp x_B \mid x_U). \tag{3.19}$$

Un problema de este conjunto de independencia instanciadas es que existe *redundancia*. Para ilustrar a qué nos referimos con redundancia, tomemos una independencia instanciada arbitraria en $\mathcal{I}$, digamos $I(x_A \perp x_B \mid x_U)$, donde, por definición, A, B y U son conjuntos disjuntos. Esta independencia se cumple independientemente del estado que tomen las restantes variables X_W, con $W = V \setminus (A \cup B \cup U)$. En otras palabras, asumiendo variables binarias, la independencia $I(x_A \perp x_B \mid x_U)$ puede ser asociada a $2^{|W|}$ contextos canónicos diferentes, es decir, todo contexto canónico $x^k \in val(V)$ tal que $x^k_A = x_A \wedge x^k_B = x_B \wedge x^k_U = x_U$.

Desde otra perspectiva, para cada uno de estos contextos canónicos, puede ser construido un grafo canónico, según el Teorema 3.3, de modo tal que su topología codifique la independencia $I(x_A \perp x_B \mid x_U)$. En consecuencia, existen $2^{|W|}$ diferentes grafos canónicos que pueden codificar la misma independencia. Precisamente a este último hecho lo llamamos redundancia.

La presencia de redundancia en el conjunto $\mathcal{I}$ de independencias es lo que dificulta la definición del conjunto $\mathcal{X}$ de contextos. Por ejemplo, dada una distribución $p(X)$ positiva arbitraria, si $\mathcal{I}_{p(X)}$ es el subconjunto de las independencias en $\mathcal{I}$ que se cumple en $p(X)$, entonces puede existir más de un conjunto de contextos a partir del cual se puede construir un conjunto de grafos canónicos tal que codifique el conjunto $\mathcal{I}_{p(X)}$ de independencias.

3.8. Resumen

En este capítulo mostramos que la estructura de una distribución puede ser dividida en dos tipos de estructuras. Por un lado, una estructura global la cual sólo codifica independencias condicional. Por otro lado, un conjunto de estructuras locales, las cuales sólo codifican independencias específicas del contexto.

Al utilizar un grafo para representar la estructura de una distribución, éste sólo puede codificar la estructura global, pero no así sus estructuras locales. En contraste, al utilizar un conjunto de características, éstas pueden representar ambos tipos de estructuras. Sin embargo, un conjunto de características codifica una estructura de forma no gráfica, resultando en una representación con una baja interpretabilidad en comparación a un grafo.

Los modelos CSI son una representación alternativa de una red de Markov, cuya estructura, denominada clase generadora, combina la interpretabilidad de un grafo y la flexibilidad de un conjunto de características. En términos sencillos, una clase generadora puede ser representada gráficamente por un conjunto de grafos instanciados. Un grafo instanciado es un grafo asociado a un contexto[8]. Como resultado, una clase generadora puede codificar tanto la estructura global como las estructuras locales de una distribución.

Una misma clase generadora puede ser igualmente representada por diferentes conjuntos de grafos instanciados. Sin embargo, ciertos conjuntos de grafos instanciados pueden codificar más independencias que otros. Presentamos dos maneras de

[8]Estado tomado por un subconjunto de las variables del dominio.

representar una clase generadora a través de un conjunto de grafos instanciados. Una son los modelos divididos y la otra, nuestra propuesta, los modelos canónicos.

Los modelos divididos representan una clase generadora a través de una estructura de datos llamada grafo dividido. Un grafo dividido está formado por una lista de árboles divididos, donde un árbol dividido es un árbol cuyos nodos tienen asociados grafos instanciados u otros grafos divididos.

Los modelos canónicos representan una clase generadora a través de un conjunto especial de grafos instanciados, llamados grafos canónicos. Un grafo canónico es un grafo instanciado asociado a un contexto canónico (todas las variables del dominio tienen un estado asociado).

Desde el punto de vista algorítmico, mostramos que la construcción de un grafo dividido desde un conjunto de datos presenta grandes dificultades. Por ejemplo, a través de una revisión bibliográfica, no encontramos un algoritmo de aprendizaje de grafos divididos. En contraste, la construcción de un conjunto de grafos canónicos es mucho más sencillo en comparación a un grafo dividido.

Para construir un conjunto de grafos canónicos, se necesita definir un conjunto de contextos canónicos y luego, para cada contexto canónico, se construye un grafo canónico. Mostramos que la construcción de un grafo canónico involucra determinar la ausencia de las aristas de un grafo completo.

Capítulo 4

Aprendizaje de grafos canónicos

Todo debería ser hecho tan simple como sea posible, pero no más simple.

Albert Einstein.

Como mostramos en el capítulo anterior, una red de Markov puede ser alternativamente definida como un modelo canónico. Como resultado, la estructura de dicha red de Markov puede ser representada como un conjunto $\mathcal{G}$ de grafos canónicos. Un grafo canónico $G^k \in \mathcal{G}$ consiste en un grafo no dirigido G cuyo conjunto de nodos está asociado a un contexto canónico $x^k \in val(V)$. Esto permite que un grafo canónico, a diferencia de un grafo común, codifique independencias instanciadas por el contexto x^k. Como vimos en la Sección 2.1, toda independencia condicional y específica del contexto puede ser expresada como un conjunto de independencias instanciadas. Como resultado, un conjunto de grafos canónicos puede codificar independencias condicionales e independencias específicas del contexto, mientras que un grafo común únicamente puede codificar independencias condicionales.

De esta manera, al usar un conjunto de grafos canónicos como representación de una estructura, podemos representar un rango más amplio de estructuras, lo cual permite codificar distribuciones de forma más precisa. En este capítulo proponemos un enfoque para aprender un conjunto $\mathcal{G}$ de grafos canónicos desde un conjunto de datos. La característica principal de este enfoque es que los grafos canónicos son aprendidos utilizando algoritmos de aprendizaje de estructura bien conocidos, los algoritmos

basados en restricciones. Más concretamente, este enfoque se basa esencialmente en construir cada grafo canónico $G^k \in \mathcal{G}$ utilizando cualquier algoritmo basado en restricciones.

Como discutimos en el Capítulo 2, los algoritmos basados en restricciones fueron originalmente diseñados para aprender un grafo no canónico, es decir, un grafo que codifica únicamente independencias condicionales. Sin embargo, como mostraremos más adelante, una de las contribuciones de este trabajo consiste en mostrar que un algoritmo basado en restricciones puede también ser utilizado para construir un grafo canónico. Para lograr esto, primero hay que notar que todo algoritmo basado en restricciones se caracteriza por construir un grafo utilizando una prueba estadística de independencia. Utilizando una prueba estadística, un algoritmo basado en restricciones determina el valor de verdad de cualquier supuesto de independencia condicional. Como vimos en la Sección 2.4.1, estos valores de verdad indican al algoritmo cuáles aristas remover o agregar durante la construcción de un grafo. Como resultado, la topología del grafo construido por un algoritmo basado en restricciones codifica los valores de verdad de un conjunto de supuestos de independencia condicional.

Como veremos más adelante, para que un algoritmo basado en restricciones construya un grafo canónico G^k, únicamente hay que modificar su prueba estadística, tal que los valores de verdad retornados por dicha prueba correspondan a supuestos de independencia instanciados por el contexto x^k. Como consecuencia, las aristas removidas o agregadas en base a estos valores de verdad, resultarán en un grafo cuya topología codifique supuestos de independencia instanciados, en vez de supuestos de independencia condicional. Quizás el punto más significativo de modificar la prueba estadística es que dicha modificación no tiene ningún impacto sobre la lógica de un algoritmo basado en restricciones. Esto se debe a que la prueba de independencia es utilizada como caja negra por cualquier algoritmo basado en restricciones. Por lo tanto, una vez modificada una prueba estadística, esta misma prueba puede ser utilizada por cualquier algoritmo basado en restricciones.

Para mostrar como utilizar nuestro enfoque en la práctica, utilizaremos dos al-

goritmos prototípicos del enfoque basado en restricciones: los algoritmos PC y GS[1], introducidos en las Secciones 2.4.3 y 2.4.4, respectivamente. En base a estos algoritmos, definiremos dos instancias de nuestro enfoque: el algoritmo CSPC (acrónimo de las siglas en inglés, *Context-Specific Peter and Clark algorithm*) y el algoritmo CSGS (acrónimo de *Context-Specific Grow-Shrink algorithm*). Más específicamente, el algoritmo CSPC utiliza el algoritmo PC para construir cada uno de los grafos canónicos en un conjunto de grafos canónicos, mientras que el algoritmo CSGS utiliza el algoritmo GS para dicha tarea. Adicionalmente, mostraremos que tanto el algoritmo CSPC como el algoritmo CSGS pueden ser ejecutados en forma paralela, es decir, la construcción de un conjunto de grafos canónicos puede tomar ventaja de una arquitectura con múltiples computadoras.

Una desventaja de utilizar una representación novedosa de la estructura, como lo es un conjunto de grafos canónicos, es que dicha representación no puede ser utilizada con paquetes de software bien conocidos, los cuales son usualmente utilizados para estimación de parámetros e inferencia en redes de Markov. Más específicamente, la razón es que dichos paquetes no están diseñados para manipular una estructura representada como un conjunto de grafos canónicos.

Para evitar este problema, y permitir la utilización de paquetes de software bien conocidos con las estructuras aprendidas por nuestro enfoque, utilizamos la siguiente observación: todo conjunto $\mathcal{G}$ de grafos canonicos tiene asociado un conjunto de características equivalente, en el sentido de que dicho conjunto de características codifica exactamente las mismas independencias codificadas por el conjunto $\mathcal{G}$.

En base a esto, presentaremos un método para generar tal conjunto de características desde un conjunto de grafos canónicos. El conjunto de características generadas por dicho método garantiza codificar las independencias codificadas por cada uno de los grafos canónicos en un conjunto de grafos canónicos.

Como resultado, generando características por el método propuesto, las estruc-

[1]Como vimos en el Capítulo 2, el algoritmo GS fue originalmente diseñado para aprender la manta de Markov de una variable. Debido a que, en un grafo, la manta de Markov de una variable se corresponde con las adyacencias de un nodo, entonces obteniendo las mantas de cada variable, se puede construir un grafo.

turas aprendidas por nuestro enfoque pueden ser utilizadas con cualquier paquete de software diseñado para manipular características. Sin embargo, como veremos en profundidad más adelante, inducir un conjunto de características tiene asociado un costo, el proceso de inducción de características puede incorporar ciertos errores en la estructura, es decir, el conjunto de características generadas puede no ser exactamente equivalente al conjunto de grafos canónicos.

El resto del presente capítulo está estructurado de la siguiente manera. La Sección 4.1 describe cómo utilizar un algoritmo basado en restricciones para construir un grafo canónico. La Sección 4.2 presenta cómo utilizar una prueba estadística para independencias condicionales para determinar el valor de verdad de independencias instanciadas por un contexto. La Sección 4.3 introduce formalmente nuestro enfoque para construir un conjunto de grafos canónicos, junto a dos instanciaciones concretas del enfoque: los algoritmos CSPC y CSGS. Dentro de esta última sección, la Sección 4.3.3 mostrará como dichos algoritmos pueden ejecutarse de forma paralela. La Sección 4.4 presenta un análisis de la complejidad temporal de ambas instanciaciones. Por último, la Sección 4.5 presenta un método para generar un conjunto de características desde un conjunto de grafos canónicos.

4.1. Construcción de un grafo canónico

En esta sección discutiremos cómo utilizar un algoritmo basado en restricciones para construir un grafo canónico. Para facilitar la discusión, procederemos en dos etapas. Primero, formularemos en forma abstracta el problema de construcción de un grafo canónico. Segundo, mostraremos que un algoritmo basado en restricciones puede ser utilizado como una solución a dicho problema.

Antes que nada, es bueno recordar que, a diferencia de un grafo no dirigido G, un grafo canónico $G^k = (G, x^k)$ codifica únicamente supuestos de independencia instanciados por el contexto canónico x^k, en vez de supuesto de independencia condicional, como es en el caso de un grafo G. Como vimos en el capítulo anterior, precisamente en la Definición 3.11, dado un grafo canónico G^k, definimos como $\mathcal{I}(G^k)$ al conjunto

de todos los supuestos de independencia codificados por G^k. Es decir, sean A, B y C subconjuntos disjuntos arbitrarios en V. Si G^k cumple $\text{sep}_G(A, B; C)$, entonces G^k codifica el supuesto de independencia instanciado $I(x_A^k \perp x_B^k \mid x_C^k)$, es decir,

$$I(x_A^k \perp x_B^k \mid x_C^k) \in \mathcal{I}(G^k) \iff \text{sep}_G(A, B; C).$$

De esta manera, al construir un grafo G^k de una distribución $p(X)$, estamos interesados en obtener un grafo G^k cuya topología codifique independencias instanciadas por el contexto x^k presentes en la distribución $p(X)$. Por otro lado, por qué no, también estamos interesados en que G^k codifique el mayor número posible de independencias instanciadas presentes en $p(X)$. Para formalizar esto, decimos que un grafo G^k es una solución al problema mencionado anteriormente, si G^k cumple las siguientes dos propiedades:

1. *El grafo G^k debe ser un I-map de $p(X)$.* Decimos que G^k es un I-map de $p(X)$, si para todo $I(x_A^k \perp x_B^k \mid x_C^k) \in \mathcal{I}(G^k)$ se satisface

$$I(x_A^k \perp x_B^k \mid x_C^k) \iff p(x_A^k | x_B^k, x_C^k) = p(x_A^k | x_C^k). \tag{4.1}$$

2. *El grafo G^k debe ser un I-map mínimo de $p(X)$.* Decimos que G^k es un I-map mínimo de $p(X)$, si al remover cualquiera de sus aristas, el grafo G^k deja de ser un I-map de $p(X)$.

Si G^k cumple la primera propiedad, entonces todo supuesto en $\mathcal{I}(G^k)$ se cumple en $p(X)$, es decir, el grafo no codifica ningún supuesto de independencias falso, o técnicamente hablando, el grafo no presenta errores del tipo I. Por otro lado, si G^k cumple la segunda propiedad, entonces G^k codifica el máximo número de independencias instanciadas por x^k, es decir, el grafo G^k no presenta "aristas espurias", las cuales representan dependencias que no se cumplen en $p(X)$, los así llamados errores del tipo II.

En base a las dos propiedades descriptas, estamos en condiciones de proponer un método sencillo para construir un grafo solución G^k para una distribución $p(X)$. Este

método es formalmente presentado en el Algoritmo 6.

Algoritmo 6 Método para construir un grafo solución G^k

```
procedure ConstruirGrafoSolucion(p(X))
    G^k ← proponer un grafo inicial
    if G^k no es un I-map de p(X) then
        G^k ← ObtenerGrafoImap(G^k, p(X))
    if G^k no es un I-map mínimo de p(X) then
        G^k ← ObtenerGrafoMinimoImap(G^k, p(X))
return G^k
```

El método ConstruirGrafoSolucion construye un grafo solución G^k en base a una distribución $p(X)$ suministrada como parámetro de entrada. Para construir el grafo G^k el método ConstruirGrafoSolucion consiste en tres pasos.

El primero paso propone un grafo G^k inicial, es decir, se define un conjunto inicial de aristas en G^k.

El segundo paso consiste en utilizar la Ecuación (4.1) para verificar que todo supuesto de independencia codificado por G^k se cumple en $p(X)$. Si G^k no es un I-map, entonces existe al menos un supuesto que no cumple la Ecuación (4.1). En tal caso, el grafo G^k puede ser modificado con el objetivo de volverlo un I-map de $p(X)$, i.e., removiendo el supuesto que no cumple con la Ecuación 4.1. Precisamente, este es el objetivo del método llamado ObtenerGrafoImap, el cual modifica G^k para que los supuestos codificados por G^k cumplan con la Ecuación (4.1).

Finalmente, el último paso consiste en verificar si G^k es un mínimo I-map de $p(X)$. Si G^k no es un mínimo I-map, entonces existe al menos un supuesto que no se cumple en $p(X)$. Otra vez, es posible modificar G^k de modo de remover dicho supuesto y obtener un mínimo I-map de $p(X)$. Justamente esta es la labor del método ObtenerGrafoMinimoImap, el cual realiza modificaciones en G^k para volverlo un mínimo I-map de $p(X)$.

En lo que sigue, profundizaremos en los detalles involucrados en los métodos ObtenerGrafoImap y ObtenerGrafoMinimoImap. Sin embargo, antes de esto, es totalmente necesario remarcar un importante problema práctico que presenta el método ConstruirGrafoSolucion previamente presentado en el Algoritmo 6. El

conjunto $\mathcal{I}(G^k)$ puede contener un número exponencial de supuestos, resultando intratable verificar si G^k es un I-map de $p(X)$. Por ejemplo, para un grafo inicial G^k completo con $|V| = n$, el conjunto $\mathcal{I}(G^k)$ contiene exactamente 4^n supuestos. Para ilustrar cuán grande es este número, para $n = 132$, el conjunto $\mathcal{I}(G^k)$ contendría aproximadamente 10^{80} supuestos, es decir, tantos supuestos como átomos en el universo observable.

A pesar de esto último, utilizando el Teorema 3.2, podemos determinar si un grafo G^k es un I-map sin necesidad de utilizar explícitamente todos los supuestos en el conjunto $\mathcal{I}(G^k)$. Asumiendo una distribución $p(X)$ positiva, el conjunto $\mathcal{I}(G^k)$ equivale al conjunto $\mathcal{I}^\ell(G^k)$. El conjunto $\mathcal{I}(G^k)$ fue formalmente definido en la Definición 3.12. La importancia de esta equivalencia radica en el hecho de que el conjunto $\mathcal{I}^\ell(G^k)$ está formado exclusivamente por supuestos de la forma $I(x_a^k \perp x_{V'}^k \mid x_S^k)$, donde $V' = V \setminus S \cup \{ a \}$. Estos supuestos presentan dos características sumamente importantes para nuestro actual discusión. Primero, hay un supuesto por cada elemento $a \in V$. Segundo, el conjunto S de cualquier supuesto $I(x_a^k \perp x_{V'}^k \mid x_S^k) \in \mathcal{I}^\ell(G^k)$ equivale a la manta de Markov del nodo a en el grafo G, es decir, al conjunto de adyacencias del nodo a.

Así, dado un grafo G^k arbitrario, el conjunto $\mathcal{I}^\ell(G^k)$ puede ser definido como:

$$\mathcal{I}^\ell(G^k) = \{\, I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k) \,:\, \mathrm{sep}_G(a, V'; G(a)) \text{ para todo } a \in V \,\},$$

donde $V' = V \setminus G(a) \cup \{ a \}$.

En consecuencia, en un grafo G^k arbitrario, el conjunto $\mathcal{I}^\ell(G^k)$ contiene n supuestos, uno por cada variable $a \in V$. Este número de supuestos en $\mathcal{I}^\ell(G^k)$ no es exponencial, permitiéndonos verificar eficientemente si G^k es un I-map de $p(X)$.

Solucionado el problema de verificar si un grafo G^k es un I-map de $p(X)$, pasemos ahora con el análisis de los métodos ObtenerGrafoImap y ObtenerGrafoMinimoImap. Comencemos por el método ObtenerGrafoImap. Como hemos mencionado, sea G^k un grafo no I-map de una distribución $p(X)$, el objetivo de este

método es modificar G^k tal que dichas modificamos resulten en un nuevo grafo que sea un I-map de $p(X)$. Para entender en qué consisten estas modificaciones, primero es totalmente necesario comprender qué implicancia tiene, en términos gráficos, que un grafo G^k no sea un I-map de una distribución $p(X)$.

Para abordar esto último, tomaremos como base la equivalencia que existe entre los conjuntos $\mathcal{I}(G^k)$ y $\mathcal{I}^\ell(G^k)$. Así, si un grafo G^k no es un I-map de $p(X)$, entonces el conjunto $\mathcal{I}^\ell(G^k)$ debe contener uno o más supuestos de la forma $I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k)$ que no se cumplen en $p(X)$. ¿Qué significa específicamente que un supuesto $I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k)$ no se cumplen en $p(X)$? Responderemos esta pregunta en dos partes, la primera ofrece una respuesta general y la segunda ofrece una respuesta más específica. Así, como respuesta general, si un supuesto $I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k)$ no se cumple en $p(X)$, entonces el conjunto $G(a)$ contiene menos elementos de los que debería. Similarmente, el nodo a está separado de ciertos nodos que deberían pertenecer a sus adyacencias, i.e., al conjunto $G(a)$.

En el método ObtenerGrafoImap, esta última observación es crucial, por lo tanto, a través de un ejemplo simple, analizaremos más detalladamente esta primera respuesta, y luego generalizaremos dicho analizas de modo de ofrecer nuestra segunda respuesta.

Por ejemplo, una causa de que un grafo G^k no sea un I-map de $p(X)$ es que existe al menos una variable X_a que es independiente de otra variable, digamos X_b, en el contexto x^k. Pero, sin embargo, dichas variables, X_a y X_b, no son independientes en el contexto x^k en la distribución $p(X)$. Es decir, el supuesto $I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k)$ no se cumple en $p(X)$ porque el nodo b pertenece a V', pero el nodo b debería pertenecer a $G(a)$. Para ver esto último más claro, utilizaremos el siguiente resultado.

Proposición 4.1. Sea $p(X)$ una distribución positiva. Sea A, B y S subconjuntos disjuntos en V.

$$I(x_A \perp x_B \mid x_S) \iff \forall\ a \in A,\ \forall\ b \in B,\ I(x_a \perp x_b \mid x_S).$$

Demostración. Condición necesaria es una consecuencia de la aplicación reiterada de

descomposición. La condición suficiente es una consecuencia de aplicar transitividad. □

En palabras, en una distribución $p(X)$ positiva, cualquier supuesto de independencia que involucre conjuntos de variables puede descomponerse en un conjunto de independencias más simples, las cuales únicamente involucran pares de variables. Como una consecuencia de este resultado, un supuesto $I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k) \in \mathcal{I}^\ell(G^k)$ puede ser descompuesto de la siguiente manera.

Corolario 4.1. Sea $p(X)$ una distribución positiva. Sea $\{ a \}$, V' y $G(a)$ subconjuntos disjuntos en V.

$$I(x_a \perp x_{V'} \mid x_{G(a)}) \iff \forall\ b \in V',\ I(x_a \perp x_b \mid x_{G(a)}).$$

Demostración. La demostración de este corolario utiliza la Proposición 4.1. □

De esta manera, dado un grafo G^k arbitrario, aplicando el corolario anterior a cada supuesto en $\mathcal{I}^\ell(G^k)$, los supuestos de dicho conjunto pueden ser reexpresado como un conjunto de independencias más simples:

$$\mathcal{I}^*(G^k) = \{\ I(x_a^k \perp x_b^k \mid x_{G(a)}^k)\ :\ \mathrm{sep}_G(a, b; G(a)) \text{ para todo } a \in V, b \in V'\ \},$$

donde $V' = V \setminus (G(a) \cup \{ a \})$.

Lo interesante del conjunto $\mathcal{I}^*(G^k)$ es que todos sus supuestos involucran un par de variables junto con una manta. En base a esto, y volviendo a nuestro ejemplo anterior donde el supuesto $I(x_a^k \perp x_{V'}^k \mid x_{G(a)}^k)$ no se cumple en $p(X)$ porque $b \in V'$. Podemos utilizar el Coroloario 4.1 para descomponer dicho supuesto en un conjunto de supuestos entre pares de variables, donde uno de ellos justamente involucra el nodo b, es decir, tenemos el supuesto $I(x_a^k \perp x_b^k \mid x_{G(a)}^k)$. Dicho supuesto es una independencia falsa (error del tipo I), es decir, una independencia que no se cumple en $p(X)$. De esta manera, removiendo esta independencia falsa podemos lograr que el grafo G^k sea un I-map de $p(X)$.

Para ver cómo remover el supuesto $I(x_a^k \perp x_b^k \mid x_{G(a)}^k)$ de G^k, es necesario comprender cómo el grafo $G^k = (G, x^k)$ codifica dicho supuesto. En términos gráficos, está independencia implica la ausencia de la arista $(a, b) \in E$, donde E es el conjunto de aristas de G^k. En consecuencia, para lograr que G^k sea un I-map de $p(X)$, simplemente hay que agregar la arista (a, b) al conjunto E. Esto ocasionará que el nodo b forme parte del conjunto $G(a)$, removiendo el supuesto $I(x_a^k \perp x_b^k \mid x_{G(a)}^k)$. Resumiendo, **si un grafo no es un I-map, entonces existen una o más no-aristas (ausencia de aristas) que deberían estar presentes**. Así, agregando dichas aristas, removemos no-aristas, las cuales inducen independencias falsas.

Estas últimas observaciones nos permiten presentar el método ObtenerGrafoImap, el cual es formalmente mostrado en el Algoritmo 7.

Algoritmo 7 Modificar un grafo G^k para obtener un I-map de $p(X)$

```
procedure ObtenerGrafoImap(G^k, p(X))
    for (a, b) ∈ Ē do
        if I(x_a^k ⊥ x_b^k | x_{G(a)}^k) no se cumple en p(X) then
            E ← E ∪ { (a, b) }
return G^k
```

El método ObtenerGrafoImap tiene como entrada un grafo G^k y una distribución $p(X)$. La salida del dicho método consiste en retornar el grafo G^k pero modificándolo de tal manera que G^k sea un I-map de $p(X)$. Para lograr esto, el método ObtenerGrafoImap simplemente agrega aristas en G^k cada vez que encuentra una independencia falsa. Para encontrar una independencia falsa, el método ObtenerGrafoImap verifica si cada independencia de la forma $I(x_a^k \perp x_b^k \mid x_{G(a)}^k)$ se cumple en $p(X)$. Como estas independencias involucran un par de nodos, $a, b \in V$, el método ObtenerGrafoImap verifica cada no-arista en el grafo G^k. En el Algoritmo 7, el conjunto $\bar{E}$ denota el conjunto de no-aristas en el grafo G^k, es decir, $\bar{E} = (V \times V) \setminus E$[2]. Cada no-arista $(a, b) \in \bar{E}$ implica una independencia de la forma $I(x_a^k \perp x_b^k \mid x_{G(a)}^k)$, si esta independencia no se cumple en $p(X)$, entonces el método ObtenerGrafoImap agrega la arista (a, b) en el conjunto E de aristas. Es prudente señalar que si el grafo

[2]Por definición, un grafo no dirigido no contiene aristas (o no-aristas) que conectan un nodo con sí mismo.

G^k de entrada es un I-map de $p(X)$, entonces el método ObtenerGrafoImap no agrega ninguna arista en G^k, es decir, no se produce ninguna modificación sobre G^k.

Ahora, presentaremos el otro método que nos propusimos analizar, el método ObtenerGrafoMinimoImap, el cual modifica un grafo G^k (el cual se asume I-map) para obtener un mínimo I-map. Como mencionamos anteriormente, un grafo G^k no es un mínimo I-map de $p(X)$ porque contiene una o más aristas espurias, es decir, dependencias que no se cumplen en $p(X)$. Por lo tanto, para obtener un mínimo I-map, simplemente debemos remover todas aquellas aristas en el grafo que impliquen falsas dependencias. Esto es lo que realiza el método ObtenerGrafoMinimoImap, el cual es formalmente presentado en el Algoritmo 8.

Algoritmo 8 Modificar un grafo G^k para obtener un mínimo I-map de $p(X)$

```
procedure ObtenerGrafoMinimoImap(G^k, p(X))
    for (a, b) ∈ E do
        if I(x_a^k ⊥ x_b^k | x_{G(a)}^k) se cumple en p(X) then
            E ← E \ { (a, b) }
return G^k
```

El método ObtenerGrafoMinimoImap tiene dos parámetros de entrada, un grafo G^k y una distribución $p(X)$. La salida de dicho método es el grafo G^k, el cual es modificado para que sea un mínimo I-map de $p(X)$. Para lograr esto, el método ObtenerGrafoMinimoImap puede pensarse como la contraparte del método ObtenerGrafoImap. En otras palabras, el método ObtenerGrafoImap verifica que todas las independencias codificadas por G^k se cumplan en $p(X)$, mientras que el método ObtenerGrafoMinimoImap verifica que todas las dependencias codificadas por G^k se cumplen en $p(X)$. Si una dependencia no se cumple en $p(X)$, el método ObtenerGrafoMinimoImap remueve dicha dependencia del grafo G^k. Una dependencia no se cumple en $p(X)$, si una arista $(a, b) \in E$ tiene asociada una independencia $I(x_a^k \perp x_b^k \mid x_{G(a)}^k)$ que se cumple en $p(X)$. Si dicha arista tiene asociada tal independencia, entonces el método remueve la arista (a, b) del conjunto E, removiendo así la dependencia falsa. Vale remarcar que si el método ObtenerGrafoMinimoImap tiene como entrada un grafo G^k que es un mínimo I-map de $p(X)$,

entonces dicho método no produce modificaciones sobre el grafo G^k.

Para concluir esta sección, mostraremos que el método ConstruirGrafoSolucion, presentado en el Algoritmo 6, es esencialmente un algoritmo basado en restricciones. Para ver esto más claramente, primero hay que hacer explícita la siguiente observación. Los métodos ObtenerGrafoImap y ObtenerGrafoMinimoImap usan el contexto x^k únicamente para determinar si un supuesto de independencia se cumple (o no) en una distribución $p(X)$. En base a esto, el resto de las operaciones realizadas por ambos métodos pueden ser vistas como **operaciones realizadas sobre un grafo no dirigido** G. Para ser más concreto, supongamos lo siguiente. Definamos una función indicador f^k tal que retorne verdadero (o falso) cada vez que un supuesto de independencia se cumple (o no) en $p(X)$ bajo un contexto x^k. Ahora, redefinamos los métodos ObtenerGrafoImap y ObtenerGrafoMinimoImap tal que sus parámetros de entrada sean un grafo no dirigido G y la función f^k. Bajo tal suposición, la lógica de los métodos ObtenerGrafoImap y ObtenerGrafoMinimoImap permanecería idénticas, es decir, ambos métodos seguirían trabajando sobre un grafo G independientemente de cuál sea el contexto x^k (el cual se encuentra encapsulado en la función f^k). Sin embargo, el grafo G retornado por ambos métodos codificaría supuestos de independencia sobre el contexto x^k, debido a que la función f^k retorna valores de verdad para supuestos instanciados por el contexto x^k.

Otra manera de ver que el método ConstruirGrafoSolucion es simplemente un algoritmo basado en restricciones, consiste en compararlo frente a los algoritmos basados en restricciones. Por ejemplo, el algoritmo PC, discutido en la Sección 2.4.3, puede verse como el método ConstruirGrafoSolucion que únicamente utiliza el método ObtenerGrafoMinimoImap. Esto último es porque el algoritmo PC propone como grafo inicial un grafo completo, el cual es trivialmente un I-map, por lo tanto, no es necesario la utilización del método ObtenerGrafoImap. Por otro lado, el algoritmo GS, introducido en la Sección 2.4.4, puede ser pensado como el método ConstruirGrafoSolucion, donde ObtenerGrafoImap se corresponde con el método Grow del algoritmo GS (ver Algoritmo 3) y ObtenerGrafoMinimoImap se corresponde con el método Shrink del algoritmo GS (ver Algoritmo 4).

Resumiendo, el método ConstruirGrafoSolucion puede implementarse utilizando cualquier algoritmo basado en restricciones. En términos generales, un algoritmo basado en restricciones, el cual denotaremos como $\mathcal{L}$, tiene esencialmente dos parámetros de entrada: una prueba estadística de independencia, denotada T, y un conjunto de observaciones D tomadas desde una distribución $p(X)$. Para lograr que un algoritmo $\mathcal{L}$ construya un grafo canónico G^k, podemos utilizar una función f^k como parámetro de entrada en vez de un test estadístico T. De esta manera, el algoritmo $\mathcal{L}$ retornaría un grafo no dirigido G, cuya topología codificaría independencias instanciadas por el contexto x^k. Esta forma de implementar el método ConstruirGrafoSolucion es mostrada en el método ConstruirGrafoCanonico en el Algoritmo 9.

Algoritmo 9 Construcción de un grafo canónico por un algoritmo basado en restricciones

procedure ConstruirGrafoCanonico($D, x^k, \mathcal{L}, \mathrm{T}$)
  $\mathrm{T}^k \leftarrow$ modificar T en base al contexto x^k
  $G \leftarrow \mathcal{L}(\mathrm{T}^k, D)$
  $G^k \leftarrow (G, x^k)$
return G^k

El método ConstruirGrafoCanonico tiene como entrada cuatro parámetros: un conjunto D de observaciones tomadas desde una distribución $p(X)$, un estado $x^k \in val(V)$, un algoritmo $\mathcal{L}$ basado en restricciones, y una prueba estadística de independencia T. La salida del método ConstruirGrafoCanonico es un grafo canónico G^k, el cual codifica independencias instanciadas por el contexto x^k presentes en el conjunto D.

El método ConstruirGrafoCanonico construye un grafo $G^k = (G, x^k)$ realizando tres pasos. En el primer paso, la prueba estadística T es modificada utilizando el contexto x^k para obtener una prueba T^k. Los detalles de esta modificación serán presentados en la Sección 4.2, sin embargo, como comentamos anteriormente, la prueba T^k puede ser pensada como la función indicador f^k, la cual retorna el valor de verdad para cualquier supuesto instanciado por el contexto x^k. El segundo paso consiste en construir un grafo G cuya topología codifique independencias instanciadas

por el contexto x^k. Para construir este grafo G, el método ConstruirGrafoCanonico utiliza el algoritmo $\mathcal{L}$ con la prueba T^k (en vez de la prueba T). Como ya hemos comentado, al utilizar la prueba T^k, cada vez que $\mathcal{L}$ utilice T^k para determinar el valor de verdad de cualquier supuesto $I(X_a \perp X_b \mid X_C)$, la prueba T^k retornará el valor de verdad correspondiente al supuesto $I(x_a^k \perp x_b^k \mid x_C^k)$. Como resultado, el algoritmo $\mathcal{L}$ construirá un grafo G cuya topología únicamente codificará supuestos de independencias instanciados por el contexto x^k. Finalmente, una vez obtenido el grafo G, este es asociado al contexto x^k para obtener un grafo canónico G^k, el cual es retornado.

En la próxima sección mostraremos cómo modificar una prueba estadística T para ser utilizada para obtener valores de verdad para independencias instanciadas por un contexto x^k arbitrario.

4.2. Prueba de independencia instanciada por un contexto canónico

Como hemos visto en la Sección 2.4.2, una prueba estadística T es una función que puede ser utilizada para determinar el valor de verdad de un *supuesto de independencia condicional*. En términos generales, una prueba T intenta determinar si existe suficiente evidencia en D para aceptar o rechazar un supuesto. En otras palabras, sea el conjunto D una muestra tomada desde una distribución $p(X)$, una prueba estadística T puede ser vista como una función $\mathcal{I} \overset{\mathrm{T}}{\mapsto} \{ \mathit{verdadero}, \mathit{falso} \}$, donde $\mathcal{I}$ es el espacio de todos los posibles supuestos de independencia condicional sobre el dominio X [2].

De este modo, una prueba estadística T únicamente puede ser utilizada para determinar *el valor de verdad de un supuesto de independencia condicional*. En otras palabras, una prueba T no puede ser utilizada (en principio) para determinar el valor de verdad de un supuesto instanciado por un contexto canónico, e.g., $I(x_a^k \perp x_b^k \mid x_C^k)$. Sin embargo, existe una manera de obtener el valor de verdad de una independencia

instanciada utilizando una prueba T.

Según lo discutido en la Sección 2.1, la Ecuación (2.8) nos permite expresar un supuesto $I(X_a \perp X_b \mid X_C)$ como el siguiente conjunto de independencias contextuales:

$$I(X_a \perp X_b \mid X_C) = \bigwedge_{x^k \in val(\{C\})} I(X_a \perp X_b \mid x_C^k),$$

ahora, utilizando la Ecuación (2.9), cada independencia contextual $I(x_a \perp x_b \mid x_C^k)$ puede ser expresada como un conjunto de independencias instanciadas:

$$I(X_a \perp X_b \mid x_C^k) = \bigwedge_{x \in val(\{a,b\})} I(x_a \perp x_b \mid x_C^k).$$

Por lo tanto, el valor de verdad de un supuesto $I(x_a^k \perp x_b^k \mid x_C^k)$ puede ser obtenido indirectamente a través del valor de verdad del supuesto de independencia contextual $I(X_a \perp X_b \mid x_C^k)$. Luego, según el lemma 2.1, si el supuesto $I(X_a \perp X_b \mid \emptyset)$ se cumple en la *distribución condicional* $p(X_{V \setminus C} | x_C^k)$, entonces el supuesto $I(X_a \perp X_b \mid x_C^k)$ se cumple en $p(X)$.

Resumiendo lo anterior, si $I(X_a \perp X_b \mid \emptyset)$ se cumple en $p(X_{V \setminus C} | x_C^k)$, entonces el supuesto $I(X_a \perp X_b \mid x_C^k)$ se cumple en $p(X)$. Si el supuesto $I(X_a \perp X_b \mid x_C^k)$ se cumple, entonces el supuesto $I(x_a^k \perp x_b^k \mid x_C^k)$ debe también cumplirse en $p(X)$. Lo interesante de esto último es que el supuesto $I(X_a \perp X_b \mid \emptyset)$ es un supuesto de independencia condicional, *cuyo valor de verdad puede ser determinado utilizando cualquier prueba estadística de independencia.*

Sin embargo, hay que notar que para determinar el valor de verdad de $I(X_a \perp X_b \mid \emptyset)$, precisamos la distribución $p(X_{V \setminus C} | x_C^k)$, pero el conjunto D es una muestra tomada desde $p(X)$. Esto no es un problema, debido a que una muestra obtenida desde $p(X)$ es también una muestra para cualquiera de sus distribuciones condicionales. Por ejemplo, sea D una muestra tomada desde una distribución $p(X)$, entonces el subconjunto

$$D[x_C^k] = \{\, x \in D \colon x_C = x_C^k \,\} \subseteq D,$$

es necesariamente una muestra de la distribución $p(X_{V \setminus C} | x_C^k)$.

En consecuencia, utilizando el conjunto $D[x_C^k]$, el valor de verdad del supuesto $I(x_a^k \perp x_b^k \mid x_C^k)$ puede ser obtenido ejecutando la prueba estadística $\mathrm{T}(I(X_a \perp X_b \mid \emptyset), D[x_C^k])$. El procedimiento anteriormente descripto para obtener el valor de verdad de una independencia instanciada a través de una prueba estadística es formalmente mostrado en el Algoritmo 10.

Algoritmo 10 Prueba estadística para supuestos instanciados por x^k

procedure $\mathrm{T}^k(I(X_A \perp X_B \mid X_C); D)$
  $D[x_C^k] \leftarrow \{\, x \in D \colon x_C = x_C^k \,\}$, donde $x^k \in val(V)$
return $\mathrm{T}(I(X_A \perp X_B \mid \emptyset); D[x_C^k])$

El Algoritmo 10 muestra la función T^k la cual está definida en base a una prueba T y un contexto canónico x^k. La función T^k tiene dos parámetros de entrada: un supuesto de independencia condicional, $I(X_A \perp X_B \mid X_C)$, y un conjunto D de datos. En base a estos parámetros, la función T^k retorna el valor de verdad para el supuesto $I(x_A^k \perp x_B^k \mid x_C^k)$. Para determinar el valor de verdad de $I(x_A^k \perp x_B^k \mid x_C^k)$, se utiliza la prueba T para determinar el valor de verdad del supuesto $I(X_A \perp X_B \mid \emptyset)$ en el subconjunto $D[x_C^k] \subseteq D$.

4.3. Enfoque para construir grafos canónicos

Para construir un conjunto $\mathcal{G}$ de grafos canónicos, nuestro enfoque se dividide principalmente en dos etapas: i) proponer un conjunto $\mathcal{X}$ de contextos canónicos, donde $\mathcal{X} \subseteq val(V)$; y ii) para todo contexto $x^k \in \mathcal{X}$, construir un grafo canónico G^k. A continuación vamos a focalizarnos en la segunda etapa, dejando para más adelante los detalles sobre cómo obtener un conjunto $\mathcal{X}$. En base a las ideas presentadas en la sección anterior, el siguiente algoritmo muestra como obtener un conjunto $\mathcal{G}$ de grafos canónicos desde un conjunto $\mathcal{X}$.

El algoritmo anterior presenta el método ConstruirGrafosCanonicos, el cual tiene como parámetros de entrada: un conjunto D de datos; un conjunto $\mathcal{X}$ de contextos canónicos; un algoritmo $\mathcal{L}$ basado en restricciones; y una prueba estadística T. En base a estos parámetros, el método ConstruirGrafosCanonicos regresa un

Algoritmo 11 Enfoque para construir un conjunto de grafos canónicos

```
procedure ConstruirGrafosCanonicos(D, 𝒳, ℒ, T)
    𝒢 ← ∅
    for x^k en 𝒳 do
        G^k ← ConstruirGrafoCanonico(D, x^k, ℒ, T)
        𝒢 ← 𝒢 ∪ { G^k }
return 𝒢
```

conjunto $\mathcal{G} = \{ G^k : x^k \in \mathcal{X} \}$ de grafos canónicos, tal que cada grafo canónico $G^k \in \mathcal{G}$ codifica independencias instanciadas por el contexto x^k presentes en el conjunto D.

Para construir cada grafo canónico G^k, se utiliza el método ConstruirGrafoCanonico, el cual fue presentado en el Algoritmo 9. Como comentamos en la Sección 4.3, este método construye un grafo canónico G^k utilizando un algoritmo $\mathcal{L}$ junto con una prueba estadística T. Una vez construido el grafo canónico G^k, este es agregado al conjunto $\mathcal{G}$. Cuando todo contexto $x^k \in \mathcal{X}$ tiene un grafo canónico construido, entonces el método ConstruirGrafosCanonicos retorna el conjunto $\mathcal{G}$.

En lo que sigue, describiremos algunos puntos importantes relacionados al método ConstruirGrafosCanonicos. Más concretamente, trataremos tres puntos. El primero es cómo definir el conjunto $\mathcal{X}$ de contextos canónicos, lo cual describiremos en la Subsección 4.3.1. El segundo punto es cómo instanciar el método ConstruirGrafosCanonicos, para esto presentaremos los algoritmos CSPC y CSGS en la Subsección 4.3.2. Finalmente, el tercer punto será abordado en la Subsección 4.3.3. Este punto resalta el hecho de que el método ConstruirGrafosCanonicos puede ser fácilmente adaptado como un *algoritmo paralelo*, permitiendo ser ejecutado en *una arquitectura con múltiples computadoras*, e.g., en un *grid* o un *cluster*.

4.3.1. El conjunto de contextos canónicos

Para construir el conjunto $\mathcal{G}$ de grafos canónicos, el método ConstruirGrafosCanonicos necesita un conjunto de contextos canónicos $\mathcal{X} \subseteq val(V)$. Una alternativa para definir el conjunto $\mathcal{X}$ es utilizar el conjunto $val(V)$. Pero, de esta manera, el conjunto $\mathcal{X}$ tendría un tamaño exponencial, i.e. $|\mathcal{X}| = |val(V)| = 2^n$, volviendo

intratable la construcción del conjunto $\mathcal{G}$.

Para ilustrar esto, tomemos como ejemplo el conjunto de datos MSNBC, uno de los conjuntos de datos que utilizaremos en el Capítulo 5 para aprender un conjunto $\mathcal{G}$. Este conjunto tiene $n = 17$ variables binarias. Si $\mathcal{X} = val(V)$, entonces el conjunto $\mathcal{X}$ tendría 2^{17} contextos canónicos. En base a este conjunto $\mathcal{X}$, el método ConstruirGrafosCanonicos debe construir 2^{17} grafos canónicos. Para construir un grafo, hay que decidir la presencia/ausencia de cada posible arista en un grafo, lo cual implica al menos n^2 operaciones, donde cada operación requiere al menos la ejecución de una prueba estadística. En términos generales, el costo de ejecutar una prueba estadística es proporcional al tamaño del conjunto D de datos, en este caso el conjunto MSNBC. Por lo tanto, supongamos que cualquiera de las n^2 operaciones requiere un tiempo promedio de 1 segundo (el tiempo de recorrer el conjunto D para ejecutar una prueba estadística). Bajo este escenario, el método ConstruirGrafosCanonicos obtendría el conjunto $\mathcal{G}$ luego de un 1 año y medio de ejecución ininterrumpida.

Otra forma de definir el conjunto $\mathcal{X}$ es observar que el conjunto D de muestras tomadas desde una distribución es justamente *un subconjunto de val(V)*, es decir, es un conjunto de contextos canónicos. Por lo tanto, nuestra propuesta consiste en utilizar el conjunto D para definir el conjunto $\mathcal{X}$, debido a que el conjunto D puede pensarse como un conjunto de contextos canónicos obtenidos en base a la estructura de una distribución.

Así, el conjunto $\mathcal{X}$ puede ser definido desde D como sigue: todo estado $x \in val(V)$ que aparezca al menos una vez en el conjunto D es considerado un contexto canónico en el conjunto $\mathcal{X}$. Definiendo el conjunto $\mathcal{X}$ de esta manera, en el peor caso, el conjunto $\mathcal{X}$ tendrá $M = |D|$ contextos canónicos[3]. Por lo general, ampliamente se asume que en los problemas donde se utilizando los algoritmos basados en restricciones, el conjunto D no tiene un tamaño exponencial, en otro caso la ejecución de una única prueba de independencia sería intratable.

Utilizando la forma anterior para definir el conjunto $\mathcal{X}$, $\mathcal{X}$ tendrá un tamaño no exponencial. Por ejemplo, en nuestro ejemplo anterior sobre el conjunto MSNBC,

[3]El peor caso ocurré cuando todas las observaciones en D son distintas.

dicho conjunto tiene $|D| = M = 9228$ contextos canónicos no duplicados. Definiendo $\mathcal{X}$ con tales contextos, el método ConstruirGrafosCanonicos obtendría el conjunto $\mathcal{G}$ en aproximadamente un mes.

Finalmente, como comentario final, esta propuesta para definir el conjunto $\mathcal{X}$ no es óptima en el sentido de que no tenemos garantías de que el método ConstruirGrafosCanonicos construya un conjunto $\mathcal{G}$ que logre codificar todas las independencias instanciadas de una distribución. Analicemos un poco más en detalle las implicancias de esto último. Para esto asumamos que el método ConstruirGrafosCanonicos utiliza un "oráculo" en vez de una prueba T^k. Por oráculo, nos referimos a que cada vez que el método ConstruirGrafosCanonicos consulte el valor de verdad de un supuesto de independencia, el oráculo responderá con el valor de verdad correcto, independientemente del conjunto D utilizando. En otras palabras, el oráculo tiene acceso a un conocimiento exclusivo: conoce la distribución $p(X)$ que generó el conjunto D, la cual es totalmente desconocida desde la perspectiva del método ConstruirGrafosCanonicos.

Ante tal escenario, una definición incorrecta del conjunto $\mathcal{X}$ (ausencia de ciertos contextos canónicos) produce que el método ConstruirGrafosCanonicos obtenga un conjunto $\mathcal{G}$ que no codifica todas las independencias instanciadas de una distribución. En otras palabras, la estructura aprendida contiene errores, es decir, existen independencias instanciadas que no están siendo representadas. Como resultado, la ausencia de ciertas independencias instanciadas en el conjunto $\mathcal{G}$ puede producir que determinadas independencias condicionales o independencias específicas del contexto no sean codificadas. Esto último es una consecuencia de que toda independencia condicional e independencia específica del contexto está definida en base a conjuntos de independencias instanciadas (ver Sección 2.1).

A pesar de esto, y como veremos en el Capítulo 5, las estructuras aprendidas por nuestro enfoque, utilizando el conjunto $\mathcal{X}$ propuesto en esta sección, resultan ser más exactas en comparación a las estructuras aprendidas por otros algoritmos de aprendizaje de estructuras. Estos resultados nos permiten afirmar que, a pesar que el conjunto $\mathcal{X}$ puede no contener todos los contextos, la forma de definir dicho conjunto

permite a nuestro enfoque obtener estructuras prácticas, los cuales, como veremos, superan en términos de exactitud, a las estructuras obtenidas por varios algoritmos basados en restricciones.

4.3.2. Instanciaciones de nuestro enfoque

Uno de los elementos relevantes en el método ConstruirGrafoCanonicos es el algoritmo $\mathcal{L}$, el cual tiene como objetivo construir los grafos canónicos. Quizás, la característica más interesante del método ConstruirGrafoCanonicos es que $\mathcal{L}$ puede ser cualquier algoritmo basado en restricciones. A modo ilustrativo, presentamos a continuación dos instanciaciones del método ConstruirGrafoCanonicos: el algoritmo CSPC, donde $\mathcal{L}$ es definido como el algoritmo PC; y el algoritmo CSGS, donde $\mathcal{L}$ es definido como el algoritmo GS. Una descripción detallada de los algoritmos PC y GS puede ser encontrada en las Secciones 2.4.3 y 2.4.4, respectivamente.

Algoritmo 12 Algoritmo CSPC

procedure CSPC($D, \mathcal{X}$, T)
  $\mathcal{L}$ $\leftarrow$ PC (Algoritmo 1 en Sección 2.4.3)
  $\mathcal{G}$ $\leftarrow$ ConstruirGrafoCanonicos($D, \mathcal{X}, \mathcal{L}$, T)
return $\mathcal{G}$

El algoritmo CSPC es mostrado en Algoritmo 12. Dicho algoritmo es simplemente una *función envoltura* (*wrapper function*) del método ConstruirGrafosCanonicos. Esta función envoltura tiene como objetivo definir el algoritmo $\mathcal{L}$ como el algoritmo PC antes de llamar al método ConstruirGrafosCanonicos. Así, al ejecutar el método ConstruirGrafosCanonicos, el conjunto $\mathcal{G}$ es construido utilizando el algoritmo PC. En términos generales, al utilizar el algoritmo PC, la construcción de cada grafo canónico $G^k \in \mathcal{G}$ sigue un enfoque *bottom-up* tal como se comentó en la Sección 2.3. En otras palabras, el grafo solución es obtenido removiendo aristas desde un grafo completo. El código del algoritmo CSPC, junto con un manual de uso e instalación, están disponibles en línea en la siguiente dirección: `http://dharma.frm.utn.edu.ar/papers/ijait14/`. Para garantizar a los usuarios las libertades de uso, estudio, copia y modificación del código del algoritmo CSPC,

el mismo está liberado bajo la licencia *GNU GENERAL PUBLIC LICENSE Version 3*[4].

Algoritmo 13 Algoritmo CSGS

procedure CSGS($D, \mathcal{X}$, T)
  $\mathcal{L} \leftarrow$ GS (Algoritmo 2 en Sección 2.4.4)
  $\mathcal{G} \leftarrow$ ConstruirGrafoCanonicos($D, \mathcal{X}, \mathcal{L}$, T)
return $\mathcal{G}$

Por otro lado, el algoritmo CSGS es mostrado en Algoritmo 13. Igual que el algoritmo PC, el algoritmo CSGS es una función envoltura que define el algoritmo $\mathcal{L}$ como el algoritmo GS. A diferencia del algoritmo CSPC, el algoritmo CSGS construye cada grafo $G^k \in \mathcal{G}$ usando un enfoque *top-down* tal como se comentó en la Sección 2.3. En otras palabras, el grafo solución es obtenido agregando aristas en un grafo vacío[5]. El código del algoritmo CSGS, junto a un manual de uso e instalación, se encuentra disponible en línea en la siguiente dirección: `http://dharma.frm.utn.edu.ar/papers/iberamia14/`. Al igual que el código del algoritmo CSPC, el código del algoritmo CSGS está liberado bajo la licencia GNU GENERAL PUBLIC LICENSE Version 3.

4.3.3. Aprendizaje paralelo de grafos canónicos

Una característica interesante del método ConstruirGrafosCanonicos es que la construcción de todo grafo canónico $G(x^k) \in \mathcal{G}$ es realizada independientemente de la construcción de los restantes $\mathcal{G} \setminus \{ G(x^k) \}$ grafos canónicos. En programación paralela, un problema con esta característica es llamado problema *perfectamente paralelo* (*embarrassingly parallel problem* o *perfectly parallel problem*) [31, § 1.4.4]. Un problema perfectamente paralelo puede ser trivialmente adaptado para ser ejecutado en forma paralela. Para ilustrar este hecho, el siguiente algoritmo muestra una

[4]Vale remarcar que a licencia GNU GENERAL PUBLIC LICENSE Version 3 es una licencia *copyleft*. Una licencia copyleft se caracteriza por ofrecer *derechos a los usuarios*. Estos derechos consisten en las libertades de distribuir y modificar versiones del trabajo licenciado, *siempre y cuando* los mismos derechos se preserven en cualquier copia o adaptación de dicho trabajo.

[5]Para ser más correcto, el algoritmo GS no sólo agrega aristas, sino que también remueve aristas con el objetivo de reducir errores del tipo II.

adaptación del método ConstruirGrafosCanonicos para ser ejecutado en forma paralela.

Algoritmo 14 Construir un conjunto de grafos canónicos en forma paralela

procedure EjecucionParalela($D, \mathcal{X}, \mathcal{L}, \mathrm{T}, N$)
  $\mathcal{G} \leftarrow \emptyset$
  $\{ \mathcal{X}_i \}_{i=1}^{N} \leftarrow$ Dividir $\mathcal{X}$ en N particiones
  parallel for $\mathcal{X}_i$ en $i = 1, \ldots, N$ **do**
    $\mathcal{G}_i \leftarrow$ ConstruirGrafosCanonicos($D, \mathcal{X}_i, \mathcal{L}, \mathrm{T}$)
    $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{G}_i$
return $\mathcal{G}$

El método EjecucionParalela tiene como entrada los mismos parámetros de entrada del método ConstruirGrafosCanonicos junto a un nuevo parámetro, N, donde N es un número entero entre 1 a $|\mathcal{X}|$. Al igual que ConstruirGrafosCanonicos, el método EjecucionParalela retorna un conjunto $\mathcal{G}$ de grafos canónicos. Sin embargo, para obtener el conjunto $\mathcal{G}$, el método EjecucionParalela divide el problema de construir $\mathcal{G}$ en N subproblemas, donde cada subproblema puede ser resuelto independientemente de los restantes.

Para lograr esto, el método EjecucionParalela es definido como una función envoltura sobre el método ConstruirGrafosCanonicos. Esta función envoltura divide el conjunto $\mathcal{X}$ en N particiones, i.e. $\mathcal{X} = \bigcup_{i=1}^{N} \mathcal{X}_i$, donde $\mathcal{X}_i$ denota la partición i-ésima de $\mathcal{X}$. Ahora, por cada partición $\mathcal{X}_i$, el método EjecucionParalela utiliza ConstruirGrafosCanonicos para obtener un conjunto $\mathcal{G}_i$, donde $\mathcal{G}_i$ es un conjunto de grafos canónicos obtenidos desde el conjunto $\mathcal{X}_i$. Una vez que toda partición $\{\mathcal{X}_i\}_{i=1}^{N}$ tiene construido un conjunto $\mathcal{G}_i$, el método EjecucionParalela retorna el conjunto $\mathcal{G} = \cup_{i=1}^{N} \mathcal{G}_i$.

Es importante notar que el método EjecucionParalela construye el conjunto $\mathcal{G}$ utilizando llamadas en paralelo del método ConstruirGrafosCanonicos (notar el **parallel for** en el Algoritmo 14). Así, en base al número de computadoras disponibles, las diferentes particiones del conjunto $\mathcal{G}$ son construidas en paralelo. Asumiendo una arquitectura con N computadoras, al utilizar el método EjecucionParalela para construir el conjunto $\mathcal{G}$. El costo temporal del método ConstruirGrafosCa-

NONICOS puede ser reducido linealmente en N. En el caso de que $N = |\mathcal{X}|$, es decir, cada partición $\mathcal{X}_i$ involucra un único contexto canónico. El costo temporal de construir el conjunto $\mathcal{G}$ se reduce al costo de construir un único grafo, es decir, el costo temporal que requiere el algoritmo $\mathcal{L}$ para construir un grafo. En la siguiente sección, analizaremos más en detalle la complejidad temporal para construir un conjunto $\mathcal{G}$ utilizando nuestro enfoque.

4.4. Tiempo de ejecución

En esta sección presentamos un análisis sobre la complejidad temporal que requiere el método ConstruirGrafosCanonicos para construir un conjunto $\mathcal{G}$. Según lo mencionado en la Sección 2.4, el tiempo de ejecución de un algoritmo basado en restricciones es usualmente analizado en términos del número de decisiones que necesita para construir un grafo solución. Una decisión consiste en determinar el valor de verdad de un supuesto de independencia arbitrario, lo cual implica la ejecución de una prueba estadística. Debido a que el método ConstruirGrafosCanonicos es definido en base a un algoritmo $\mathcal{L}$ basado en restricciones, nuestro análisis consistirá en *determinar el número de decisiones tomadas por* ConstruirGrafosCanonicos *para construir un conjunto* $\mathcal{G}$. Por claridad, nuestro análisis será efectuado en dos partes. Primero, en la Sección 4.4.1, analizaremos el número de decisiones involucradas en la construcción del conjunto $\mathcal{G}$. Esta primera parte básicamente resume e integra toda la discusión sobre la complejidad temporal de los algoritmos PC y GS, la cual fue brindada en la Secciones 2.4.3 y 2.4.4. Por lo tanto, si el lector tiene presente las complejidades temporales de dichos algoritmos, la lectura de la Sección 4.4.1 no es necesaria. Por otro lado, la segunda parte, presentada en la Sección 4.4.2, analiza el costo temporal involucrado en tomar una decisión arbitraria.

4.4.1. Número de decisiones

Ante todo, la construcción del conjunto $\mathcal{G}$ implica la construcción reiterada de grafos canónicos, en consecuencia, el número total de decisiones involucradas en cons-

truir el conjunto $\mathcal{G}$ es proporcional al número de grafos canónicos en $\mathcal{G}$. Como vimos en la Sección 4.3, el número de grafos canónicos está definido por el número de contextos en el conjunto $\mathcal{X}$. Según lo visto en la Sección 4.3.1, el número de contextos en $\mathcal{X}$ es igual a $M = |D|$ en el peor caso. Así, el conjunto $\mathcal{G}$ involucra un número de decisiones proporcional a $O(M)$. Como vimos en la Sección 4.3.3, la construcción de $\mathcal{G}$ es un problema perfectamente paralelo, entonces el número $O(M)$ de decisiones puede ser reducido idealmente a $O(1)$. En otras palabras, el número de decisiones para construir $\mathcal{G}$ es proporcional al número de decisiones para construir un grafo canónico G^k. Pero, ¿cuál es el número de decisiones para construir un grafo canónico G^k? Este número de decisiones depende del algoritmo $\mathcal{L}$ utilizado por el método ConstruirGrafosCanonicos. Como vimos en la Sección 4.3.2, propusimos dos instanciaciones del método ConstruirGrafosCanonicos, los algoritmos CSPC y CSGS. En el algoritmo CSPC, el método ConstruirGrafosCanonicos utiliza el algoritmo PC como algoritmo $\mathcal{L}$, mientras que, en el algoritmo CSGS, el algoritmo GS es utilizado como algoritmo $\mathcal{L}$. A continuación, analizaremos la complejidad temporal de los algoritmos PC y GS para construir un grafo. Este análisis está fuertemente basado en la discusión de complejidad temporal de ambos algoritmos presentada en las Secciones 2.4.3 y 2.4.4, respectivamente.

Comencemos con el número de decisiones que requiere el algoritmo PC para construir un grafo. Sea n el número de nodos en un grafo $G = (V, E)$, el algoritmo PC construye un grafo solución removiendo aristas desde un grafo inicial, el cual es un grafo completo. Para remover una arista $(a, b) \in E$ arbitraria, el algoritmo PC utiliza una prueba estadística para determinar si el supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$ es verdadero. Así, el número de decisiones tomadas por el algoritmo PC es proporcional al número de aristas en un grafo completo, es decir, el algoritmo PC toma al menos $O(\binom{n}{2} \approx n^2)$ decisiones. Sin embargo, como vimos en la Sección 2.4, para construir un grafo con sólo $O(n^2)$ decisiones, es necesario un número exponencial de datos. Para solucionar este problema, el algoritmo PC puede utilizar un enfoque incremental. En el enfoque incremental, el valor de verdad del supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$ es obtenido a través de los valores de verdad de varios supuestos de la forma $I(X_a \perp X_b \mid X_C)$,

donde C es un subconjunto de $G(a) \setminus \{ b \}$. El enfoque es incremental porque considera supuestos $I(X_a \perp X_b \mid X_C)$ tal que sus subconjuntos C son de tamaño creciente. En el peor caso, para determinar el valor de verdad de un supuesto $I(X_a \perp X_b \mid X_{G(a)\setminus\{b\}})$, es necesario determinar el valor de verdad de tantos supuestos como subconjuntos hay en el conjunto $G(a) \setminus \{ b \}$. Como el grafo inicial es el grafo completo, el conjunto $G(a) \setminus \{ b \}$ contiene $(n-2)$ nodos, entonces el conjunto $G(a) \setminus \{ b \}$ tiene $2^{(n-2)}$ subconjuntos posibles. Para evitar tomar un número exponencial de decisiones, el enfoque incremental puede ser acotado utilizando un límite máximo, denotado por $K_{\max}$, para el tamaño de C, i.e., $|C| = K_{\max}$. De este modo, utilizando un valor límite $K_{\max}$, cada vez que el algoritmo PC intenta remover una arista (a, b), $\sum_{i=0}^{K_{\max}} \binom{n-2}{i}$ decisiones son tomadas en el peor caso. En base a esto, como el grafo inicial tiene n^2 aristas, el algoritmo PC necesita tomar a lo sumo $O(n^2 \sum_{i=0}^{K_{\max}} \binom{n-2}{i})$ decisiones para construir un grafo solución.

Ahora, analicemos el número de decisiones que requiere el algoritmo GS para construir un grafo. A diferencia de PC, el algoritmo GS utiliza un grafo vacío como grafo inicial, e identifica la manta de cada nodo $a \in V$ para obtener un grafo solución. Dado que hay $n = |V|$ nodos, el algoritmo GS toma un número de decisiones proporcional a n. Ahora, una manta $G(a)$ es identificada utilizando las fases de crecimiento (mostrada en el Algoritmo 3) y encogimiento (mostrada en el Algoritmo 4). La fase de crecimiento agrega nuevos nodos a una manta $G(a)$. Para esto, define un conjunto U de nodos candidatos, i.e. $U = V \setminus G(a) \cup \{ a \}$, y, por cada nodo $b \in U$, se decide si agregar b o no a la manta $G(a)$. Por lo tanto, la fase de crecimiento toma $|U|$ decisiones, una por cada nodo candidato en U. Debido a que inicialmente cualquier manta es un conjunto vacío, entonces el tamaño máximo del conjunto U es $n-1$. Así, en el peor caso, la fase de crecimiento toma $n-1$ decisiones. Finalizada la fase de crecimiento, la fase de encogimiento decide cuáles nodos en $G(a)$ remover. Para esto, la fase de encogimiento toma una decisión por cada nodo en $G(a)$, lo cual implica $|G(a)|$ decisiones. En el peor caso, el conjunto $G(a)$ puede tener $n-1$ nodos, lo cual implica que la fase de encogimiento puede tomar a lo sumo $n-1$ decisiones por cada manta. Resumiendo, la identificación de una manta involucra $(n-1)(n-1) \approx n^2$

decisiones en el peor caso. Debido a que hay n mantas a identificar, el algoritmo GS toma $O(n^3)$ decisiones en el peor caso.

Finalmente, asumiendo que $|\mathcal{X}| = M$, el Cuadro 4.1 muestra el número de decisiones que involucra la construcción del conjunto $\mathcal{G}$ por los algoritmos CSPC y CSGS.

Número de decisiones	
CSPC	CSGS
$O(M \cdot n^2 \sum_{i=0}^{K_{\max}} \binom{n-2}{i})$	$O(M \cdot n^3)$

Cuadro 4.1: Número de decisiones para construir el conjunto $\mathcal{G}$, donde $\mathcal{G}$ está formado por $M = |\mathcal{G}|$ grafos canónicos, donde cada grafo canónico tiene n nodos.

4.4.2. Complejidad temporal de una decisión

Como vimos anteriormente, la Tabla 4.1 presenta el número de decisiones que toman los algoritmos CSPC y CSGS. Sin embargo, la pregunta que resta responder ahora es, ¿cuál es el tiempo necesario para tomar una decisión? Como hemos comentado, tomar una decisión involucra ejecutar una prueba estadística. En términos generales, para determina el valor de verdad de un supuesto, una prueba estadística construye una tabla de contingencia (para más detalles, consultar la Sección 2.4.2). La construcción de una tabla de contingencia involucra recorrer cada ejemplo en el conjunto D. Por lo tanto, el costo de ejecutar un prueba es $O(M)$, donde $M = |D|$. Así, si el número M es grande, el costo de construir un grafo puede ser particularmente alto.

Una forma de reducir considerablemente el costo involucrado en la construcción de una tabla de contingencia es utilizando una estructura de datos llamada *ADTree* [66]. El ADTree puede pensarse como una cache que almacena todas las posibles tablas de contingencias para un dominio en particular. En otras palabras, una vez creado un ADTree, todas las posibles tablas de contingencia entre variables se encuentran almacenadas en el ADTree jerárquicamente, lo cual facilita su acceso. De esta manera, utilizando un ADTree, una tabla de contingencia utilizada por una prueba estadística puede ser eficientemente recuperada desde el ADTree en vez de ser construida

recorriendo el conjunto D. En consecuencia, el tiempo requerido para construir cualquier tabla de contingencia se reduce a una constante independiente de M, es decir, el número de ejemplos en D. Más concretamente, como se muestra en [66], el costo de recuperar una tabla de contingencia en un ADTree es log-linear al número de variables involucradas en la tabla de contingencia.

De este modo, al utilizar el ADTree para construir un grafo, logramos un *tradeoff espacio-tiempo*, es decir, *la complejidad temporal de construir un grafo es reducida a expensas de un mayor consumo de memoria*. Sin embargo, cuando nuestro problema presenta una alta dimensionalidad, la construcción y el almacenamiento de un ADTree requieren una cantidad de tiempo y memoria exponencial [47]. Esto se debe a que la construcción del ADTree implica, en términos generales, construir todas las posibles tablas de contingencia para un dominio en particular, lo cual implica una cantidad de tiempo/espacio exponencial. Para estos casos, existen varias soluciones para reducir el costo de construir y almacenar un ADTree. A continuación, describiremos dos de estas soluciones.

La primera solución consiste en construir el ADTree *bajo demanda*. Esto es, en vez de crear un ADTree con todas las posibles tablas de contingencia para un domino en particular, el ADTree es inicialmente construido vacío. Cada vez que una tabla de contingencia es solicitada al ADTree, el ADTree verifica si dicha tabla se encuentra internamente almacenada. Si la tabla se encuentra almacenada, entonces esta es retornada. En caso contrario, el ADTree construye dicha tabla, recorriendo todos los ejemplos en D, la almacena, y finalmente la retorna. Bajo este enfoque, el ADTree únicamente almacena aquella tablas que son demandadas al menos una vez. Esto permite reducir significativamente el tiempo de construcción del ADTree como también el espacio necesario para almacenarlo [47].

La segunda solución es llamada *Leaf-List* [66]. En términos sencillos, esta solución consiste en evitar que un ADTree almacene ciertas tablas de contingencia. Para lograr esto, la idea consiste en construir ciertas tablas cada vez que son solicitadas en vez de almacenarlas y recuperarlas. Esto permite controlar el tamaño máximo que puede tener un ADTree en memoria, a expensas de aumentar ligeramente el costo temporal

de recuperar ciertas tablas de contingencia. El truco detrás de esta solución, se basa en la siguiente intuición. Como hemos dicho, para construir una tabla de contingencia, es necesario recorrer los M ejemplos en el conjunto D. Sin embargo, en ciertas tablas, los valores en sus celdas involucran un reducido número de ejemplos[6]. En base a esto, es válido pensar que una tabla de contingencia se encuentra asociada a un determinado número de ejemplos en el conjunto D. Por así decirlo, algunas tablas de contingencia estarán asociadas a más ejemplos en comparación a otras. Para ver esto último desde otra perspectiva, si conocemos los ejemplos asociados a una tabla de contingencia, entonces la construcción de ciertas tablas requerirán una menor cantidad de tiempo en comparación a otras, debido a que unas están asociadas a menos ejemplos que otras. Así, definiendo una variable umbral $M_{\text{mín}}$, toda tabla de contingencia asociada a menos de $M_{\text{mín}}$ ejemplos, no es almacenada en el ADTree, sino, en cambio, es construida cada vez que esta es solicitada. Para reducir el costo de construir dicha tabla de contingencia, el ADTree almacena los $M_{\text{mín}}$ ejemplos involucrados en la construcción de dicha tabla (a través de una estructura de datos llamada Leaf-List, de ahí el nombre de la solución). De esta manera, cuando una tabla está asociada a menos de $M_{\text{mín}}$ ejemplos, el costo de construir dicha tabla de contingencia es proporcional a $O(M_{\text{mín}})$ en vez de $O(M)$.

4.5. Inducción de características

Una vez obtenida la estructura de un modelo canónico, el conjunto $\mathcal{G}$ de grafos canónicos, el conjunto de parámetros del modelo canónico puede ser aprendido con las técnicas de estimación de parámetros mostradas en la Sección 2.5. Obtenido un modelo canónico (estructura + parámetros), podemos utilizarlo para realizar inferencia, e.g., utilizando las técnicas mostradas en la Sección 2.6. Sin embargo, debido a que un modelo canónico utiliza una representación novedosa de la estructura, un conjunto de grafos canónicos, el aprendizaje de parámetros y la inferencia no pueden ser realizados utilizando paquetes de software bien conocidos. Esto último se debe

[6]Para más detalles sobre el significado de las celdas de una tabla de contingencia, ver Sección 2.4.2.

a que dichos paquetes fueron originalmente diseñados para utilizar representaciones estándares de la estructura, e.g., un conjunto de características o un grafo simple.

En la práctica, una forma de resolver el problema anteriormente mencionado consiste en inducir un conjunto $\mathcal{F}$ de características desde la estructura cuya representación no es convencional. Por ejemplo, este estrategia puede ser encontrada en dos trabajos en el área [51, 56]. De esta manera, el conjunto $\mathcal{F}$ obtenido desde la estructura puede ser utilizado para estimar parámetros, o realizar inferencia, a través de cualquier paquete de software bien conocido. La idea detrás de inducir un conjunto $\mathcal{F}$ consiste en construir características de tal modo que codifiquen los mismos supuestos de independencias codificados por la estructura.

Esta sección presenta un algoritmo para inducir un conjunto $\mathcal{F}$ de características desde un conjunto $\mathcal{G}$ de grafos canónicos. Este algoritmo induce un conjunto $\mathcal{F}$ tal que sus características codifican los mismos supuestos de independencia codificados por cada uno de los grafos canónicos en el conjunto $\mathcal{G}$. Sin embargo, es necesario realizar ciertas advertencias sobre las características inducidas por dicho algoritmo. Básicamente, nuestra propuesta para inducir características puede introducir errores del tipo I, es decir, independencias instanciadas que no están presentes en el conjunto $\mathcal{G}$. Como veremos más adelante, estos errores no surgen de un problema lógico en el método de inducción, sino más bien, como una consecuencia de intentar interpretar la ausencia de todos los restantes grafos canónicos que no pertenecen al conjunto $\mathcal{G}$ de grafos canónicos.

De este modo, esta sección está dividida en dos partes. En la primera parte, la Sección 4.5.1, presentaremos nuestro algoritmo para inducir características e ilustraremos su funcionamiento a través de un ejemplo. En la segunda parte, la Sección 4.5.2, discutiremos más a fondo los errores que puede introducir el algoritmo de inducción de características.

4.5.1. Método para inducir características

El Algoritmo 15 presenta un método llamado INDUCIR, el cual puede ser utilizado para inducir un conjunto $\mathcal{F}$ de características desde un conjunto $\mathcal{G}$ de grafos

canónicos. El método INDUCIR tiene dos parámetros de entrada: un conjunto $\mathcal{G}$ de grafos canónicos y un conjunto X de variables. En base a estos parámetros, el método INDUCIR induce un conjunto $\mathcal{F}$ de características, cada una de las cuales codifica los mismos supuestos de independencia codificados por cada grafo canónico en el conjunto $\mathcal{G}$. Para obtener el conjunto $\mathcal{F}$, el método INDUCIR sucesivamente induce un conjunto $\mathcal{F}'$ de características desde cada grafo canónico $G^k \in \mathcal{G}$, donde $G^k = (G, x^k)$, utilizando el método INDUCIRDESDEGRAFO. Una vez que un conjunto de características fue inducido desde cada grafo canónico en $\mathcal{G}$, el método INDUCIR construye un conjunto características unitarias para cada variable $X_a \in X$, es decir, construye una característica f_a por cada uno de los estados en $val(a)$. Como se muestra en [18, 88], este paso garantiza que, al estimar los parámetros del conjunto $\mathcal{F}$, la distribución resultante sea positiva. Una vez construidas las características unitarias, el método INDUCIR retorna el conjunto $\mathcal{F}$.

Algoritmo 15 Inducción de características desde un conjunto $\mathcal{G}$

```
procedure INDUCIR(G, X)
    F ← ∅
    for (G, x^k) ∈ G do
        F' ← INDUCIRDESDEGRAFO(G, x^k)
        F ← F ∪ F'
    F' ← Características unitarias para todo X_a ∈ X.
    F ← F ∪ F'
return F
```

Para obtener un conjunto $\mathcal{F}'$ de características desde un grafo canónico (G, x^k), INDUCIR utiliza el método INDUCIRDESDEGRAFO. Este método es mostrado formalmente en el Algoritmo 16. En términos sencillos, dicho método consiste de dos pasos. El primer paso consiste en obtener el conjunto de cliques máximos del grafo G, denotado por $\mathcal{C}(G)$[7]. El segundo paso consiste en utilizar cada uno de los cliques en $\mathcal{C}(G)$, junto con el contexto canónico x^k, para construir características. Para construir dichas características, INDUCIR procede de la siguiente manera. Sea C un clique máximo obtenido desde el grafo canónico (G, x^k), una característica f_C^k es construi-

[7]Una forma de obtener los cliques máximos de un grafo G es utilizando el *algoritmo Bron-Kerbosch* [14].

da definiendo una función indicador sobre el estado que toman las variables X_C en el contexto x^k, es decir, el estado $x_C^k = \bigwedge_{a \in C} x_a^k$. De esta manera, por cada clique $C \in \mathcal{C}(G)$, INDUCIR construye una característica, cada una de las cuales forma parte del conjunto $\mathcal{F}'$.

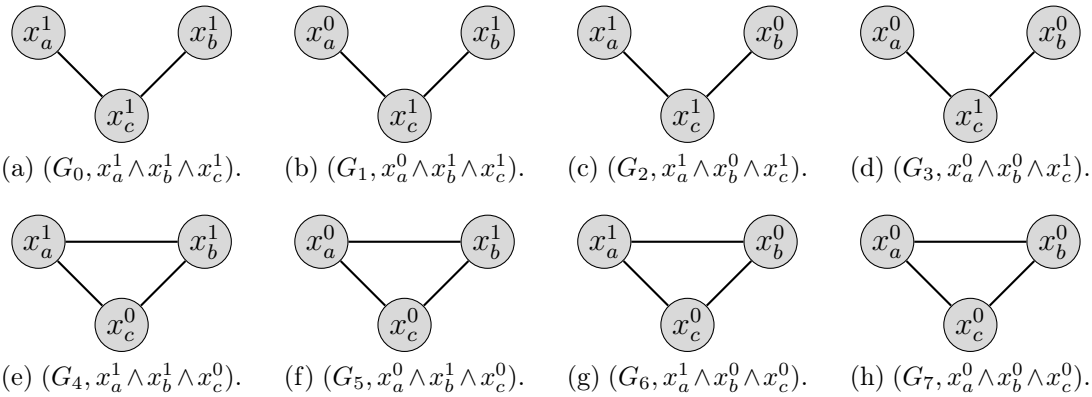
Algoritmo 16 Inducción de características desde un grafo G^k

```
procedure INDUCIRDESDEGRAFO(G, x^k)
    F ← ∅
    for C ∈ C(G) do
        f_C^k ← Construir una función indicador con ⋀_{a∈C} x_a^k
        F ← F ∪ { f_C^k }
return F
```

De esta manera, a través del método INDUCIR podemos generar características desde un conjunto de grafos canónicos. Para ilustra en más detalle el método INDUCIR, presentamos el siguiente ejemplo.

Ejemplo 4.1. En el Ejemplo 3.1 se presentó una distribución $p(X)$ cuya estructura involucraba independencias específicas del contexto. En el Ejemplo 3.17, vimos que la estructura de dicha distribución podía ser correctamente representada utilizando un conjunto $\mathcal{G}$ de grafos canónicos. La Figura 4-1 muestra este conjunto de grafos canónicos (G_0, G_1, G_2 y G_3), junto con otros grafos canónicos adicionales. Estos grafos adicionales (los grafos G_4, G_5, G_6 y G_7) fueron agregados para ilustrar de mejor manera el funcionamiento del método INDUCIR. En este ejemplo, mostraremos paso a paso cómo el método INDUCIR obtiene un conjunto $\mathcal{F}$ desde el conjunto $\mathcal{G}$ de grafos canónicos mostrados en la Figura 4-1.

Para obtener un conjunto $\mathcal{F}$, el método INDUCIR obtiene un conjunto de características para cada grafo canónico en $\mathcal{G}$. Dado un grafo canónico (G, x^k) arbitrario, el método INDUCIR obtiene el conjunto $\mathcal{C}(G)$ de cliques máximos en G y luego, por cada clique $C \in \mathcal{C}(G)$, define una característica utilizando el contexto canónico x^k. Debido a que el conjunto $\mathcal{G}$ contiene 8 grafos canónicos, el método INDUCIR inducirá 8 conjuntos de características, las cuales denotaremos por $\mathcal{F}_i$, con $i = 0, \ldots, 7$. La Figura 4-2 muestra los conjuntos de características inducidos desde cada uno de los grafos $(G_i, x^k) \in \mathcal{G}$, con $i = 0, \ldots, 7$, mostrados en la Figura 4-1.



(a) $(G_0, x_a^1 \wedge x_b^1 \wedge x_c^1)$. (b) $(G_1, x_a^0 \wedge x_b^1 \wedge x_c^1)$. (c) $(G_2, x_a^1 \wedge x_b^0 \wedge x_c^1)$. (d) $(G_3, x_a^0 \wedge x_b^0 \wedge x_c^1)$.

(e) $(G_4, x_a^1 \wedge x_b^1 \wedge x_c^0)$. (f) $(G_5, x_a^0 \wedge x_b^1 \wedge x_c^0)$. (g) $(G_6, x_a^1 \wedge x_b^0 \wedge x_c^0)$. (h) $(G_7, x_a^0 \wedge x_b^0 \wedge x_c^0)$.

Figura 4-1: Conjunto $\mathcal{G}$ de grafos canónicos que codifican las independencias de $p(X)$.

$$\mathcal{F}_0 = \{\ (x_a^1 \wedge x_c^1), (x_b^1 \wedge x_c^1)\ \}; \qquad \mathcal{F}_4 = \{\ (x_a^1 \wedge x_b^1 \wedge x_c^0)\ \};$$

$$\mathcal{F}_1 = \{\ ((x_a^0 \wedge x_c^1), (x_b^1 \wedge x_c^1)\ \}; \qquad \mathcal{F}_5 = \{\ (x_a^0 \wedge x_b^1 \wedge x_c^0)\ \};$$

$$\mathcal{F}_2 = \{\ (x_a^1 \wedge x_c^1), (x_b^0 \wedge x_c^1)\ \}; \qquad \mathcal{F}_6 = \{\ (x_a^1 \wedge x_b^0 \wedge x_c^0)\ \};$$

$$\mathcal{F}_3 = \{\ (x_a^0 \wedge x_c^1), (x_b^0 \wedge x_c^1)\ \}; \qquad \mathcal{F}_7 = \{\ (x_a^0 \wedge x_b^0 \wedge x_c^0)\ \}.$$

Figura 4-2: Características inducidas desde el conjunto $\mathcal{G}$ de grafos canónicos mostrado en la figura 4-1.

Analicemos más en detalles los conjuntos de características mostrados en la Figura 4-2. Por ejemplo, el conjunto $\mathcal{F}_0$ contiene las características inducidas desde el grafo canónico $(G_0, (x_a^1 \wedge x_b^1 \wedge x_c^1))$. Como el grafo G_0 tiene el conjunto $\mathcal{C}(G_0) = \{\ \{\ a, c\ \}, \{\ b, c\ \}\ \}$ de cliques máximos, entonces el conjunto $\mathcal{F}_0$ tiene dos características, una por cada clique en $\mathcal{C}(G_0)$. Estas características son definidas utilizando el contexto del grafo canónico, el estado canónico $(x_a^1 \wedge x_b^1 \wedge x_c^1)$. Utilizando dicho contexto, las dos características del conjunto $\mathcal{F}_0$ son definidas en base a los estados $(x_a^1 \wedge x_c^1)$ y $(x_b^1 \wedge x_c^1)$.

Dado que los grafos $G_1, G_2,$ y G_3 tienen la misma estructura que el grafo G_0, entonces el conjunto $\mathcal{C}$ de cliques máximos en todos estos grafos es el mismo, es decir, cada uno de estos grafos tiene el siguiente conjunto de clique máximo: $\{\ \{\ a, c\ \}, \{\ b, c\ \}\ \}$. En consecuencia, los conjuntos $\mathcal{F}_0, \ldots, \mathcal{F}_3$ contienen el mismo número de características. Sin embargo, debido a que los grafos $G_1, G_2,$ y G_3 están asociados a otros contextos canónicos, los conjuntos de características $\mathcal{F}_1, \mathcal{F}_2,$ y $, \mathcal{F}_3$ contienen características

que difieren de las características en el conjunto $\mathcal{F}_0$. Por ejemplo, el grafo G_3 está asociado al contexto $(x_a^0 \wedge x_b^0 \wedge x_c^1)$, resultando en el conjunto $\mathcal{F}_3$, cuyas características están definidas sobre los estados $(x_a^0 \wedge x_c^1)$ y $(x_b^0 \wedge x_c^1)$.

Por otro lado, los grafos G_4, G_5, G_6 y G_7 presentan una topología diferente a la topología de los grafos G_0, G_1, G_2 y G_3. Más concretamente, los grafos G_4, G_5, G_6 y G_7 son un grafo completo. Un grafo completo se caracteriza por tener un único clique máximo, en nuestro caso, $\{\{a, b, c\}\}$. En consecuencia, cada uno de los conjuntos $\mathcal{F}_4, \ldots, \mathcal{F}_7$ contendrá una única característica.

Una vez inducido un conjunto de características desde cada grafo canónico, el método INDUCIR construye un conjunto de características unitarias. Para esto, para cada una de las variables $X_a \in X$, INDUCIR construye una característica por cada estado en $val(a)$. Debido a que las variables son binarias, $2n$ características unitarias son construidas. Estas características son listadas a continuación:

- (x_a^0),
- (x_b^0),
- (x_c^0),
- (x_a^1),
- (x_b^1),
- (x_c^1).

Finalmente, el método INDUCIR retorna un conjunto $\mathcal{F}$ que contiene todas las características mostradas hasta el momento. Más precisamente, el conjunto $\mathcal{F}$ retornado por INDUCIR es mostrado en la siguiente figura.

1. $(x_a^0 \wedge x_b^0 \wedge x_c^0)$;
2. $(x_a^1 \wedge x_b^0 \wedge x_c^0)$;
3. $(x_a^0 \wedge x_b^1 \wedge x_c^0)$;
4. $(x_a^1 \wedge x_b^1 \wedge x_c^0)$;
5. $(x_a^0 \wedge x_c^1)$;
6. $(x_a^1 \wedge x_c^1)$;
7. $(x_b^0 \wedge x_c^1)$;
8. $(x_b^1 \wedge x_c^1)$;
9. (x_a^0);
10. (x_a^1);
11. (x_b^0);
12. (x_b^1);
13. (x_c^0);
14. (x_c^1).

Figura 4-3: Conjunto $\mathcal{F}$ de características inducido desde el conjunto $\mathcal{G}$.

Para verificar que el conjunto $\mathcal{F}$ inducido por INDUCIR codifica las mismas independencias presentes en el conjunto $\mathcal{G}$, podemos usar la misma estrategia discutida en el Ejemplo 3.4. Esta estrategia nos permite verificar la presencia de independencias

en un conjunto de características de la siguiente manera. Por ejemplo, si inducimos un grafo G desde el subconjunto de características $\mathcal{F}[x_c^1]$ (el subconjunto de características en $\mathcal{F}$ que satisface el estado x_c^1), entonces obtenemos un grafo similar al mostrado en la Figura 4-4.

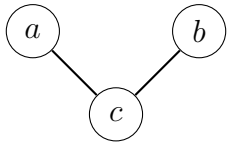


Figura 4-4: Grafo G inducido desde el conjunto $\mathcal{F}[x_c^1] \subseteq \mathcal{F}$.

En el grafo G de la Figura 4-4 tenemos que $\text{sep}_G(a, b; c)$, es decir, dicho grafo codifica el supuesto $I(X_a \perp X_b \mid X_c)$. Sin embargo, dado que el grafo es generado desde un conjunto de características donde la variable X_c únicamente toma el estado x_c^1, entonces el supuesto anterior corresponde al supuesto $I(X_a \perp X_b \mid x_c^1)$. Ahora, por la Ecuación (2.9), el supuesto anterior puede ser expresado de la siguiente manera

$$I(X_a \perp X_b \mid x_c^1) = I(x_a^0 \perp x_b^0 \mid x_c^1) \wedge I(x_a^1 \perp x_b^0 \mid x_c^1) \wedge I(x_a^0 \perp x_b^1 \mid x_c^1) \wedge I(x_a^1 \perp x_b^1 \mid x_c^1).$$

Como el supuesto $I(X_a \perp X_b \mid x_c^1)$ es verdadero, entonces, por la ecuación de arriba, cada una de sus independencias instanciadas también es verdadera. Precisamente, las independencias instanciadas mostradas arriba se corresponde con las independencias codificadas por los grafos $(G_3, x_a^0 \wedge x_b^0 \wedge x_c^1)$, $(G_2, x_a^1 \wedge x_b^0 \wedge x_c^1)$, $(G_1, x_a^0 \wedge x_b^1 \wedge x_c^1)$, y $(G_0, x_a^1 \wedge x_b^1 \wedge x_c^1)$ mostrados en la Figura 4-1. Por lo tanto, el conjunto $\mathcal{F}$ de características codifica el mismo conjunto de independencias que el conjunto $\mathcal{G}$.

□

4.5.2. Advertencias sobre la inducción de características

En esta sección mostraremos que el conjunto $\mathcal{F}$ de características inducidas por el método INDUCIR desde un conjunto $\mathcal{G}$ puede codificar errores del tipo I. Para mostrar esto, primero mostraremos que las características inducidas desde cada grafo

$G^k \in \mathcal{G}$, a través del método InducirDesdeGrafo, no introducen errores del tipo I. Utilizando este resultado, discutiremos la causa de por qué el método Inducir introduce errores del tipo I.

La siguiente proposición muestra que el método InducirDesdeGrafo no puede inducir características que codifiquen errores del tipo I, es decir, independencias que no se encuentran codificadas en un grafo canónico.

Proposición 4.2. Sea G^k un grafo canónico. Sea $\mathcal{F}$ un conjunto de características. Si $\mathcal{F}$ es obtenido por InducirDesdeGrafo desde el grafo G^k, entonces $\mathcal{F}$ no puede codificar independencias no codificadas por G^k.

Demostración. La demostración es realizada por contradicción.

- Supongamos que el conjunto $\mathcal{F}$ codifica una independencia no presente en G^k.
- Para esto último, pensemos en el caso más general posible. Según el Teorema 3.2, cualquier independencia instanciada puede ser expresada como un conjunto de independencias en parejas.
- Una independencia en parejas tiene la siguiente forma: $I(x_a \perp x_b \mid x_{V\setminus\{a,b\}})$.
- Por lo tanto, si $\mathcal{F}$ codifica una independencia instanciada no presente en G^k, entonces $\mathcal{F}$ debe codificar al menos una independencia en pareja no codificada por G^k.
- En base a esto, digamos que el conjunto $\mathcal{F}$ codifica la independencia $I(x_a^k \perp x_b^k \mid x_{V\setminus\{a,b\}}^k)$, la cual no está presente en G^k.
- Si $I(x_a^k \perp x_b^k \mid x_{V\setminus\{a,b\}}^k)$ no esté presente en G^k, entonces, por el Teorema 3.3, el grafo G^k tiene que presentar la arista (a, b).
- Si G^k presenta la arista (a, b), entonces debe existir al menos un clique $C \in \mathcal{C}(G^k)$ tal que $a, b \in C$. De otro modo, la arista (a, b) no estaría presente en G^k.

- Por otro lado, según el procedimiento introducido en la Sección 2.2.1, el conjunto $\mathcal{F}$ codifica la independencia $I(x_a^k \perp x_b^k \mid x_{V\setminus\{a,b\}}^k)$, si el grafo G inducido desde $\mathcal{F}$ no contiene la arista (a, b).

- Para que el grafo G inducido desde $\mathcal{F}$ no presente la arista (a, b), no debe existir ninguna característica $f_C \in \mathcal{F}$, donde $a, b \in C$.

- Cada una de las características en el conjunto $\mathcal{F}$ son obtenidas por el método InducirDesdeGrafo desde G^k.

- Según el método InducirDesdeGrafo, el conjunto $\mathcal{F}$ debe contener una característica por cada clique $C \in \mathcal{C}(G^k)$, es decir, debe existir una característica $f_C \in \mathcal{F}$, donde $a, b \in C$; lo cual es una contradicción.

□

De este modo, las características inducidas desde cada grafo $G^k \in \mathcal{G}$ no contienen errores del tipo I. Sin embargo, una vez inducidas las características desde cada grafo canónico, el método Inducir introduce un conjunto de características atómicas. A continuación, analizaremos más en detalle la razón y las consecuencias de agregar dichas características atómicas.

Hablando en términos generales, las características atómicas son agregadas por el método Inducir como una forma de responder a la siguiente pregunta: *¿cómo debe interpretarse todo aquel grafo canónico que no pertenece al conjunto $\mathcal{G}$?* O, equivalentemente, ¿cómo es la topología de todos aquellos grafos canónicos no construidos? Existen varias respuestas para esta pregunta, concretamente, una respuesta posible por cada posible topología. Es decir, la respuesta podría ser obtenida aprendiendo los restantes grafos canónicos. Sin embargo, como comentamos en la Sección 4.3.1, decidimos únicamente aprender los grafos canónicos para un determinado conjunto de contextos. Segundo, la inducción de características no está diseñada para realizar aprendizaje de estructuras, sino, más bien, está diseñada para transformar la representación de una estructura, es decir, obtener un conjunto de características desde un conjunto de grafos canónicos.

De este modo, para responder a la pregunta formulada anteriormente, nos resta asumir cierta topología en los restantes grafos canónicos. Como discutimos en la Sección 2.3, el espacio de estructuras se encuentra organizado a través de una relación *más-específico-que*. De este modo, en dicho espacio podemos encontrar dos estructuras extremas: la estructura completa y la estructura vacía. Utilizaremos esta idea para responder a nuestra pregunta. Así, una respuesta sería interpretar los restantes grafos canónicos como grafos completos. Al interpretarlos como grafos completos, nos aseguramos de no introducir ninguna otra independencia instanciada más allá de las codificadas por el conjunto $\mathcal{G}$. La segunda respuesta consiste en interpretar los restantes grafos canónicos como grafos vacíos. Al interpretarlos como grafos vacíos, nos aseguramos de no introducir ninguna dependencia instanciada más allá de las codificadas por el conjunto $\mathcal{G}$.

Cada una de estas respuestas tiene sus ventajas y desventajas. Concretamente, asumir grafos completos introduce errores del tipo II, mientras que asumir grafos vacíos introduce errores del tipo I. Como analizamos en el Capítulo 2, la presencia en la estructura de cada uno de estos errores impacta de diferente manera en la distribución construida a partir de dicha estructura. Sin embargo, aquí nos ocuparemos exclusivamente del impacto que ocasiona asumir un grafo completo o un grafo vacío únicamente dentro del método INDUCIR. Analicemos cada caso por separado.

Antes que nada, supongamos que $|\mathcal{G}| = K$, de este modo, asumiendo variables binarias, existen $2^n - K$ contextos o grafos canónicos que no están en $\mathcal{G}$. Para cada uno de estos contextos, el método INDUCIR debe asumir un grafo. Si el grafo asumido es un grafo completo, el método INDUCIR debería generar $2^n - K$ características, es decir, una característica por el único clique presente en cada uno de los $2^n - K$ grafos canónicos completos. Por otro lado, si el grafo asumido es vacío, entonces el método INDUCIR debe generar $n(2^n - K)$ características, es decir, una características por cada uno de los n cliques en los $2^n - K$ grafos canónicos vacíos. Sin embargo, en este último caso, las características son atómicas. Por lo tanto, en el peor caso, el conjunto de $n(2^n - K)$ características no puede ser nunca mayor a $2n$, debido a que, para n variables binarias, hay $2n$ características atómicas como máximo.

En base a nuestra anterior discusión, ahora queda claro por qué el método INDUCIR agrega las características atómicas al conjunto $\mathcal{F}$, es decir, el método INDUCIR esta asumiendo que todo grafo canónico que no pertenece al conjunto $\mathcal{G}$ es vacío. Asumir lo contrario, implicaría generar un número exponencial de características en el peor caso, es decir, $2^n - K$ características. Es por esta última razón que el método INDUCIR asume grafos vacíos, lo cual tiene como desventaja que puede introducir errores del tipo I. Como veremos en la próxima sección, más allá de los posibles errores del tipo I introducidos por el método INDUCIR, nuestro enfoque logra construir estructuras mucho más precisas que la mayoría de los algoritmos de aprendizaje de estructura evaluados.

4.6. Resumen

En este capítulo presentamos un nuevo enfoque para aprender la estructura de una distribución desde un conjunto de datos. Como principal característica, este enfoque utiliza un conjunto de grafos canónicos como representación de la estructura.

Al representar una estructura como un conjunto de grafos canónicos, un grafo canónico puede codificar únicamente un conjunto de independencias instanciadas. Debido a que cualquier independencia condicional e independencia específica del contexto puede ser expresada como un conjunto de independencias instanciadas, un conjunto de grafos canónicos tiene la capacidad de codificar independencias condicionales e independencias específicas del contexto.

A grandes rasgos, para aprender un conjunto de grafos canónicos, el enfoque propuesto consiste en dos pasos. El primer paso consiste en la definición de un conjunto de contextos canónicos. En base al conjunto de contextos canónicos, el segundo paso consiste en aprender un grafo canónico por cada contexto canónico. Para aprender un grafo canónico, mostramos que es posible utilizar cualquier algoritmo basado en restricciones.

Los algoritmos basados en restricciones están diseñados para aprender un grafo no canónico, es decir, un grafo no dirigido. Para aprender un grafo, estos algorit-

mos utilizan una prueba estadística de independencia. Dicha prueba es utilizada para determinar la presencia de una independencia en un conjunto de datos. Cuando una independencia es encontrada, un algoritmo basado en restricciones codifica dicha independencia en el grafo, modificando su topología. Por ejemplo, agregando o removiendo ciertas aristas.

Una prueba estadística de independencia está diseñada para determinar la presencia de independencias condicionales. Debido a que una independencia instanciada puede ser expresada como una independencia condicional, mostramos un método para identificar la presencia de independencias instanciadas basado en una prueba estadística de independencia.

Por lo tanto, utilizando este método para identificar independencias instanciadas, cualquier algoritmo basado en restricciones puede ser utilizado para aprender un grafo canónico. Para mostrar esto, presentamos dos algoritmos concretos. El algoritmo CSPC y el algoritmo CSGS, los cuales utilizan los algoritmos PC y GS, dos algoritmos bien conocidos del enfoque basado en restricciones.

También mostramos que nuestro enfoque para aprender un conjunto de grafos canónicos puede ser ejecutado en forma paralela, lo cual permite tomar ventaja de una arquitectura con múltiples computadoras. Como resultado, el número de operaciones realizado por nuestro enfoque para aprender un conjunto de grafos canónicos puede ser idealmente reducido al mismo número de operaciones realizado por un algoritmo basado en restricciones para construir un grafo.

Debido a que nuestro enfoque aprende la estructura de una distribución usando una representación novedosa (un conjunto de grafos canónicos), las estructuras aprendidas por nuestro enfoque no pueden ser utilizadas por paquetes de software bien conocidos. Por ejemplo, paquetes de software para aprendizaje de parámetros e inferencia. Por esta razón, presentamos un método para inducir un conjunto de características desde un conjunto de grafos canónicos. Debido a que un conjunto de características es una representación ampliamente usada en la práctica, este método de inducción de características permite que las estructuras aprendidas por nuestro enfoque puedan ser utilizadas con otros paquetes de software.

Capítulo 5

Estudio experimental

No es la más fuerte de las especies la que sobrevive, ni la más inteligente.
Es la más adaptable al cambio.

Charles Darwin.

Este capítulo está exclusivamente dedicado a: describir la experimentación que fue diseñada para evaluar empíricamente el rendimiento de nuestro enfoque en el problema de aprendizaje de estructuras de redes de Markov; presentar los resultados obtenidos en tal evaluación; y brindar un análisis de dichos resultados. Más concretamente, la experimentación tuvo como único objetivo evaluar el impacto en la calidad del aprendizaje de estructuras cuando las independencias específicas del contexto son explícitamente consideras durante el aprendizaje. Como hemos visto en el Capítulo 2, esta última proposición implica que tanto la representación de la estructura cómo la lógica del aprendizaje deben estar diseñadas tanto para codificar y aprender, respectivamente, independencias específicas del contexto. La ausencia de algunas de estas dos características, podría resultar en un algoritmo que aprende estructuras que no codifican correctamente las posibles independencias específicas del contexto de un problema.

Nuestra evaluación empírica consistió en comparar las estructuras aprendidas por nuestro enfoque, los algoritmos CSPC y CSGS presentados formalmente en el Capítulo 4, frente a las estructuras aprendidas por otros algoritmos de aprendizaje de

estructuras. Para estos últimos algoritmos, intentamos seleccionar una muestra lo más representativa posible. Para esto, seleccionamos algoritmos que son ampliamente considerados, por la comunidad de aprendizaje de estructuras, como el estado del arte en el problema de aprendizaje de estructuras. Los algoritmos seleccionados fueron tomados desde las dos clases de algoritmos de aprendizaje de estructuras (ver Capítulo 1), es decir, los algoritmos basados en restricciones y los algoritmos de estimación de densidad. Como mencionamos en el Capítulo 2, cada una de estas clases de algoritmos se caracterizan por representar la estructura de una manera particular. Los algoritmos basados en restricciones utilizan un grafo no dirigido como representación de la estructura, mientras que los algoritmos de estimación de densidad utilizan un conjunto de características. Como veremos en los resultados de este capítulo, esta diferencia en la representación de la estructura conduce a diferentes ventajas y desventajas, permitiéndonos extraer conclusiones sumamente interesantes con respecto al problema de aprendizaje de estructuras.

Para evaluar el rendimiento de un algoritmo, la metodología consistió en primero construir una red de Markov desde un conjunto de datos, lo cual, según lo discutido en el Capítulo 2, involucra dos pasos: aprender una estructura utilizando cada uno de los algoritmos propuestos; y luego, para cada estructura, construir una red de Markov a través de la estimación de sus parámetros. Una vez obtenida una red de Markov, hicimos dos tipos de mediciones: i) medimos la correctitud de su estructura (la cual fue aprendida por medio de un algoritmo de aprendizaje de estructuras); y ii) medimos la precisión de dicha red de Markov. En el primer caso, analizamos el número de errores presentes en la estructura. Como mencionamos en la Sección 2.1.1, una estructura puede tener dos tipos de errores: errores del tipo I, la estructura codifica independencias que no se cumplen en la distribución subyacente; y errores del tipo II, la estructura codifica dependencias que no se cumplen en la distribución subyacente. Por el otro lado, en el segundo caso, el análisis de la precisión de una red de Markov aporta información sumamente útil sobre la correctitud de su estructura. Como se mencionó en la Sección 2.2, la precisión de una red de Markov se encuentra íntimamente sujeta a los errores presentes en su estructura. Recordando, la presen-

cia de un error del tipo I resulta en una factorización incorrecta de la distribución, produciendo una distribución imprecisa. Por otro lado, la presencia de un error del tipo II resulta en una factorización correcta de la distribución, pero esta distribución presenta parámetros espurios. Durante la estimación de parámetros, los parámetros espurios son propensos a producir errores (e.g., overfitting o sobreestimación) ante la presencia de pequeñas cantidades de datos (una situación bastante usual en la mayoría de los problemas de la práctica). De esta manera, a través del análisis de estas dos mediciones, podemos evaluar la calidad del aprendizaje de un determinado algoritmo de aprendizaje, permitiéndonos comparar nuestro enfoque frente a los restantes algoritmos.

Por otro lado, los conjuntos de datos utilizados para construir las redes de Markov fueron generados desde distribuciones de probabilidad pertenecientes a dos escenarios diferentes: un *escenario sintético* y un *escenario del mundo real*. La diferencia fundamental entre ambos escenarios es que, en el escenario sintético, la distribución subyacente del problema es conocida; mientras que, en el escenario del mundo real, los conjuntos de datos fueron generados desde distribuciones totalmente desconocidas. Para ser más precisos, en el escenario sintético, la distribución subyacente fue definida manualmente por nosotros, es decir, entre otras cosas, la estructura de dicha distribución fue intencionadamente definida para presentar características afines a los objetivos de nuestra evaluación. Más concretamente, las estructuras de las distribuciones del escenario sintético presentan independencias específicas del contexto. En contraste, el escenario del mundo real intenta simular el escenario típico que surge en la práctica cuando se intenta aprender una distribución de probabilidad desde un conjunto de datos observados desde un fenómeno real, el cual se asume estar regido por una distribución de probabilidad desconocida.

Debido a la naturaleza de cada escenario, las medidas propuestas para evaluar una red de Markov, y por ende el rendimiento de un algoritmo de aprendizaje de estructuras, se encuentran fuertemente entrelazadas con las particularidades de cada escenario. Por ejemplo, en el escenario sintético, la correctitud de una estructura aprendida por un algoritmo puede ser directamente determinada al comparar dicha

estructura con la estructura de la distribución subyacente. En contraste, en el escenario del mundo real, la correctitud de una estructura aprendida no puede ser medida del mismo modo, debido a que la distribución subyacente del problema es desconocida y, en consecuencia, su estructura. De este modo, para tener una estimación de la correctitud de una estructura aprendida por un algoritmo en el escenario del mundo real, medimos la precisión de la red de Markov estimada desde dicha estructura. Usualmente esto es llevado a cabo en la práctica midiendo que tan precisa es una red de Markov en tareas de predicción. Para realizar predicciones con una red de Markov, necesitamos ejecutar métodos de inferencia sobre dicha red. Como comentamos en la Sección 2.6, la precisión de los resultados alcanzados por cualquier método de inferencia depende directamente de la topología de la estructura de una red de Markov. Consecuentemente, si la estructura de una red de Markov presenta errores (e.g., errores del tipo I o del tipo II), al ejecutar inferencia sobre dicha red, los resultados obtenidos serán imprecisos, afectando así la habilidad predictiva de una red de Markov. Esta última observación puede ser utilizada como un criterio para seleccionar la red de Markov más precisa dentro de un conjunto de redes de Markov.

Los resultados obtenidos en nuestra experimentación aportan una fuerte evidencia a favor de que nuestro enfoque para representar y aprender la estructura de una red de Markov permite obtener estructuras más correctas frente a gran parte de los algoritmos evaluados. En comparación a los algoritmos basados en restricciones, en el escenario sintético, nuestros algoritmos aprendieron estructuras más correctas frente a todos los algoritmos evaluados. Adicionalmente, en el escenario del mundo real, las distribuciones estimadas, desde las estructuras aprendidas por nuestros algoritmos, resultaron ser significativamente más precisas que aquellas obtenidas por todos los algoritmos basados en restricciones. Por otro lado, en comparación a los algoritmos de estimación de densidad, en el escenario sintético, nuestros algoritmos aprendieron estructuras más correctas frente a todos los algoritmos. Sin embargo, en el escenario en el mundo real, las distribuciones obtenidas a través de nuestros algoritmos únicamente resultaron ser significativamente más precisas frente a la mitad de los algoritmos evaluados.

En términos generales, estos resultados son totalmente esperables. Por un lado, nuestros algoritmos obtuvieron estructuras más correctas que los algoritmos basados en restricciones. Como comentamos extensamente en el Capítulo 3, esto es debido principalmente a las ventajas que presenta nuestra representación de la estructura. En otras palabras, la estructura de un modelo canónico puede ser representada utilizando un conjunto de grafos canónicos, donde cada grafo canónico codifica un determinado número de independencias. De este modo, a través de un conjunto de grafos canónicos, es posible codificar un rango más amplio de independencias en comparación a utilizar un único grafo no dirigido. En términos prácticos, nuestros algoritmos puede aprender un conjunto más amplio de estructuras en comparación a los algoritmos basados en restricciones. En base a esto, también es importante remarcar que *nuestros algoritmos son esencialmente algoritmos basados en restricciones*. Como vimos en el Capítulo 4, CSPC y CSGS pueden pensarse como un meta-algoritmo que utiliza como algoritmo base un algoritmo basado en restricciones. En otras palabras, la mejora en precisión alcanzada por nuestros algoritmos es principalmente debida por la representación de la estructura utilizada, más que por la lógica del algoritmo. Por así decirlo, nuestros algoritmos no son más "fuertes" o más "inteligentes" que los algoritmos basados en restricciones, sino en cambio, nuestros algoritmos utilizan una representación de la estructura que puede "adaptarse mejor" a las diferentes regularidades que pueden surgir desde los datos.

Por otro lado, nuestros algoritmos resultaron ser semejantes a los algoritmos de estimación de densidad debido, en cierta medida, a que dichos algoritmos también utilizan una representación de la estructura que les permite capturar un mayor número de regularidades en los datos. A tal punto, que en términos de precisión, nuestros algoritmos no logran superar significativamente a los algoritmos de estimación de densidad. Sin embargo, como veremos más en detalle en la Sección 5.4.1, las estructuras aprendidas por los algoritmos de estimación de densidad no son tan interpretables como las estructuras aprendidas por nuestros algoritmos. Aquí, por interpretabilidad, nos referimos al siguiente hecho, al analizar una estructura uno puede determinar el valor de verdad de un supuesto de independencia utilizando el concepto de separabi-

lidad entre nodos (ver Capítulo 2).

El resto de este capítulo se estructura de la siguiente manera. La Sección 5.1 presenta los diferentes conjuntos de datos utilizados durante la evaluación. La Sección 5.2 enumera los algoritmos de aprendizaje de estructuras seleccionados para la experimentación. La Sección 5.3 describe en detalle los dos escenarios que tuvimos en cuenta en la experimentación: el escenario sintético y el escenario del mundo real. La Sección 5.4 muestra los resultados obtenidos en el escenario sintético. Finalmente, la Sección 5.5 muestra los resultados obtenidos en el escenario del mundo real.

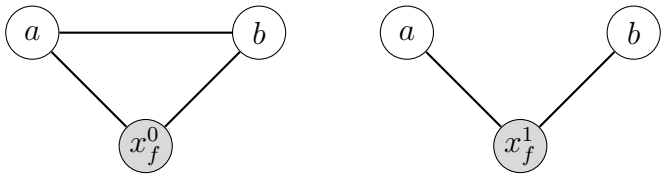
5.1. Conjuntos de datos

Esta sección describe en detalle los conjuntos de datos, utilizados en cada uno de los escenarios propuestos, para aprender redes de Markov. Para el escenario sintético, describiremos principalmente cómo fueron definidas las distribuciones de probabilidad, las cuales fueron muestreadas para generar los conjuntos de datos. Para el escenario real, describiremos algunas de las características generales de los conjuntos de datos utilizados.

5.1.1. Conjuntos de datos sintéticos

Esta sección describe cómo fueron generados los conjuntos de datos utilizados en el escenario sintético. En términos muy sencillos, la generación de dichos conjuntos puede dividirse en dos partes: primero, la definición de una red de Markov (la cual representa una distribución de probabilidad); y, segundo, la generación de un conjunto de datos a través del muestreo de dicha red de Markov. En lo que sigue, profundizaremos en cada parte.

La distribución de probabilidad que definimos es similar a la presentada en el Ejemplo 3.1, cuya estructura es mostrada en la Figura 5-1. Esta distribución está definida por $n = |X|$ variables aleatorias binarias, y *su estructura está especialmente caracterizada por ser asimétrica, es decir, ésta codifica independencias específicas del contexto*. Los supuestos de independencia codificados tienen la forma $I(X_a \perp X_b \mid$



(a) Grafo instanciado por el contexto $X_f = 0$, denotado por $G(X_f = 0)$.

(b) Grafo instanciado por el contexto $X_f = 1$, denotado por $G(X_f = 1)$.

Figura 5-1: Representación gráfica de una estructura subyacente con $n = 3$ en el escenario sintético.

$X_f = 1$) para todo par $a, b \in V \setminus \{ f \}$; mientras que, cuando $X_f = 0$, todo par de variables se vuelve dependiente. Como se puede apreciar en la Figura 5-1, la estructura puede ser representada gráficamente por dos grafos instanciados, uno por cada estado de la *variable bandera* X_f. Cuando $X_f = 0$, la estructura puede ser representada como un grafo completo, el cual es denotado por $G(X_f = 0)$. En cambio, cuando $X_f = 1$, la estructura es representada como un *grafo estrella*[1], el cual es denotado por $G(X_f = 1)$.

A pesar de la simplicidad de esta estructura, *cualquier algoritmo basado en restricciones está imposibilitado en construirla correctamente*. Esto se debe a que, al usar un único grafo como representación de una estructura, dicha representación no tiene la suficiente flexibilidad para capturar correctamente las independencias específicas del contexto de la distribución propuesta (ver Capítulo 3). Por otro lado, como el grafo $G(X_f = 0)$ es completo, es decir el grado máximo de cualquier nodo es n, este problema resulta desafiante para cualquier algoritmo basado en restricciones, debido a que éstos son usualmente evaluados en problemas donde el grado máximo de los nodos rara vez supera el valor 6 u 8 [38, 51, 81].

Para definir una distribución $p(X)$ cuya estructura codifique las independencias específicas del contexto mostradas en la Figura 5-1, definimos una red de Markov sobre un conjunto X de variables binarias tal que cumpla con la siguiente factorización:

[1]Decimos que $G(X_f = 1)$ es un grafo estrella porque el nodo x_f^1 puede verse como un nodo central conectado a cada uno de los restantes nodos $V \setminus \{f\}$.

$$p(X) \propto \begin{cases} \prod_{a,b \in V \setminus \{f\}} \phi_{a,b}(X_a, X_b, X_f), & \text{si } X_f = 0; \quad (5.1a) \\ \prod_{a \in V \setminus \{f\}} \phi_a(X_a, X_f), & \text{si } X_f = 1. \quad (5.1b) \end{cases}$$

Debido a la presencia de independencias específicas del contexto, los potenciales de dicha factorización pueden agruparse en dos conjuntos. El primer conjunto, mostrado en la Ecuación (5.1a), se caracteriza por tener tres variables en su alcance. El otro conjunto, mostrado en la Ecuación (5.1b), se caracteriza por tener sólo dos variables en su alcance. Como el conjunto X está únicamente formado por variables discretas, cada potencial mostrado en la Ecuación (5.1) puede representarse como una tabla de parámetros, la cual asocia para cada combinación de los estados de sus variables un parámetro. Por ejemplo, cada potencial en la Ecuación (5.1a) pueden ser representado como una tabla con 2^3 parámetros; mientras que cada potencial en la Ecuación (5.1b) puede ser representado por una tabla de 2^2 parámetros.

A continuación, describiremos en detalle cómo los parámetros de cada potencial fueron definidos. Este punto resulta crucial debido a que una definición incorrecta de parámetros puede conducir a una distribución $p(X)$ que no codifique correctamente la estructura mostrada en la Figura 5-1, ocasionando que $p(X)$ no codifique independencias específicas del contexto. Como veremos más abajo, los parámetros involucrados en los potenciales de la Ecuación (5.1b) se encuentran también involucrados en los potenciales de la Ecuación (5.1a). Por esta razón, comenzaremos describiendo los parámetros presentes en la Ecuación (5.1b), y luego seguiremos con los parámetros en la Ecuación (5.1a).

Para definir los parámetros correspondientes a los potenciales mostrados en la Ecuación (5.1b), debemos definir una relación de dependencia entre cada variable X_a y la variable bandera X_f. En términos gráficos, la dependencia entre X_a y X_f implica la presencia de una arista entre los nodos a y x_f^1 en el grafo instanciado $G(X_f = 1)$ (Figura 5-1b). Una forma usualmente utilizada para definir una relación de dependencia entre variables es utilizar el logaritmo de la *razón de oportunidades*

(o *log-odds ratio*) como función paramétrica (o potencial) entre las variables.

La justificación de utilizar esta función como potencial es que una definición arbitraria del mismo (e.g., seleccionar arbitrariamente sus parámetros) podría resultar en dos problemas íntimamente relacionados: i) la dependencia entre X_a y X_f podría no cumplirse (ambas variables son independientes); o ii) la dependencia entre dichas variables podría ser muy débil. Una dependencia débil implica que dicha dependencia sólo puede ser inferida cuando hay una gran número de datos. En otras palabras, cuando el número de datos es bajo, ambas variables podrían resultar independientes. Estas situaciones dificultarían nuestros análisis de las estructuras obtenidas por un algoritmo de aprendizaje de estructuras. Por ejemplo, si una dependencia no está presente en una estructura aprendida ¿esto se debe a un problema en la lógica (o la representación de la estructura) del algoritmo? O, más bien ¿se debe a que la dependencia es débil y no hay suficientes datos para que un algoritmo pueda inferir dicha dependencia? Por esta razón, por una cuestión de correctitud y eficiencia muestreal, elegimos valores de parámetros que resulten en dependencias fuertes, i.e., dicha dependencia pueda ser inferida desde un número relativamente bajo de datos.

X_a	X_f	$\phi_a(X_a, X_f)$
0	0	θ_0
1	0	θ_1
0	1	θ_2
1	1	θ_3

Cuadro 5.1: Potencial $\phi_a(X_a, X_f)$.

Tal como se muestra en [34, 59, 81], al satisfacer la razón de oportunidades, una dependencia es definida entre un par de variables. Para que los parámetros involucrados en el potencial $\phi_a(X_a, X_f)$ satisfagan la razón de oportunidades, estos deben cumplir la siguiente ecuación:

$$\varepsilon = \log\left(\frac{\theta_0 \times \theta_3}{\theta_1 \times \theta_2}\right), \tag{5.2}$$

donde ε describe la fuerza de la dependencia entre X_a y X_f; y θ_0, θ_1, θ_2, y θ_3 son los parámetros para cada combinación de estados de las variables X_a y X_f (ver el

Cuadro 5.1). En la Ecuación (5.2), $\varepsilon = 0$ si y sólo si X_a y X_f son independientes, es decir, cuando $\theta_0 \times \theta_3 = \theta_1 \times \theta_2$. De este modo, para definir una dependencia entre X_a y X_f hay que definir los parámetros $\theta_i, i = \{\ 0, 1, 2, 3\ \}$ de tal modo que $\varepsilon \neq 0$. Como se muestra en [34, 59, 81], si los parámetros son definidos tal que $\varepsilon = 1$, entonces se logra una dependencia fuerte entre X_a y X_f.

Para simplificar la definición de los parámetros de un potencial, y sin pérdida de generalidad, asumiendo que $\theta_0 = \theta_3$ y $\theta_1 = \theta_2$, entonces la Ecuación (5.2) puede ser simplificada de la siguiente manera:

$$\varepsilon = \log\left(\frac{\theta_0^2}{\theta_1^2}\right). \tag{5.3}$$

En otras palabras, para definir una dependencia entre X_a y X_f sólo tenemos que definir dos parámetros, θ_0 y θ_1, tal que la Ecuación (5.3) sea igual a 1.

Una manera de lograr esto último se basa en los siguientes dos puntos. Primero, uno de los parámetros es definido aleatoriamente, supongamos θ_1. Segundo, una vez que θ_1 tiene un valor asignado, el valor de θ_0 es seleccionado para satisfacer $\varepsilon = 1$ en la Ecuación (5.3). Esto último se logra resolviendo la siguiente ecuación:

$$\theta_0 = \sqrt{\exp(1 + \log(\theta_1^2))} = \textsc{Log-Odds}(\theta_1). \tag{5.4}$$

Sin embargo, una selección arbitraria del parámetro θ_1 en cada potencial, puede conducir a grandes diferencias entre los parámetros de diferentes potenciales. Como consecuencia, esto podría resultar en que ciertas dependencias predominen frente a otras, es decir, ciertas dependencias pueden ser débiles.

Para evitar el problema anterior, el valor del parámetro θ_1 fue aleatoriamente muestreado desde una distribución normal con media igual a 0.5 y desviación estándar igual a 0.001, denotada por $\mathcal{N}(0.5;0.001)$. Seleccionando los parámetros de esta manera, se evita grandes diferencias entre los parámetros de diferentes potenciales.

Por otro lado, para definir los parámetros de los potenciales mostrados en la Ecuación (5.1a), seguimos el siguiente procedimiento. Primero, es importante notar que los parámetros de cualquier potencial $\phi_{a,b}(X_a, X_b, X_f)$ pueden ser agrupados en

dos subconjuntos disjuntos: los parámetros que satisfacen el estado $X_f = 0$, lo cual denotaremos como $\phi_{a,b}(X_a, X_b, X_f = 0)$, y los parámetros que satisfacen el estado $X_f = 1$, denotado por $\phi_{a,b}(X_a, X_b, X_f = 1)$. En el primer conjunto de parámetros, las variables X_a y X_b son dependientes. En términos gráficos, el grafo instanciado $G(X_f = 0)$ presenta una arista entre los nodos a y b (Figura 5-1a). Para definir esta dependencia, los parámetros involucrados en el potencial $\phi_{a,b}(X_a, X_b, X_f = 0)$ fueron obtenidos siguiendo el mismo procedimiento para definir los parámetros de un factor $\phi_a(X_a, X_f)$.

En contraste, los parámetros del potencial $\phi_{a,b}(X_a, X_b, X_f = 1)$ deben cumplir la independencia específica del contexto $I(X_a \perp X_b \mid X_f = 1)$, es decir, las variables X_a y X_b son independientes dado el contexto $X_f = 1$. En términos gráficos, el grafo $G(X_f = 1)$ tiene que cumplir $\text{sep}_G(a, b; f)$ (Figura 5-1b). De esta manera, por la definición de independencia específica del contexto, el potencial $\phi_{a,b}(X_a, X_b, X_f = 1)$ debe cumplir la siguiente igualdad:

$$\phi_{a,b}(X_a, X_b, X_f = 1) = \phi_a(X_a, X_f = 1) \times \phi_b(X_b, X_f = 1).$$

En palabras, el potencial $\phi_{a,b}(X_a, X_b, X_f = 1)$ es el producto de los potenciales $\phi_a(X_a, X_f = 1)$ y $\phi_b(X_b, X_f = 1)$. Estos dos últimos potenciales se corresponden con los potenciales de la Ecuación (5.1b). Por ejemplo, el potencial $\phi_a(X_a, X_f = 1)$ se corresponden con los parámetros asociados al estado $X_f = 1$ en el potencial $\phi_a(X_a, X_f)$. En consecuencia, los parámetros del potencial $\phi_{a,b}(X_a, X_b, X_f = 1)$ son obtenidos multiplicando los parámetros de los potenciales $\phi_a(X_a, X_f = 1)$ y $\phi_b(X_b, X_f = 1)$.

Para resumir el procedimiento previamente comentado, con el cual se obtienen los parámetros de la distribución $p(X)$ mostrada en la Ecuación (5.1), presentamos un método denominado Generar-Red-De-Markov, el cual es mostrado en el Algoritmo 17. En términos generales, Generar-Red-De-Markov tiene como entrada un número n de variables y su salida es un conjunto Φ de potenciales. Internamente, Generar-Red-De-Markov selecciona una de las variables del dominio como variable bandera, X_f, y luego, para las restantes variables $X \setminus X_f$, define los dos

tipos de potenciales, primero definiendo el conjunto de potenciales mostrados en la Ecuación (5.1b) y luego los potenciales mostrados en la Ecuación (5.1a).

Algoritmo 17 Generador de redes de Markov con independencias contextuales

procedure GENERAR-RED-DE-MARKOV(n)
$V \leftarrow (1, \ldots, n)$
$f \leftarrow (n-1)$
$\Phi \leftarrow \emptyset$
for $a \in V \setminus \{ f \}$ **do**
$\theta_1 \sim \mathcal{N}(0.5;0.001)$
$\theta_0 \leftarrow$ LOG-ODDS(θ_1)

$\phi_a(X_a, X_f) \leftarrow$

X_a	X_f	$\phi_a(X_a, X_f)$
0	0	θ_0
1	0	θ_1
0	1	θ_1
1	1	θ_0

$\Phi \leftarrow \Phi \cup \{ \phi_a(X_a, X_f) \}$
for $a, b \in V \setminus \{ f \}$ **do**
$\theta_1 \sim \mathcal{N}(0.5;0.001)$
$\theta_0 \leftarrow$ LOG-ODDS(θ_1)

$\phi_{a,b}(X_a, X_b, X_f) \leftarrow$

X_a	X_b	X_f	$\phi_{a,b}(X_a, X_b, X_f)$
0	0	0	θ_0
1	0	0	θ_1
0	1	0	θ_1
1	1	0	θ_0
0	0	1	$\phi_a(X_a = 0, X_f = 1) \times \phi_b(X_b = 0, X_f = 1)$
1	0	1	$\phi_a(X_a = 1, X_f = 1) \times \phi_b(X_b = 0, X_f = 1)$
0	1	1	$\phi_a(X_a = 0, X_f = 1) \times \phi_b(X_b = 1, X_f = 1)$
1	1	1	$\phi_a(X_a = 1, X_f = 1) \times \phi_b(X_b = 1, X_f = 1)$

$\Phi \leftarrow \Phi \cup \{ \phi_{a,b}(X_a, X_b, X_f) \}$
return Φ

Por lo tanto, para obtener los conjuntos de datos del escenario sintético, GENERAR-RED-DE-MARKOV fue utilizado para generar 10 redes de Markov para cada $n \in \{ 6, 7, 8, 9 \}$. Para un n fijo, las 10 redes de Markov generadas se diferencian únicamente en sus parámetros (debido a que su definición se basa en el muestreo de una distribución normal), pero *sus estructuras son exactamente iguales a la estructura mostrada en la Figura 5-1.* Finalmente, para cada una de las redes de Markov generadas con GENERAR-RED-DE-MARKOV, un conjunto de datos es generado utilizando muestreo de Gibbs[2] (ver el Algoritmo 5). El muestreo de Gibbs fue confi-

[2]Puntualmente, el muestreo de Gibbs fue realizado utilizando Libra toolkit (http://libra.cs.uoregon.edu/) versión 0.5.0 [58].

gurado con 10 *cadenas* y un *burn-in* igual a 100. El número de muestras tomadas desde cada red fue: 100, 200, 500, 700, 1k, 2k, 5k, 7k, 10k, 20k, 50k, 70k, 100k. Todos los conjuntos de datos generados se encuentran públicamente disponibles en: `http://dharma.frm.utn.edu.ar/papers/ijait14`.

5.1.2. Conjunto de datos del mundo real

Los conjuntos de datos del mundo real han sido usado por varios trabajos recientes relacionados con el aprendizaje de estructuras de redes de Markov [54, 57, 36, 88]. El Cuadro 5.2 muestra los conjuntos de datos utilizados durante nuestra experimentación junto con sus características generales.

Conjunto D	n	$\|D\|$
NLTCS	16	21,574
MSNBC	17	388,434
Abalone	31	4,177
Wine	48	6,499
KDDCup 2000	65	234,954
Plants	69	23,215
Covertype	84	40,000
Audio	100	20,000
Netflix	100	20,000
Accidents	111	17,009
Adult	125	48,842
Retail	135	29387
Pumsb Star	163	16,349
DNA	180	3,186
MSWeb	294	37,711
WebKB	839	4,199
BBC	1,058	2,225
Ad	1,556	3,279

Cuadro 5.2: Conjuntos de datos de problemas del mundo real. La primera columna muestra el nombre del conjunto D de datos, la segunda columna muestra su número de variables, la tercer columna muestra el número de observaciones.

Todos los conjuntos de datos están definidos por variables binarias que varían de 16 a 1556, y la cantidad de datos varía de 2k a 291k. Cada conjunto D de datos fue dividido en tres conjuntos disjuntos: un *conjunto de entrenamiento*, denotado por D_E;

un *conjunto de validación*, denotado por D_V; y un *conjunto de prueba*, denotado por D_P. Las razones de esta división serán expuestas en la Sección 5.3. Cada uno de estos conjuntos de datos son muestras tomadas desde múltiples dominios. A continuación presentaremos un breve descripción para cada uno de los dominios.

NLTCS es un muestra de la *Encuesta Nacional de Cuidado a Largo Plazo* (*National Long Term Care Survey*) diseñada para estudiar los cambios en la salud y el estado funcional de adultos norteamericanos. Este conjunto consiste de variables binarias que miden las habilidades de un individuo para realizar diferentes actividades durante la vida diaria.

MSNBC son datos web anónimos que contienen información sobre si un usuario visitó un página (de nivel superior) en MSNBC durante una sesión. Cada variable está asociada a una página y es verdadera si un usuario visitó dicha página durante una sesión.

Abalone, Adult, Covertype y Wine son conjuntos de datos bien conocidos provenientes del *repositorio de aprendizaje de máquinas UCI* [52].

Plants consiste de diferentes tipos de plantas y dónde estás localizadas. Cada variable representa una localización y es verdadera si planta se encuentre allí.

Audio y Netflix son conjuntos de datos para problema de filtrado colaborativo. El conjunto Audio consiste de información sobre que tan frecuente un usuario escucha a un artista particular. Cada variable representa a un arista y es verdadera si un usuario escuchó a dicho artista. El conjunto Netflix es una muestra al azar del *conjunto de datos del desafío de Netflix*. Cada variable representa una película y es verdadera si un usuario calificó como buena a dicha película.

KDDCup2000 es un conjunto de datos que consiste de sesiones web tomados desde Gazelle.com, un minorista en línea. Cada variable está asociada a una de las categorías de producto en la página web y es verdadero cuando una sesión visitó cierta categoría.

El conjunto MSWeb son datos web de 294 áreas (Vroots) del sitio web de Microsoft tomados durante una semana de febrero en 1998.

WebKB es un conjunto datos relacionado con documentos de textos. Cada variable representa una palabra y es verdadera si esta aparece en un cuerpo de 50 documentos.

El conjunto DNA representa secuencias de ADN, donde cada variable está asociada a un nucleótido (A, G, T, C), el cual es representado como una variable binaria indicador de longitud 3.

Accidents contiene una muestra tomada desde datos (anónimos) de accidentes de tráficos del *National Institute of Statistics* (NIS) de la región de Flanders (Bélgica) durante el período 1991-2000. Cada variable representa circunstancias en las cuales un accidente ocurrió.

El conjunto BBC consiste de varios documentos del sitio web de noticias BBC durante el período 2004-2005. Cada variable está asociado a una palabra y es verdadera si ésta aparece en algún documento.

El conjunto Pumsb star consiste de datos de censo sobre población. El conjunto Retail consiste de datos sobre carros de compras de una tienda de compras en Bélgica.

Finalmente, el conjunto Ad representa anuncios publicitarios en páginas de Internet. Cada variable está asociada a la localización del anuncio y al texto que contiene.

5.2. Aprendizaje de redes de Markov

Utilizando los conjuntos de datos descriptos en la sección anterior, aprendimos la estructura de una red de Markov utilizando diferentes algoritmos de aprendizaje de estructuras con diferentes parámetros de configuración. Usando la estructura previamente aprendida, obtuvimos una red de Markov estimando sus parámetros. Las próximas dos secciones dan detalles sobre los algoritmos de aprendizaje de estructuras y estimación de parámetros utilizados.

5.2.1. Aprendizaje de estructuras

Para aprender la estructura de una red de Markov, utilizamos los algoritmos CSPC y CSGS junto a 8 algoritmos de aprendizaje de estructuras del estado del arte: 4 algoritmos basados en restricciones y 4 algoritmos de estimación de densidades. El nombre de los algoritmos utilizados, junto con las referencias a sus publicaciones, son mostrados en el Cuadro 5.3. Vale remarcar que todos los algoritmos propuestos se

encuentran públicamente disponibles[3].

Algoritmos basados en restricciones	Algoritmos de estimación de densidades
PC [87]	BLM [18]
GSMN [13]	L1 [78]
HHC-MN [3, 81]	GSSL [89]
IBMAP-HC [81]	DTSL [56]

Cuadro 5.3: Algoritmos de aprendizaje de estructuras utilizados en nuestra experimentación.

Para aprender la estructura, todos los algoritmos basados en restricciones construyen un grafo desde un conjunto de datos. Para construir el grafo, dichos algoritmos utilizan una prueba estadística. Para alcanzar una comparación justa, utilizamos la misma prueba estadística para todos los algoritmos: la prueba χ^2 de Pearson con un nivel de significancia igual a 0.05. Sin embargo, IBMAP-HC solamente funciona con la *prueba de independencia Bayesiana* [61]. Como es recomendado en [81], la prueba de independencia Bayesiana utiliza un *valor de corte* igual a 0.5.

Por otro lado, los algoritmos de estimación de densidad construyen un conjunto de características desde un conjunto de datos. Excepto por DTSL, todos los algoritmos de estimación de densidad seleccionados aprenden conjuntos de características positivas[4]. Debido a que los algoritmos de estimación de densidad son propensos a errores de sobreajustes, su rendimiento puede ser ajustada a través de diferentes parámetros de configuración. Por ello, para cada algoritmo, replicamos los parámetros de configuración recomendados. Para ser más preciso, para BLM, usamos 1, 2, 5, 10, 20, 25, 50, 100 y 200 *observaciones más cercanas*; *frecuencia de características aceptadas* de 1, 5, 10 y 25; y *desviación estándar* de 0.1, 0.5 y 1. Para L1, empleamos el mismo procedimiento usado por [57], utilizando *priors* igual a 0.1, 0.5, 1, 2, 5, 10 15 y 20. Para GSSL, fueron generadas 0.5, 1, 2 y 5 millones de características y luego fueron filtradas utilizando *valores de corte* igual a 1, 2 y 5. Finalmente, para DTSL,

[3]https://dharma.frm.utn.edu.ar/papers/ijait14

[4]Una característica positiva consiste de una conjunción de estados de variables, donde no hay estados con valor 0 (falso).

usamos κ igual a 1.0, 0.1, 0.01, 0.001 y 0.0001 para obtener una estructura desde la cual fue generado un conjunto de características utilizando 4 métodos de generación: DEFAULT, PRUNE, PRUNE-5 y PRUNE-10.

Para los algoritmos CSPC y CSGS, utilizamos la prueba χ^2 con un nivel de significancia igual a 0.05, la cual utiliza un ADTree para reducir significativamente los tiempos de ejecución (ver Sección 4.4.2). Para ambos algoritmos, utilizamos el mismo conjunto de contextos, el cual fue construido utilizando aquellas observaciones que aparecían al menos alguna vez en el conjunto de datos utilizado para construir la estructura (ver Sección 4.3.1). Finalmente, para reducir tiempos de ejecución, ambos algoritmos fueron ejecutados en forma paralela (ver el Algoritmo 14).

5.2.2. Estimación de parámetros

Una vez obtenida la estructura de una red de Markov, sus parámetros fueron estimados optimizando la pseudo-verosimilitud de los datos a través del algoritmo LBFGS [53], el cual es parte del paquete *Libra toolkit*[5]. Debido a que Libra toolkit únicamente estima parámetros para estructuras representadas a través de un conjunto de características [58], para los grafos aprendidos por los algoritmos basados en restricciones, obtuvimos un conjunto de características de una manera similar a la discutida en la Sección 4.5. Es decir, dado un grafo, obtuvimos sus cliques máximos. Para cada clique máximo, generamos una característica por cada posible combinación de estados de las variables involucradas en el clique máximo.

Para optimizar la pseudo-verosimilitud, tuvimos en cuenta dos alternativas: una sin regularizar y otra regularizada. Para la regularización, utilizamos una combinación de las normas Gaussiana y Laplaciana. Para la norma Gaussiana, utilizamos como hiperparámetros los valores 0.1, 0.5 y 1. Para la norma Laplaciana, utilizamos como hiperparámetros los valores 1, 5 y 10. Estos valores fueron seleccionados según los recomendados en la literatura [56, 89, 88].

[5]Libra toolkit está públicamente disponible en: `http://libra.cs.uoregon.edu/`. En nuestros experimentos, utilizamos la versión 0.5.0.

5.3. Metodología

Para evaluar empíricamente la correctitud de las estructuras obtenidas por los algoritmos CSPC y CSGS, nuestra metodología se basó en la siguiente observación. Sea $p^\star(X)$ una distribución arbitraria sobre X y D una muestra tomada desde $p^\star(X)$. Sea $(\tilde{S}, \tilde{\theta})$ una red de Markov construida desde D, donde $\tilde{p}(X)$ es la distribución definida por $(\tilde{S}, \tilde{\theta})$. Entonces diremos que la red de Markov $(\tilde{S}, \tilde{\theta})$, y puntualmente su estructura $\tilde{S}$, son "correctas", si la distribución $\tilde{p}(X)$ es "cercana" a la distribución $p^\star(X)$, i.e., $\tilde{p}(X) \approx p^\star(X)$.

Claramente, la forma de evaluar lo anterior varía ampliamente entre los dos escenarios de experimentación que hemos propuesto, debido a que la distribución $p^\star(X)$ es únicamente conocida en el escenario sintético. Por esta razón, en cada escenario, se siguió una metodología distinta para evaluar la correctitud de las estructuras aprendidas. Sin embargo, ambas metodologías son esencialmente similares, ellas consisten básicamente en dos grandes pasos: i) obtener una red de Markov $(\tilde{S}, \tilde{\theta})$ desde un conjunto D; y ii) medir la correctitud de la red $(\tilde{S}, \tilde{\theta})$. En lo que resta de esta sección, se describirá el primer paso, mientras que dejaremos para la Subsección 5.3.3 los detalles involucrados en el segundo paso.

5.3.1. Metodología en el escenario sintético

Para el escenario sintético, se utilizó los conjunto de datos descriptos en la Sección 5.1.1. Dado un conjunto D, una red de Markov $(\tilde{S}, \tilde{\theta})$ fue obtenida en dos etapas. Primero, la estructura $\tilde{S}$ fue obtenida utilizando cada uno de los algoritmos mostrados en la Sección 5.2.1. Segundo, para cada estructura $\tilde{S}$, los parámetros $\tilde{\theta}$ fueron estimados optimizando la pseudo-verosimilitud no regularizado[6] del conjunto D. Sin embargo, debido que los algoritmos de estimación de densidad presentan diferentes combinaciones de parámetros de configuración, lo cual afecta la correctitud de

[6] Al utilizar regularización durante la optimización, se pueden producir cambios estructurales en la estructura $\tilde{S}$ [46, Capítulo 20]. Estos cambios son ocasionados cuando una característica toma como parámetro un valor muy cercano (o igual) a cero. Como estamos únicamente interesados en evaluar las capacidades de cada algoritmo de aprendizaje de estructuras, utilizamos pseudo-verosimilitud no regularizado para evitar cambios estructurales debido al valor de los parámetros.

la estructura aprendida, procedimos de la siguiente manera para obtener una red de Markov. Antes de aprender la estructura $\tilde{S}$, el conjunto D fue dividido en dos subconjuntos disjuntos: el *conjunto de entrenamiento*, denotado por D_E, el cual representa un 70 % del conjunto D; y el *conjunto de validación*, denotado por D_V, el cual representa el restante 30 % del conjunto D. El conjunto D_E fue utilizado para obtener una estructura por cada combinación de parámetros de configuración. Por ejemplo, con BLM se obtuvieron $9 \times 4 \times 3$ estructuras, con L1 se obtuvieron 8, con GSSL se obtuvieron 4×3, y con DTSL se obtuvieron 4. Luego, para cada una de estas estructuras, se estimaron sus parámetros utilizando el conjunto D_E. Para determinar cuál de las redes de Markov era la más correcta, seguimos el procedimiento propuesto en [51, 55, 89, 88, 57, 36, 56]. Este procedimiento consistió en seleccionar la red de Markov con máximo pseudo-verosimilitud en el conjunto D_V.

5.3.2. Metodología en el escenario del mundo real

Para el escenario del mundo real, utilizamos los conjuntos de datos mostrados en la Sección 5.1.2. Para obtener una red de Markov $(\tilde{S}, \tilde{\theta})$ desde un conjunto D, utilizamos *holdout testing* [41]. Para ello, el conjunto D fue dividido en tres subconjuntos disjuntos: el *conjunto de entrenamiento* (D_E), el *conjunto de validación* (D_V), y el *conjunto de prueba* (D_P). Los conjuntos de datos mostrados en el Cuadro 5.2 fueron divididos de igual manera como son habitualmente divididos en otros trabajos en la práctica, garantizando la evaluación de nuestros resultados frente a los de otros trabajos. Los tamaños de cada uno de estos conjuntos son mostrados en el Cuadro 5.4.

Siguiendo *holdout testing*, el conjunto D_E fue utilizado para construir una estructura utilizando cada algoritmo de aprendizaje de estructuras. Obtenida la estructura, y a diferencia del escenario sintético, los parámetros de la red de Markov fueron estimados *usando regularización*. La razón es doble. Primero, a diferencia del escenario sintético, en el escenario del mundo real no conocemos la estructura subyacente, por lo tanto no estamos preocupados por los cambios estructurales debido a los valores de los parámetros. Segundo, utilizar regularización durante la optimización de la pseudo-log-verosimilitud garantiza reducir posibles errores de sobreajuste. Los errores de so-

Conjunto D	$\lvert D_E \rvert$	$\lvert D_V \rvert$	$\lvert D_P \rvert$
NLTCS	16,181	2,157	3,236
MSNBC	291,326	38,843	58,265
Abalone	3,134	417	626
Wine	4,874	650	975
KDDCup 2000	80,092	19,907	34,955
Plants	17,412	2,321	3,482
Covertype	30,000	4,000	6,000
Audio	15,000	2,000	3,000
Netflix	15,000	2,000	3,000
Accidents	12,758	1,700	2,551
Adult	36,631	4,884	7,327
Retail	22,041	2,938	4,408
Pumsb Star	12,262	1,635	2,452
DNA	1,600	400	1,186
MSWeb	29,441	3,270	5,000
WebKB	2,803	558	838
BBC	1,670	225	330
Ad	2,461	327	491

Cuadro 5.4: Conjuntos de datos de problemas del mundo real. Las columnas $\lvert D_E \rvert$, $\lvert D_V \rvert$ y $\lvert D_P \rvert$ muestran el número de observaciones para los conjuntos de entrenamiento, validación y prueba, respectivamente.

breajuste conducen a un pobre rendimiento en *holdout testing* [41]. Por estas razones, utilizando la estructura $\tilde{S}$ obtenida por un algoritmo de aprendizaje de estructuras, se estimaron 3×3 conjuntos de parámetros diferentes, uno por cada combinación de hiperparámetros (ver Sección 5.2.2). Así, utilizando el conjunto D_V, seleccionamos la red de Markov con máximo pseudo-verosimilitud.

5.3.3. Métricas

Según lo comentado en la Sección 5.1.1, en el escenario sintético, la distribución $p^\star(X)$ fue definida desde una red de Markov, la cual denotaremos como $(S^\star, \theta^\star)$. Así, al aprender una red de Markov $(\tilde{S}, \tilde{\theta})$ desde una muestra D tomada de $p^\star(X)$, la correctitud de $(\tilde{S}, \tilde{\theta})$ puede ser medida a través de dos formas: i) comparar la semejanzas entre las estructuras $\tilde{S}$ y $S^\star$; y ii) comparar la similitud entre las distribuciones $\tilde{p}(X)$ y $p^\star(X)$. En contraste, como en el escenario del mundo real utilizamos *holdout*

testing, medimos la habilidad predictiva de la red de Markov $(\tilde{S}, \tilde{\theta})$ (la cual fue obtenida utilizando los conjuntos D_E y D_V) sobre el conjunto D_P. En lo que resta de esta sección, detallaremos cada una de las comparaciones previamente mencionadas.

Para comparar las estructuras $\tilde{S}$ y $S^\star$, definimos una métrica en base a la siguiente observación. Como muestra la Figura 5-1, la estructura $S^\star$ puede ser representada utilizando dos grafos instanciados, i.e., $G(X_f = 0)$ y $G(X_f = 1)$. Así, la estructura $S^\star$ muestra un interesante patrón cualitativo a saber: ambos grafos instanciados tienen diferentes topologías. Más concretamente, el grafo $G(X_f = 0)$ tiene un único clique máximo de tamaño n, mientras que el grafo $G(X_f = 1)$ tiene $(n-1)$ cliques máximos de tamaño 2. *Este patrón es una consecuencia de la presencia de las independencias específicas del contexto en la estructura.*

Como se mostró en el Ejemplo 3.4, estos dos grafos pueden ser alternativamente representados por un conjunto $\mathcal{F}$ de características. Debido a la asimetría en la topología de ambos grafos instanciados, al representar dichos grafos como un conjunto $\mathcal{F}$, el conjunto $\mathcal{F}$ está formado por dos conjuntos disjuntos de características: *i)* el conjunto $\mathcal{F}_0 \equiv \mathcal{F}[X_f = 0]$, que se corresponde con el grafo $G(X_f = 0)$; y *ii)* el conjunto $\mathcal{F}_1 \equiv \mathcal{F}[X_f = 1]$, que se corresponde con el grafo $G(X_f = 1)$. **Ambos conjuntos de características están precisamente diferenciados por la longitud de sus características**. En otras palabras, según el estado de la variable bandera X_f, las características del conjunto $\mathcal{F}_0$ tienen longitud igual a n, mientras que las características del conjunto $\mathcal{F}_1$ tienen longitud 2.

Utilizando este hecho podemos evaluar la correctitud de una estructura $\tilde{S}$ aprendida por un algoritmo de aprendizaje de estructuras. Como comentamos en la Sección 5.2.2, la estructura $\tilde{S}$ aprendida por cualquier algoritmo es representada como un conjunto $\tilde{\mathcal{F}}$ de características. Así, para evaluar si $\tilde{\mathcal{F}}$ codifica las independencias específicas del contexto del problema, simplemente medimos las longitudes promedio de los subconjuntos $\tilde{\mathcal{F}}_0$ y $\tilde{\mathcal{F}}_1$, las cuales deberían tender a n y 2, respectivamente.

Por otro lado, para medir que tan similares son las distribuciones $\tilde{p}(X)$ y $p^\star(X)$, utilizamos la *divergencia de Kullback-Leibler* (KL). La KL nos ofrece una forma para evaluar la diferencia entre dos distribuciones [49, 17]. La KL puede ser pensada como

una "métrica de distancia" entre dos distribuciones. Aunque la KL no es simétrica, satisface el resto de las propiedades básicas de una métrica [17]. Formalmente, la KL de la distribución $\tilde{p}(X)$ con respecto a la distribución $p^\star(X)$ se define como sigue:

$$KL(p^\star \mid\mid \tilde{p}) = \sum_{x \in val(V)} p^\star(x) \log \frac{p^\star(x)}{\tilde{p}(x)},$$

donde la KL es cero, si y solo si $p^\star(x) = \tilde{p}(x)$ para todo $x \in val(V)$, y mayor a cero en cualquier otro caso. En otras palabras, cada vez que la probabilidad de un estado x es igual en ambas distribuciones, $p^\star(x) = \tilde{p}(x)$, entonces $log(\frac{p^\star(x)}{\tilde{p}(x)}) = 0$. Si la KL es cero, entonces ambas distribuciones, $p^\star(X)$ y $\tilde{p}(X)$, *asignan la misma probabilidad para todo estado* $x \in val(V)$.

Desde el punto de vista de la teoría de la información, la KL puede ser intuitivamente interpretada como la *pérdida en información* cuando la distribución $\tilde{p}(X)$ es usada en vez de $p^\star(X)$. En teoría de la información, "información" es técnicamente definida como el negativo de el logaritmo de una distribución de probabilidad [17].

En base a esto, cuando utilizamos una distribución $\tilde{p}(X)$ como una aproximación de $p^\star(X)$ ocurre una pérdida de información cada vez que $p^\star(x) \neq \tilde{p}(x)$. Por lo tanto, dado un conjunto $\tilde{p}_1(X), \ldots, \tilde{p}_k(X)$ de distribuciones que aproximan a una distribución $p^\star(X)$ (e.g., aquellas obtenidas desde las estructuras aprendidas por diferentes algoritmos de aprendizaje de estructuras), podemos utilizar la KL como un criterio para seleccionar cuál de las aproximaciones es más "semejante" a $p^\star(X)$. En otras palabras, decimos que la distribución $\hat{p}(X) \in \{\, \tilde{p}_1(X), \ldots, \tilde{p}_k(X) \,\}$ es la más semejante a $p^\star(X)$, si

$$\hat{p}(X) = \operatorname*{arg\,mín}_{\tilde{p} \in \{\, \tilde{p}_1, \ldots, \tilde{p}_k \,\}} KL(\tilde{p} \mid\mid p^\star).$$

Desafortunadamente, el cómputo de la KL requiere conocer la distribución $p^\star(X)$, la cual, en el mayoría de los problemas del mundo real, es desconocida[7]. Debido a esta razón, en nuestro escenario del mundo real, no podemos utilizar la KL para medir la

[7]Por así decirlo, si la distribución $p^\star(X)$ fuese conocida en un problema, entonces, sólo en principio, no sería necesario el aprendizaje de una distribución $\tilde{p}(X)$ desde un conjunto D (muestreado desde $p^\star(X)$).

precisión de una red de Markov estimada desde un conjunto de datos. Sin embargo, es posible obtener una métrica aproximada de la KL, la cual no requiera conocer de forma directa la distribución $p^\star(X)$. Para mostrar esto último, analicemos más en detalle la definición de la KL entre dos distribuciones:

$$\begin{aligned} KL(p^\star \,||\, \tilde{p}) &= \sum_{x \in val(V)} p^\star(x) \log \frac{p^\star(x)}{\tilde{p}(x)}, \\ &= \sum_{x \in val(V)} p^\star(x) \left\{ \log p^\star(x) - \log \tilde{p}(x) \right\}, \\ &= \sum_{x \in val(V)} p^\star(x) \log p^\star(x) - p^\star(x) \log \tilde{p}(x), \\ &= \sum_{x \in val(V)} p^\star(x) \log p^\star(x) - \sum_{x \in val(V)} p^\star(x) \log \tilde{p}(x). \end{aligned} \tag{5.5}$$

Las ecuaciones de arriba muestran diferentes expresiones de la KL, las cuales son todas equivalentes. De estas ecuaciones, la Ecuación (5.5) permite entrever un hecho importante. El primer término de la Ecuación (5.5) es un valor que no depende de la distribución $\tilde{p}(X)$.

De este modo, si estamos comparando diferentes distribuciones que aproximan a una distribución $p^\star(X)$ fija, e.g. $\{ \tilde{p}_1(X), \ldots, \tilde{p}_k(X) \}$, el valor del primer término de la Ecuación (5.5) es el mismo para cualquier distribución $\tilde{p}(X)$ que aproxima $p^\star(X)$, por lo tanto dicho término no tiene un rol relevante al momento de seleccionar la mejor aproximación de $p^\star(X)$. En consecuencia, bajo la situación planteada anteriormente, la minimización de la KL con respecto a $p^\star(X)$ implica la selección de una distribución $\tilde{p}(X)$ que *maximice* la ecuación

$$\sum_{x \in val(V)} p^\star(x) \log \tilde{p}(x), \tag{5.6}$$

donde la Ecuación (5.6) es precisamente el segundo término de la Ecuación (5.5).

A pesar de la simplificación realizada sobre la Ecuación (5.5), la Ecuación (5.6)

depende de la distribución $p^\star(X)$. Sin embargo, analizando la Ecuación (5.6), esta es similar a la log-verosimilitud de $\tilde{p}(X)$, ver la Ecuación (2.24). Para ser más preciso, se puede demostrar que la log-verosimilitud es un estimador de la Ecuación (5.6) valido asintóticamente, es decir, teniendo un muestra D de $p^\star(X)$ lo suficientemente grande, la Ecuación (5.6) se corresponde con la log-verosimilitud de $\tilde{p}(X)$. Más concretamente,

$$\sum_{x \in val(V)} p^\star(x) \log \tilde{p}(x) \approx \sum_{x \in D} \log \tilde{p}(x), \tag{5.7}$$

donde D es una muestra tomada desde $p^\star(X)$. Como vimos en la Sección 2.5, para dominios con un alto número de variables, el computo de la log-verosimilitud resulta intratable. Para dichos casos, podemos utilizar la pseudo-logverosimilitud como una aproximación de la log-verosimilitud, ver la Ecuación (2.25). Por lo tanto, utilizando la pseudo-logverosimilitud, otra aproximación para la Ecuación (5.6) puede ser obtenida:

$$\sum_{x \in D} \log \tilde{p}(x) \approx \sum_{x \in D} \sum_{a \in V} \log \tilde{p}(x_a | x_{V \setminus \{a\}}). \tag{5.8}$$

En la práctica, la precisión de una red de Markov es usualmente medida a través del *conditional marginal log-likelihood* (CMLL) [51, 18, 56, 89, 36], cuya definición se basa en la Ecuación (5.8). En otras palabras, la CMLL nos permite determinar qué tan semejante es una distribución aprendida, $\tilde{p}(X)$, con respecto a una distribución subyacente, $p^\star(X)$.

En nuestra experimentación en el escenario del mundo real, utilizamos la CMLL para medir la precisión de una red de Markov, lo cual nos brinda un criterio para seleccionar la red de Markov más precisa entre las redes obtenidas a partir de estructuras aprendidas por diferentes algoritmos de aprendizaje de estructuras. A continuación, explicaremos en detalle como computar la CMLL.

Para computar el valor de la CMLL, se utiliza una distribución $\tilde{p}(X)$ y un conjunto de datos *no utilizado durante la construcción de la distribución* $\tilde{p}(X)$. En nuestro caso,

utilizamos el conjunto D_P, el cual no es utilizado para obtener una distribución (el aprendizaje se realiza utilizando el conjunto D_E).

De este modo, las variables X del dominio son divididas en dos subconjuntos disjuntos: X_Q (conocidas como las *variables de consulta*) y X_E (conocidas como las *variables de evidencia*), tal que $E = V \setminus Q$. En base a lo anterior, la CMLL es computada sumando la distribución condicional marginal de cada variable $X_a \in X_Q$ condicionada en las restantes variables X_E. Formalmente, el valor de la CMLL se obtiene como sigue:

$$\text{CMLL}(\tilde{p}, D_P, Q, E) = \sum_{x \in D_P} \sum_{a \in Q} \log \tilde{p}(x_a | x_E), \tag{5.9}$$

donde el valor de $\tilde{p}(x_a|x_E)$ fue obtenido utilizando 1000 muestras tomadas desde $\tilde{p}(X)$ con un muestreo de Gibbs (a través de Libra Toolkit) configurado con 10 cadenas y un valor de *burn-in* igual a 100.

Sin embargo, el valor de la CMLL obtenido por la Ecuación (5.9) se encuentra sesgado según qué elementos conforman los conjuntos Q y E. Por ejemplo, diferentes elementos en el conjunto Q puede resultar en valores de CMLL diferentes. Para minimizar este sesgo, se utiliza el siguiente procedimiento, el cual tiene como objetivo que cada elemento en V aparezca al menos una vez en el conjunto Q.

El conjunto V es dividido en cuatro subconjuntos disjuntos: $\{ V_i \}_{i=1}^4$, donde $V = \bigcup_{i=1}^4 V_i$ y $V_i \cap V_j = \emptyset$ para todo $i, j = 1, \ldots, 4$ con $i \neq j$. Luego, cada subconjunto V_i es utilizado como conjunto de consulta, mientras que los restantes subconjuntos, $V \setminus V_i$, son utilizados como conjunto de evidencia. Entonces, este procedimiento es repetido tal que cada subconjunto V_i es tomado al menos una vez como conjunto de consulta. En otras palabras, el valor de la CMLL es obtenido de la siguiente manera:

$$\text{CMLL}(\tilde{p}, D_P) = \sum_{i=1}^{4} \text{CMLL}(\tilde{p}, D_P, V_i, V \setminus V_i). \tag{5.10}$$

Finalmente, para obtener valores de CMLL más comparables entre diferentes do-

minios, la Ecuación (5.10) puede ser dividida por el número de variables:

$$\mathrm{NCMLL}(\tilde{p}, D_P) = \frac{\mathrm{CMLL}(\tilde{p}, D_P)}{|V|}. \tag{5.11}$$

5.4. Resultados en el escenario sintético

Como comentamos en la Sección 5.1.1, para cada número $n \in \{ 6, 7, 8, 9 \}$ de variables, generamos conjuntos de datos cuyos tamaños varían desde 100 a 100k observaciones. Por cada uno de estos conjuntos de datos se obtuvo una red de Markov, cuya estructura fue aprendida por cada uno de los algoritmos de aprendizaje de estructuras. Para homogeneizar nuestros análisis, toda estructura aprendida por un algoritmo fue transformada a un conjunto $\mathcal{F}$ de características. Luego, las redes de Markov obtenidas fueron evaluadas utilizando la longitud promedio de sus características y su valor de KL.

Es importante remarcar la razón de utilizar conjuntos de datos de tamaño variable. Esto se debe a que una prueba estadística, utilizada por cualquier algoritmo basado en restricciones (esto incluye nuestros algoritmos, CSPC y CSGS), asume un número mínimo de observaciones para garantizar resultados correctos. Al no cumplirse dicho supuesto, las pruebas estadísticas pueden retornar valores de verdad incorrectos, lo cual resulta en estructuras incorrectas. Por lo tanto, al utilizar mayor número de datos, podemos distinguir dos situaciones en relación a la disponibilidad de datos.

Una situación es ante una *alta disponibilidad de datos*. Esto implica situaciones donde el número n de variables es bajo y el número $|D|$ de observaciones es alto (e.g., $n = 6$ y $|D| = 100k$). Así, en tales situaciones, se espera que una prueba de independencia no cometa errores, resultando en que los algoritmos basados en restricciones obtengan estructuras más correctas, es decir, que dichas estructuras presenten un bajo número de errores del tipo I o II.

En contraste, la otra situación surge ante una *baja disponibilidad de datos*. Esto implica situaciones donde el número n es alto y el número $|D|$ es bajo (e.g., $n = 9$ y $|D| = 100$). Como resultado, en situaciones con baja disponibilidad de datos, se

espera que las estructuras construidas puedan presentar un alto número de errores del tipo I o II.

En lo que resta de esta sección, la Sección 5.4.1 analiza las longitudes promedio obtenidas desde las estructuras aprendidas. La Sección 5.4.2 analiza los valores de KL obtenidos desde las redes de Markov aprendidas. Finalmente, la Sección 5.4.3 presenta una comparación del número de pruebas de independencia ejecutadas por los algoritmos CSPC y CSGS para construir una estructura.

5.4.1. Análisis de longitudes de características

La Figura 5-2 muestra las longitudes promedio computadas desde los conjuntos $\mathcal{F}_0$ y $\mathcal{F}_1$ construidos por cada algoritmo de aprendizaje. Dicha figura está compuesta por cuatro subfiguras, una por cada valor de n. Para cada valor de n, las longitudes fueron visualmente representadas utilizando un mapa de calor. En cada mapa de calor (uno por cada valor de n), cada fila representa un algoritmo, cada columna representa una cantidad determinada de datos, y cada celda muestra la longitud promedio. Así, una celda en el mapa representa la longitud promedio obtenida por un algoritmo desde un conjunto de datos con un tamaño en particular.

La idea de utilizar un color (o escala del gris) asociado a una longitud promedio es para mostrar visualmente el valor de una celda. El color asociado a una celda sigue la siguiente regla. Mientras mayor es el valor de una celda (alta longitud promedio) su color tiende a ser rojo intenso (gris oscuro). En caso contrario, cuando la celda presenta una baja longitud promedio, el color de la celda tiende a ser blanco.

Para nuestra experimentación, esta codificación de valores de longitud a colores (escala de grises) facilita la detección de dos patrones de especial interés a observa en los mapas de calor.

Primero, un patrón esperado es que las longitudes promedio de las características en $\mathcal{F}_0$ estén asociados con colores (escala de grises) tendiendo a rojo (gris oscuro), mientras que las longitudes promedio de $\mathcal{F}_1$ estén asociados a colores tendiendo a blanco. Al presentarse este contraste entre ambas colores se garantiza que la estructura aprendida por un algoritmo está codificando las independencias específicas del

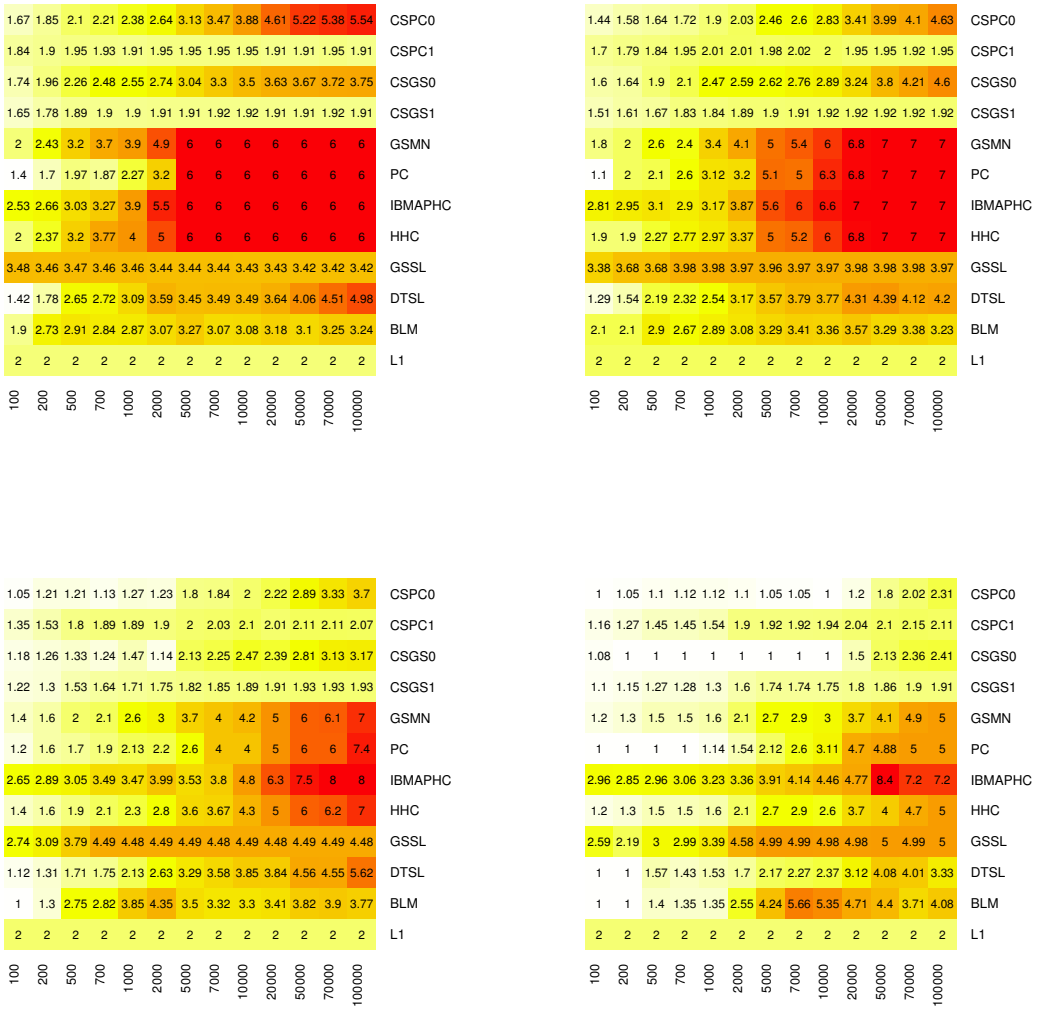


Figura 5-2: Longitud promedio de las características aprendidas por cada algoritmo para cada valor de n ($n = 6$ arriba izquierda, $n = 7$ arriba derecha, $n = 8$ abajo izquierda, y $n = 9$ abajo derecha). Cada celda es la longitud promedio para 10 conjuntos de datos de tamaño fijo.

contexto del problema.

El segundo patrón esperado es que las longitudes promedio obtenidas desde las estructuras aprendidas por cualquier algoritmo basado en restricciones deberían tender siempre a rojo (gris oscuro). Esto es porque la distribución subyacente del problema no presenta independencias condicionales, entonces los algoritmos basados en restricciones construyen un grafo completo. Como un grafo completo tiene un único clique

máximo de tamaño n, la longitud de las características inducidas desde dicho grafo toman el máximo valor (el cual es igual a n).

Antes de analizar en detalle los mapas de calor, es necesario remarcar algo sumamente importante. En cada uno de los mapas de calor encontramos un par de filas denominadas "CSPC0" y "CSPC1" (como también "CSGS0" y "CSGS1"). Cada una de estas filas se corresponden a los conjuntos $\mathcal{F}_0$ y $\mathcal{F}_1$ obtenidos por CSPC (y CSGS). En contraste, para todos los restantes algoritmos, las filas únicamente se corresponden con un único conjunto de características, el conjunto $\mathcal{F}$.

La causa de esto se debe sencillamente a dos razones. En el caso de los algoritmos basados en restricciones, el conjunto $\mathcal{F}$ es inducido desde un único grafo no dirigido, en consecuencia, *las longitudes promedio de los subconjuntos $\mathcal{F}_0$ y $\mathcal{F}_1$ son iguales*. En el caso de los algoritmos de estimación de densidad, el conjunto $\mathcal{F}$ consiste únicamente de características positivas[8], en consecuencia, el conjunto $\mathcal{F}_0 = \emptyset$, *impidiendo el computo de la longitud promedio del conjunto $\mathcal{F}$.*

Al analizar los mapas de calor, una primer tendencia que se destaca son las longitudes promedio obtenidas por los algoritmos CSPC y CSGS. En otras palabras, CSPC y CSGS son los únicos algoritmos que aprenden una estructura donde las longitudes promedio de los conjuntos $\mathcal{F}_0$ y $\mathcal{F}_1$ son diferentes (o asimétricas). En términos estrictamente estructurales, esta falta de asimétrica en las longitudes obtenidas por los restantes algoritmos ofrece una fuerte evidencia a favor de que los algoritmos CSPC y CSGS son los únicos algoritmos con la capacidad de capturar las independencias específicas del contexto.

Adicionalmente, las longitudes de los conjuntos $\mathcal{F}_0$ y $\mathcal{F}_1$ muestran el patrón esperado, es decir, la longitud promedio del conjunto $\mathcal{F}_0$ es cercana a n (tienden a rojo), mientras que la longitud promedio del conjunto $\mathcal{F}_1$ es cercana a 2 (tiende a blanco). Este patrón resulta más pronunciado en situaciones con alta disponibilidad de datos debido a que los resultados obtenidos por una prueba de independencia son más correctos.

[8]Vale aclarar que el algoritmo DTSL puede aprender características no positivas. Sin embargo, al igual que los algoritmos basados en restricciones, las longitudes de los conjuntos $\mathcal{F}_0$ y $\mathcal{F}_1$ son iguales.

Por el lado de los algoritmos basados en restricciones, las longitudes promedio obtenidas tienden a ser cercanas a n cuando hay una alta disponibilidad de datos. Este resultado es esperable porque justamente la *distribución* $p^\star(X)$ *no codifica supuestos de independencia condicional*, por lo tanto, un algoritmo basado en restricciones construye un grafo que no codifica independencias (un grafo completo).

Vale remarcar que, dentro de los algoritmos basados en restricciones, IBMAP-HC tiene una tendencia particular. Por ejemplo, en $n = 9$ y $|D| >= 50k$, IBMAP-HC es el único algoritmo basado en restricciones que obtiene las más altas longitudes. La razón de esta tendencia es doble. Primero el algoritmo IBMAP-HC está justamente diseñado para reducir errores producidos por ejecutar varias pruebas de independencia de forma independiente. Segundo, IBMAP-HC utiliza una prueba de independencia diferente a la prueba de independencia utilizada por todos los restantes algoritmos. Así, es esperable que los resultados de IBMAP-HC sean ligeramente diferentes.

Por el lado de los algoritmos de estimación de densidad, la tendencia de las longitudes promedia obtenidas son bastantes diferentes a aquellas obtenidas por los restantes algoritmos. GSSL y DTSL son los algoritmos de estimación de densidad que obtiene las longitudes más altas. En contraste, el algoritmo L1 tiene una longitud promedio igual a 2 para todos los casos. Esto se debe a que el algoritmo L1 está exclusivamente diseñado para aprender características de tamaño 2 (ampliamente conocidas como *pairwise features*).

5.4.2. Análisis de valores de KL

Para realizar un profundo análisis de KL, dicho análisis fue realizado en tres etapas consecutivas. En la primera etapa, se obtuvieron valores de KL desde redes de Markov cuyas estructuras fueron fijadas de antemano, las cuales denominamos *estructuras de control*. Estas estructuras no fueron obtenidas por algoritmos de aprendizaje de estructuras, sino que fueron intencionadamente seleccionadas para representar situaciones hipotéticas de errores estructurales. Por ejemplo, una de las estructuras de control seleccionada fue la estructura $S^\star$, es decir, la estructura subyacente del problema. El valor de KL obtenido desde una red de Markov cuya estructura es $S^\star$ nos

aporta información relevante. Dicho valor de KL representa el valor mínimo de KL que puede aspirar un algoritmo de aprendizaje de estructuras. En otras palabras, en el mejor de los casos, un algoritmo aprenderá la estructura $S^\star$. De esta manera, los valores de KL obtenidos en esta primera etapa servirán como base para comprender mejor los análisis realizados en las próximas etapas, e.g., comprender los valores de KL obtenidos desde las estructuras aprendidas por diferentes algoritmos.

En la segunda etapa, los valores de KL fueron obtenidos desde redes de Markov cuyas estructuras fueron construidas por los diferentes algoritmos de aprendizaje de estructura propuestos para nuestra experimentación. El objetivo de esta etapa es analizar las tendencias de KL que surgen desde las estructuras aprendidas por cada algoritmo. Como veremos más adelante, los valores de KL obtenidos en esta etapa se encuentran íntimamente relacionados con las características particulares de cada algoritmo de aprendizaje. Por ejemplo, los valores de KL obtenidos por nuestros algoritmos se diferencian de los valores de KL obtenidos por los algoritmos basados en restricciones y los algoritmos de estimación de densidad; y los valores de KL obtenidos por estos dos últimos tipos de algoritmos, se diferencian entre sí.

En la tercera etapa, compararemos los valores de KL obtenidos en las dos etapas previamente descriptas. El objetivo de esta comparación es analizar los comportamientos de los valores de KL obtenidos desde las estructuras aprendidas por los diferentes algoritmos y desde las estructuras de control. Principalmente, analizaremos cuáles algoritmos logran aprender estructuras que mejor aproximen los valores de KL obtenidos desde la estructura $S^\star$.

Para todas las etapas, los valores de KL fueron computados de la misma manera. Primero se obtuvo una estructura. Segundo, utilizando dicha estructura, se obtuvo una red de Markov a través de la estimación de sus parámetros. Finalmente, la red de Markov resultante fue utilizada para computar el valor de KL según lo descripto en la Sección 5.3.3. Este procedimiento lo realizamos para cada uno de los valores de n y para cada uno de los diferentes conjuntos de datos. Debido a que para un valor fijo de n y un tamaño fijo de conjunto de datos, tenemos 10 conjuntos de datos diferentes (ver Sección 5.3), computamos 10 valores de KL distintos, los cuales fueron

promediados para obtener un único valor.

Estructuras de control

Para obtener los valores de KL, en esta etapa se seleccionó un conjunto de estructuras de control. Las estructuras de control seleccionadas fueron las siguientes: la *estructura completa*, la *estructura subyacente*, la *estructura vacía* y la *estructura estrella*. Estas estructuras pueden ser agrupadas en dos grupos.

El primer grupo, denominado *grupo I-map*, está compuesto por la estructura completa y subyacente. Estas estructuras se caracterizan por ser estructuras I-map. Según lo visto en la Sección 2.1.1, una estructura I-map puede presentar errores del tipo II, es decir, dicha estructura no codifican independencias falsas pero puede (o no) codificar dependencias falsas (e.g., la estructura subyacente no codifica dependencias falsas).

El segundo grupo, denominado *grupo no I-map*, está compuesto por la estructura vacía y estrella. Estas estructuras se caracterizan por ser estructuras no I-map. Tales estructuras pueden presentar errores del tipo I, es decir, la estructura puede codificar independencias falsas.

La siguiente lista presenta una descripción detallada de las características que presenta cada una de las estructuras de control.

1. Grupo no I-map

    - La estructura vacía codifica el número máximo de independencias falsas para nuestro problema. Esta estructura corresponde a un grafo $G = (V, E)$ donde $E = \emptyset$. Como consecuencia, para todo $a, b \in V$, tenemos que $\text{sep}_G(a, b; \emptyset)$, lo cual implica el supuesto $I(X_a \perp X_b \mid \emptyset)$, el cual no se satisface en $p^\star(X)$. Debido a la presencia de independencias falsas, se espera que las redes de Markov cuya estructura sea vacía obtengan valores altos de KL.
    - La estructura estrella consiste de un grafo $G = (V, E)$ donde existe una arista $(a, f) \in E$ para cada nodo $a \in V \setminus \{ f \}$, donde $f \in V$ es el nodo ban-

dera. Esta estructura codifica independencias falsas para nuestro problema debido a que, para cada par $a, b \in V \setminus \{ f \}$, tenemos que $\text{sep}_G(a, b; f)$, lo cual implica el supuesto $I(X_a \perp X_b \mid X_f)$, el cual no se satisface en $p^\star(X)$. Más concretamente, únicamente el supuesto $I(X_a \perp X_b \mid X_f = 1)$ se satisface en $p^\star(X)$, mientras que el supuesto $I(X_a \perp X_b \mid X_f = 0)$ no se cumple en $p^\star(X)$.

2. Grupo I-map

   - La estructura completa codifica dependencias falsas, concretamente, las independencias específicas del contexto no están codificadas. Como resultado, la red de Markov, cuya estructura es completa, contendrá parámetros espurios. Como analizamos en la Sección 5.4.1, la estructura completa es la estructura que tienden a construir los algoritmos basados en restricciones en nuestro problema.
   - La estructura subyacente es precisamente la estructura $S^\star$, es decir, la estructura que codifica el número correcto de independencias y dependencias. De este modo, se espera que las redes de Markov, cuya estructura sea $S^\star$, obtengan los valores de KL más bajos.

De esta manera, el grupo I-map nos permite entender el impacto que tienen la presencia de errores del tipo II[9] sobre los valores de KL. En contraste, el grupo no I-map nos permite entender el impacto que tienen los errores del tipo I sobre los valores de KL.

La Figura 5-3 muestra los valores de KL obtenidos al utilizar las cuatro estructuras propuestas. Cada subfigura corresponde a un determinado valor de n: 6 (arriba izquierda), 7 (arriba derecha), 8 (fondo izquierda) y 9 (fondo derecha). El eje Y indica el valor de KL (en espacio logarítmico) y el eje X indica la cantidad de datos. Las curvas en cada subfigura están asociadas a las cuatro estructuras propuestas. En una curva, cada punto representa el promedio de los valores de KL para los 10 diferentes

[9] excepto la estructura subyacente, la cual no contiene errores del tipo II.

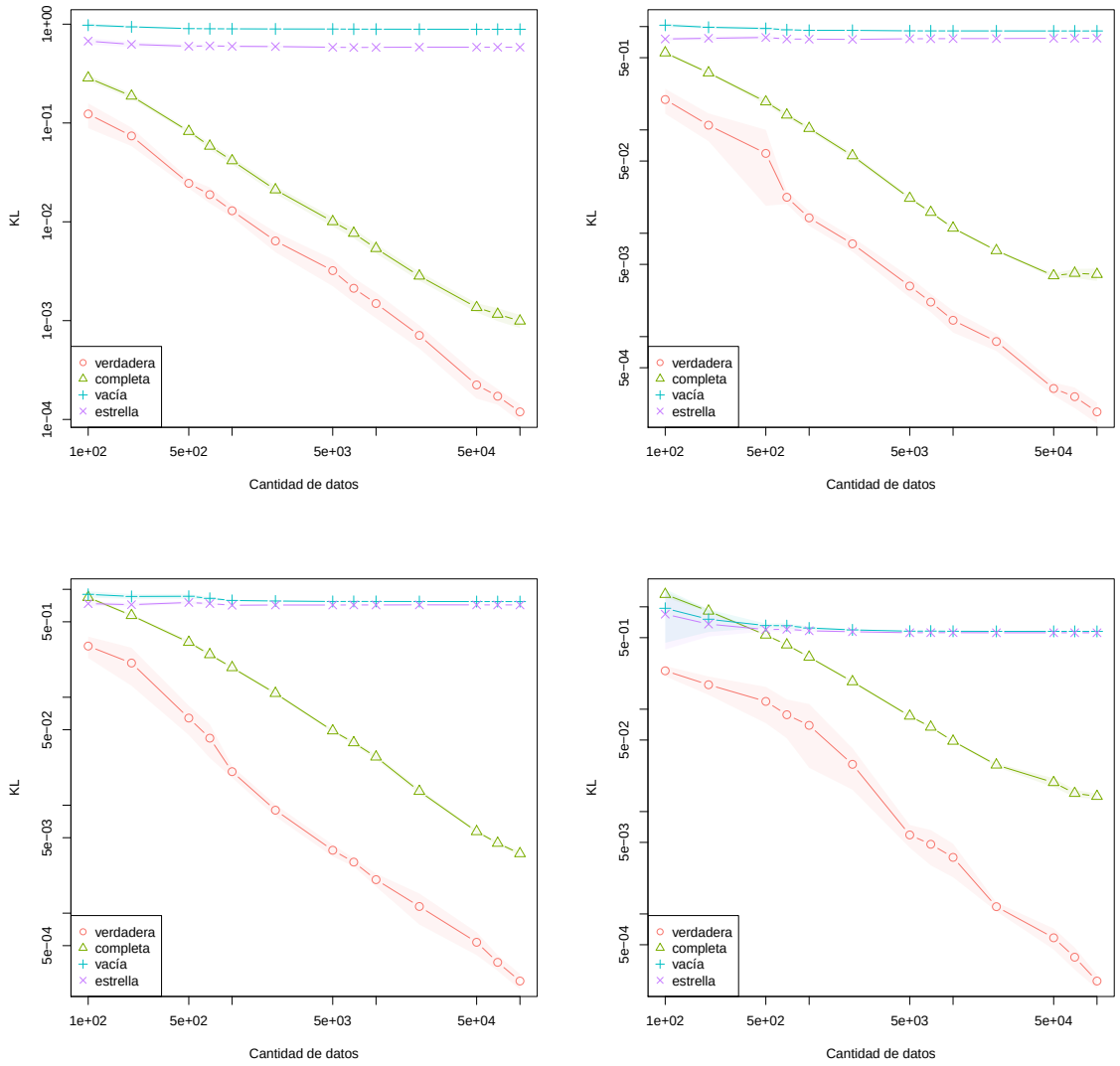


Figura 5-3: Valores de KL (en espacio logarítmico) para diferentes tipos de estructuras y tamaños crecientes de datos. Cada subfigura corresponde a un n: 6 (arriba izquierda), 7 (arriba derecha), 8 (fondo izquierda), y 9 (fondo derecha). Cada punto representa el promedio para 10 conjuntos de datos de tamaño fijo y su sombra la desviación estándar. Valores de KL bajo son mejores.

conjuntos de datos de tamaño fijo, mientras que la sombra de dicha curva representa su desviación estándar. La sombra de una curva fue obtenida de la siguiente manera. En base a las desviaciones estándares asociadas a cada punto de la curva, se construyo un polígono (sombra) que uniera los extremos de cada desviación estándar.

En base a los resultados mostrados en la Figura 5-3, dos importantes tendencias

puede ser destacadas. La primera consiste en que el valor de KL tiende a reducirse cuando la cantidad de datos incrementa. Como se muestra en el Teorema 20.3 [46], la estimación de parámetros tiende a ser más precisa a medida que la cantidad de datos aumenta. Consecuentemente, dada una estructura fija, las redes de Markov obtenidas desde conjuntos de datos con mayor número de observaciones resultan en valores de KL menores. La segunda tendencia es que *el valor de KL está fuertemente sujeto a los errores presentes en una estructura.* Más precisamente, los valores de KL obtenidos por el grupo I-map son menores que los valores obtenidos por el grupo no I-map, donde dicha diferencia se amplia considerablemente a medida que la estimación de parámetros es más precisa (aumenta la disponibilidad de datos). Analicemos en más detalle cada uno de estos grupos.

Dentro del grupo I-map, es interesante observar la diferencia entre los valores de KL obtenidos por las estructuras subyacente y completa. Los valores de KL obtenidos por la estructura subyacente son mucho menores que los valores obtenidos por la estructura completa. Esto se debe a que la estructura completa, al no codificar las independencias específicas del contexto (errores del tipo II), produce parámetros espurios, los cuales requieren mayores cantidades de datos para ser estimados correctamente. En contraste, en el grupo no I-map, los valores de KL alcanzan una cota mínima desde la cual, cualquier incremento en la cantidad de datos, no necesariamente produce una reducción significativa del valor de KL. Esto se debe a que una estructura no I-map (presencia de errores del tipo I) produce una factorización incorrecta de la distribución, eliminando parámetros que luego no pueden ser estimados, produciendo redes de Markov con baja precisión, es decir, no similares a la distribución subyacente.

Para concluir, en base a los resultados obtenidos, remarcaremos los principales puntos sobre los cuales apoyaremos los análisis de las próximas etapas. Primero, los resultados obtenidos muestran una clara correlación entre el valor de la KL y los errores presentes en la estructura de una red de Markov, ofreciéndonos un criterio para seleccionar entre un grupo de redes de Markov, cuál de ellas presenta la estructura más correcta. Segundo, los valores de KL remarcan la importancia de aprender estructuras correctas, estructuras incorrectas producen redes de Markov que divergen de la

distribución subyacente. Tercero, los valores de KL obtenidos por la estructura subyacente muestran la importancia de codificar independencias específicas del contexto, las cuales nos permiten obtener redes de Markov más precisas.

Estructuras aprendidas

Debido al gran número de algoritmos de aprendizaje de estructuras utilizados en la experimentación, los resultados de KL fueron divididos en dos grupos para facilitar su análisis. La Figura 5-4 muestra los valores de KL obtenidos desde las estructuras aprendidas por los algoritmos basados en restricciones, mientras que la Figura 5-5 muestra los valores de KL obtenidos desde las estructuras aprendidas por los algoritmos de estimación de densidad. Para facilitar las comparaciones, en ambas figuras, agregamos los valores de KL obtenidos desde las estructuras aprendidas por CSPC y CSGS. Las Figuras 5-4 y 5-5, junto con sus subfiguras, deben ser interpretadas de la misma forma descripta en la sección de las estructuras de control.

Analizando la Figura 5-4, una de las tendencias más claras es la diferencia que presentan los valores de KL obtenidos por CSPC y CSGS frente a los valores obtenidos por los algoritmos basados en restricciones. Independientemente del número de variables, los valores de KL de ambos tipos de algoritmos se encuentran claramente diferenciadas: los valores de KL obtenidos por CSPC y CSGS son, en la mayoría de los casos, menores a los valores obtenidos por los restantes algoritmos. Particularmente, cuando $|D| > 1k$, los valores de KL obtenidos por CSPC y CSGS resultan ser significativamente más bajos que aquellos obtenidos por los algoritmos basados en restricciones. Esto es porque las curvas de los algoritmos CSPC y CSGS no se solapan con las curvas de los algoritmos basados en restricción. Cuando $|D| < 1k$, las diferencias de los valores de KL entre ambos tipos de algoritmos no resultan ser significativas (i.e., las curvas se solapan).

El hecho de que CSPC y CSGS obtengan valores de KL más bajos que los restantes algoritmos refuerza los resultados analizados en la Sección 5.4.1, donde las estructuras obtenidas por CSPC y CSGS fueron las únicas estructuras que pudieron capturar las independencias específicas del contexto. También es importante notar que todos

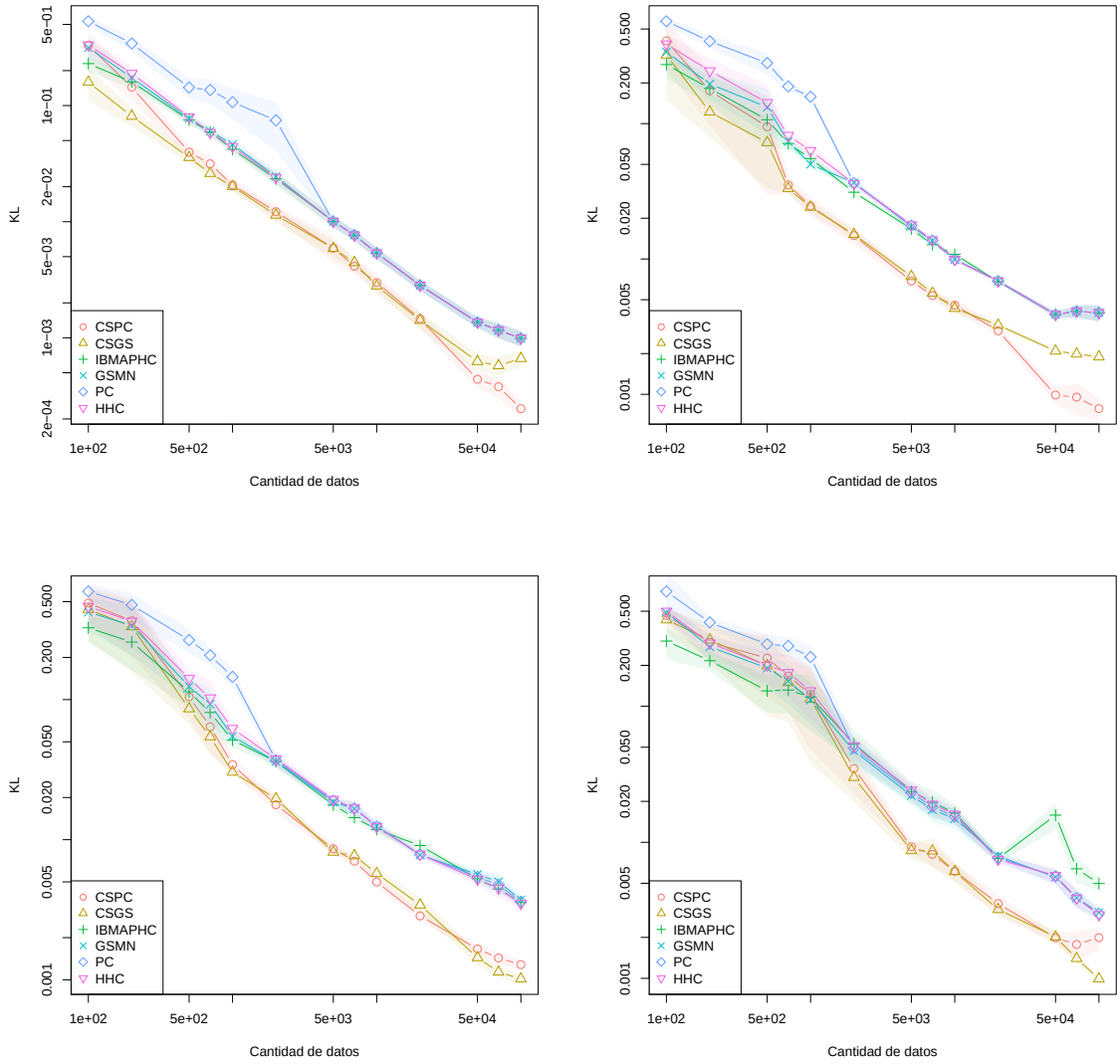


Figura 5-4: Valores de KL (en espacio logarítmico) para diferentes tipos de estructuras y tamaños crecientes de datos. Cada subfigura corresponde a un n: 6 (arriba izquierda), 7 (arriba derecha), 8 (fondo izquierda) y 9 (fondo derecha). Cada punto representa el promedio para 10 conjuntos de datos de tamaño fijo y su sombra la desviación estándar. Valores de KL bajo son mejores.

los algoritmos basados en restricciones obtienen valores de KL bastante semejantes, reforzando el hecho de que todos los algoritmos de aprendizaje de estructuras tienden a construir la misma estructura, i.e. un grafo completo, el cual no captura las independencias específicas del contexto.

Dentro de los algoritmos basados en restricciones, cuando $|D| < 1k$ y $n \geq 8$,

IBMAP-HC obtiene valores promedios de KL más bajos que aquellos obtenidos por los restantes algoritmos. Como mencionamos en la Sección 5.4.1, este patrón se debe a que el diseño del algoritmo IBMAP-HC es bastante diferente a la de los restantes algoritmos basados en restricciones. Sin embargo, al igual que los restantes algoritmos basados en restricciones, las estructuras obtenidas por IBMAP-HC tienden a ser un grafo completo, el cual no codifica independencias específicas del contexto.

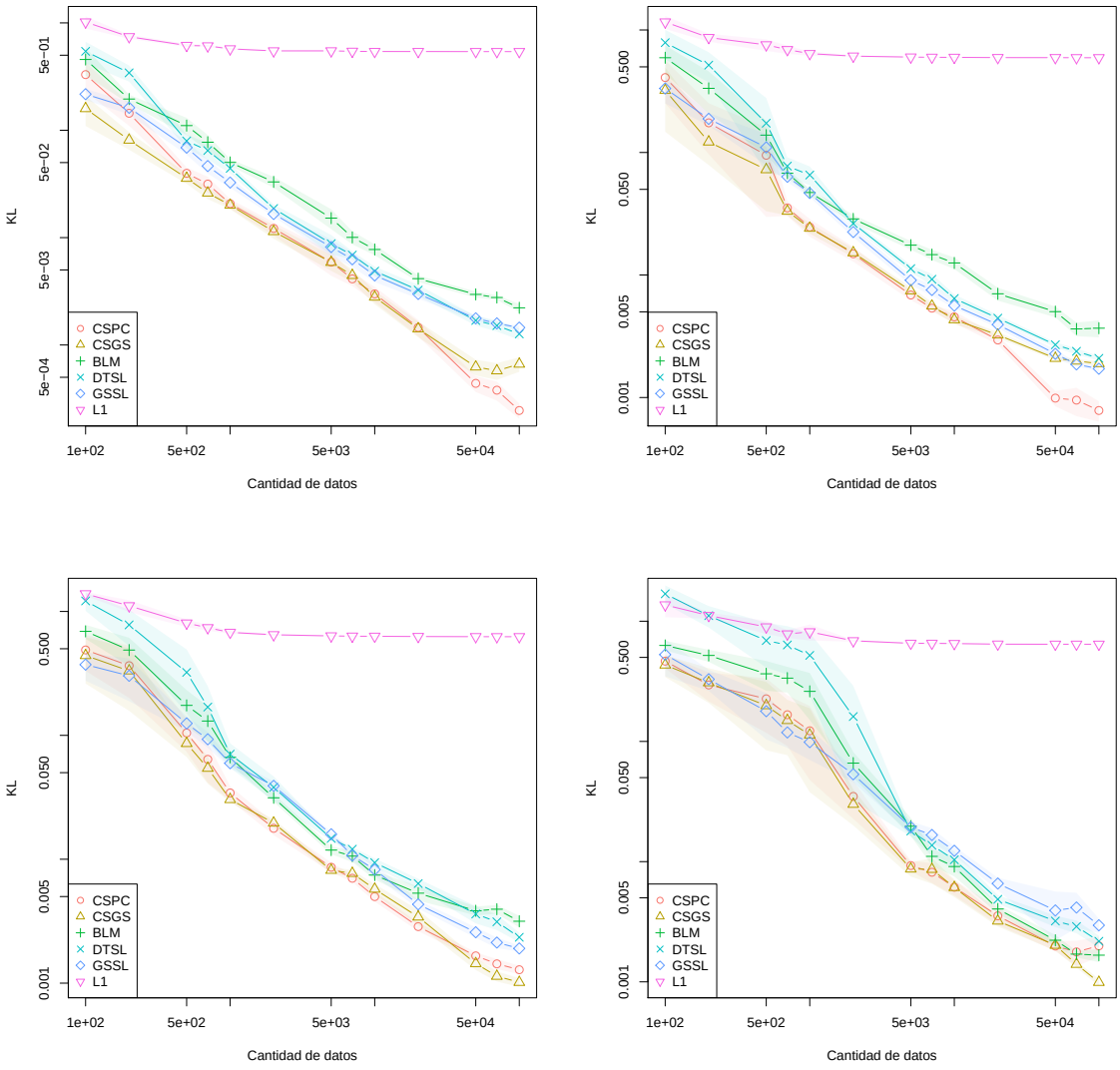


Figura 5-5: Valores de KL (en espacio logarítmico) para diferentes tipos de estructuras y tamaños crecientes de datos. Cada subfigura corresponde a un n: 6 (arriba izquierda), 7 (arriba derecha), 8 (fondo izquierda), y 9 (fondo derecha). Cada punto representa el promedio para 10 conjuntos de datos de tamaño fijo y su sombra la desviación estándar. Valores de KL bajo son mejores.

Ahora, analicemos la Figura 5-5, la cual muestra los valores de KL obtenidos por los algoritmos de estimación de densidad. En primera instancia, al comparar la Figura 5-5 con la Figura 5-4 (la cual muestra los valores de KL obtenidos por los algoritmos basados en restricciones), podemos observar que los valores de KL obtenidos por los algoritmos de estimación de densidad son más bajos en comparación a los obtenidos por los algoritmos basados en restricciones. Por ejemplo, las curvas de KL de los algoritmos de estimación de densidad se encuentran más próximas a las curvas de KL de los algoritmos CSPC y CSGS. Inclusive, hay varias situaciones en que los algoritmos de estimación de densidad alcanzan valores de KL bastante similares a aquellos obtenidos por los algoritmos CSPC y CSGS. Por ejemplo, para $n = 7$ y $|D| > 20k$, los valores de KL obtenidos por DTSL y (especialmente) GSSL son similares a los obtenidos por CSGS. Otro caso ocurre para $n = 9$ y $|D| > 20k$, donde BLM obtiene valores de KL bastante similares a los obtenidos por CSGS.

Esta tendencia de los algoritmos de estimación de densidad es esperable. Justamente dichos algoritmos están diseñados para aprender estructuras que logran codificar complejas interacciones entre los estados de las variables (a través de un conjunto de características), permitiéndoles lograr un mejor ajuste a los conjuntos de datos. Sin embargo, como vimos en la Sección 5.4.1, las estructuras aprendidas por los algoritmos de estimación de densidad no son interpretables. En otras palabras, tal como vimos en la Figura 5-2, los conjuntos de características obtenidos por dichos algoritmos no codificaban correctamente las independencias específicas del contexto.

Por otro lado, dentro de los algoritmos de estimación de densidad, L1 obtiene los peores valores de KL (los valores más altos). La causa de esto puede deberse a dos razones. Primero, L1 puede únicamente aprender características positivas de longitud 2 (ver [78]), las cuales no pueden capturar correctamente las interacciones que existen entre las variables. Segundo, los parámetros de configuración de L1 pueden no ser lo suficientemente adecuados para las distribuciones del escenario sintético. Esta última razón tiene bastante peso, ya que, como veremos en el escenario del mundo real, los resultados obtenidos por L1 son uno de los más competitivos.

Finalmente, algo destacable a resaltar en los resultados de la Figura 5-5 es que los

valores de KL obtenidos por los algoritmos CSPC y CSGS son bastante semejantes a los valores de KL obtenidos por los algoritmos de estimación de densidad, lo cual no sucede con los valores de KL obtenidos por los algoritmos basados en restricciones. Esto se debe principalmente a la representación de la estructura utilizada por CSPC y CSGS, la cual es lo suficientemente flexible para capturar interacciones asimétricas entre las variables (independencias específicas del contexto). Dichas interacciones también pueden ser capturadas por un conjunto de características (utilizada por los algoritmos de estimación de densidad) pero no por un único grafo (utilizada por los algoritmos basados en restricciones).

Por otro lado, según los resultados de la Figura 5-2, los algoritmos CSPC y CSGS fueron los únicos cuya estructura era lo suficientemente interpretable para recuperar las independencias específicas del contexto. Todo estos resultados aportan evidencia a favor de nuestra propuesta para representar la estructura de una distribución (conjunto de grafos canónicos), la cual combina las ventajas/desventajas de los representaciones utilizadas por los algoritmos basados en restricciones (un grafo) y los algoritmos de estimación de densidad (un conjunto de características).

Estructuras control frente a estructuras aprendidas

Esta sección combina los resultados presentados en las últimas dos secciones. En especial, comparamos los valores de KL que se encuentran divididos en diferentes figuras para poder hacer una análisis en conjunto. Para facilitar el análisis actual, hicimos una selección de los resultados presentados previamente. Puntualmente, seleccionamos las siguientes estructuras. Entre las estructuras de control, la estructura subyacente y la estructura completa fueron seleccionadas debido a que los valores de KL obtenidos por todos los algoritmos deberían encontrarse principalmente entre los valores de KL de dichas estructuras de control. Entre los algoritmos basados en restricciones, seleccionamos las estructuras obtenidas por IBMAP-HC, debido a que sus valores de KL resultaron los más competitivos entre los algoritmos basados en restricciones, especialmente en situaciones de baja disponibilidad de datos. Entre los algoritmos de estimación de densidad, seleccionamos las estructuras obtenidas por

GSSL, debido a que en promedio obtuvo los mejores valores de KL entre los algoritmos de estimación de densidad. Finalmente, seleccionamos las estructuras obtenidas por los algoritmos CSPC y CSGS. Los valores de KL de todas estas estructuras son mostrados en la Figura 5-6. Dicha figura tiene que ser interpretada de la misma forma que fue descripta en la primera etapa.

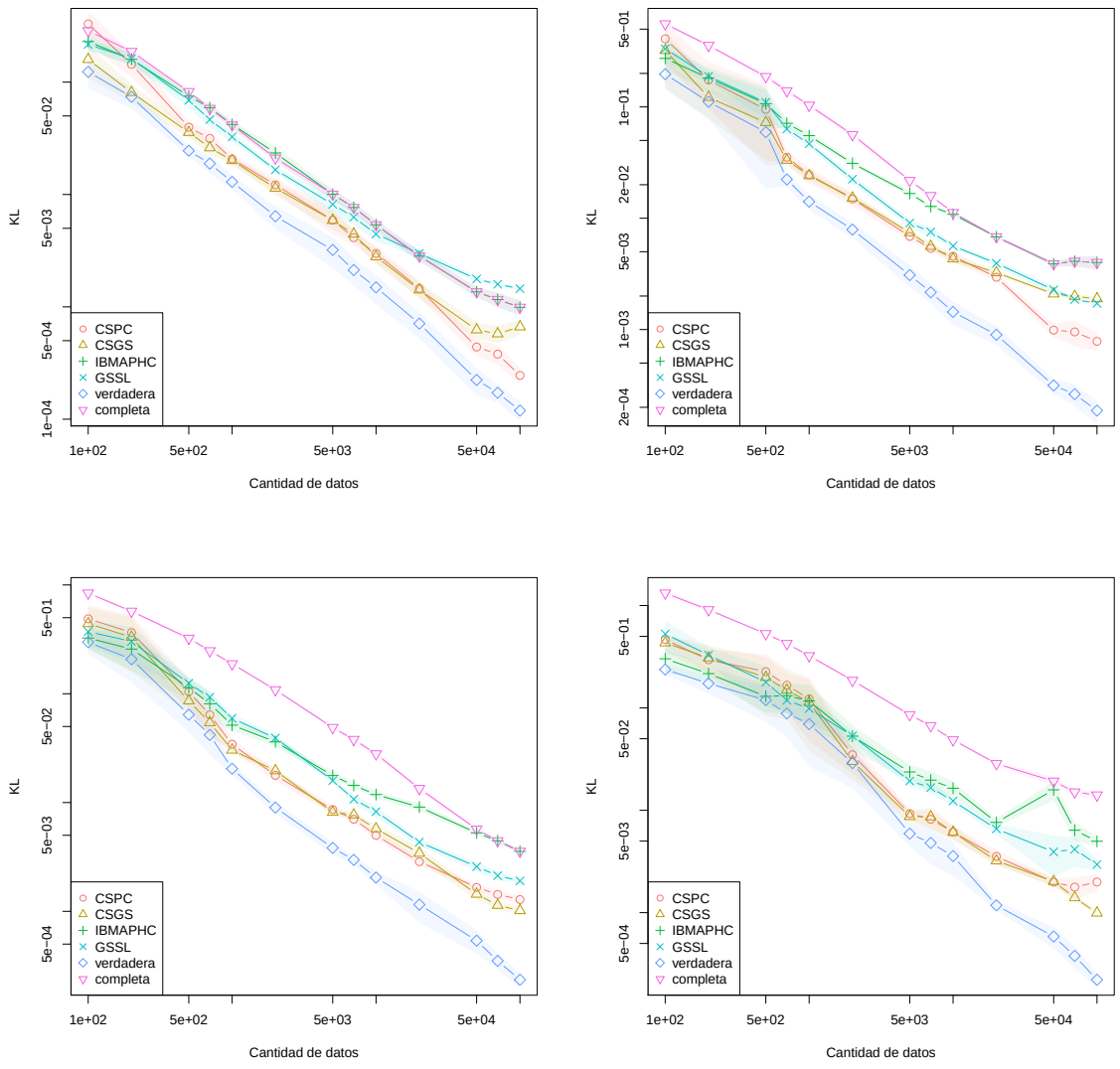


Figura 5-6: Valores de KL (en espacio logarítmico) para diferentes tipos de estructuras y tamaños crecientes de datos. Cada subfigura corresponde a un n: 6 (arriba izquierda), 7 (arriba derecha), 8 (fondo izquierda), y 9 (fondo derecha). Cada punto representa el promedio para 10 conjuntos de datos de tamaño fijo y su sombra la desviación estándar. Valores de KL bajo son mejores.

Al analizar la Figura 5-6, la principal tendencia a remarcar es que las curvas de KL de cualquier algoritmo se encuentran limitadas por la curva de la estructura subyacente (límite inferior) y por la curva de la estructura completa (límite superior). Esto último no se cumple en la curva de GSSL cuando $n = 6$ y hay una alta disponibilidad de datos. Esto se debe a que la estructura aprendida por GSSL presenta errores del tipo I. Pero exceptuando dicho caso, el resto de las curvas se encuentran limitadas por las curvas de la estructura subyacente y la estructura completa.

La segunda tendencia a remarcar es que, en situaciones de alta disponibilidad de datos, las curvas de los algoritmos se diferencian claramente según el tipo de algoritmo, es decir, las curvas obtenidas por nuestro enfoque, el enfoque basado en restricciones y el enfoque de estimación de densidad. El único algoritmo que cambia de tendencia según el valor de n es el algoritmo GSSL. En algunos casos ($n = 7$ y $n = 8$) la curva de GSSL es bastante similar a las curvas obtenidas por CSPC y CSGS. En otros casos, $n = 6$ y $n = 9$, las curvas de GSSL son similares a las curvas de IBMAP-HC.

Por otro lado, para situaciones de alta disponibilidad de datos y $n < 9$, las curvas de IBMAP-HC son significativamente similares a las curvas de la estructura completa. Este resultado es esperable y es acorde a los resultados mostrados en la Figura 5-2, donde las longitudes promedio de las características, inducidas desde las estructuras aprendidas por IBMAP-HC, mostraban un valor que tendía hacia n, significando que la estructura aprendida era un grafo completo.

Por otro lado, como mencionamos más arriba, los valores de GSSL varían considerablemente entre los distintos valores de n. Por ejemplo, para $n = 6$, la curva de GSSL obtiene valores de KL por arriba de la curva de la estructura completa; para $n = 7$, la curva de GSSL es bastante similar a la curva de CSGS; para $n = 8$ y $n = 9$, la curva de GSSL se aleja de las curvas de CSPC y CSGS. La razón de esta variabilidad en las estructuras aprendidas por GSSL se debe en gran parte a que dicho algoritmo es aleatorio, es decir, su lógica se encuentra sujeta a cierta cantidad de aleatoriedad [89].

Finalmente, otra tendencia importante a remarcar es el comportamiento de las curvas de CSPC y CSGS. En comparación a los restantes curvas, las curvas de CSPC y CSGS son las más próximas a las curvas de la estructura subyacente, mostrando las

ventajas de codificar explícitamente las independencias específicas del contexto. Esta tendencia pude ser especialmente apreciada cuando $n = 6$ y $n = 7$. En contraste, para $n = 8$ y $n = 9$, la diferencia entre las curvas de CSPC y CSGS y las curvas de la estructura subyacente son mayores. Esto es debido a que, al aumentar el número de variables, se reduce la disponibilidad de datos. Por ejemplo, 10k observaciones no son lo mismo para un problema con $n = 6$ variables que en un problema con $n = 9$ variables. De esta manera, ante la baja disponibilidad de datos, las estructuras aprendidas por CSPC y CSGS contienen errores. Esto último puede ser apreciado en la Figura 5-2, observando las longitudes promedio obtenidas para los diferentes valores de n. Precisamente, para $n = 8$ y $n = 9$, las diferencias entre las longitudes de $\mathcal{F}_0$ y $\mathcal{F}_1$ no son tan contrastante como sucede en el caso de $n = 6$ y $n = 7$, mostrando que las estructuras aprendidas por CSPC y CSGS contienen errores. Como mencionamos en la Sección 5.4.1, estos errores son causados por los resultados erróneos obtenidos por ejecutar una prueba estadística ante baja disponibilidad de datos.

5.4.3. Número de decisiones

La Figura 5-7 muestra el número de decisiones tomadas por CSPC y CSGS para construir las estructuras del escenario sintético. Cada subfigura corresponde a un valor de n: 6 (arriba izquierda), 7 (arriba derecha), 8 (fondo izquierda) y 9 (fondo derecha). En cada subfigura, cada valor del eje X tiene dos barras, las cuales toman diferentes valores en el eje Y. Dado un par de barras, la primer barra está asociada a CSPC, mientras que la otra barra está asociada a CSGS. Una barra representa el número promedio de pruebas ejecutadas por un algoritmo para construir una estructura desde 10 conjuntos de datos diferentes de tamaño fijo.

Al comparar el número de decisiones tomadas por ambos algoritmos, podemos observar que CSPC toma un número exponencial de decisiones en comparación a CSGS. Estas diferencias resultan más acentuadas a medida que aumenta el número de variables. Precisamente, estos resultados se corresponde con el análisis de complejidad temporal presentado en la Sección 4.4. En dicha sección se mostró teóricamente que el algoritmo CSPC toma un número exponencial de decisiones con respecto al número

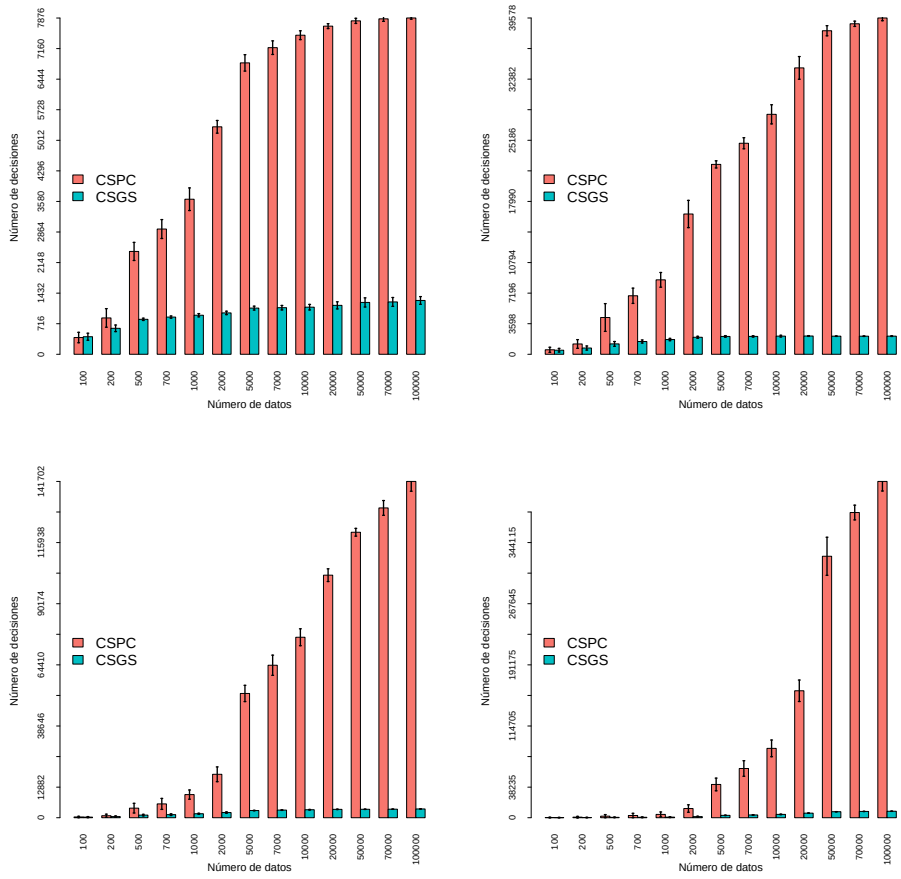


Figura 5-7: Número de pruebas ejecutadas por CSPC y CSGS para constuir las estructuras del escenario sintético. Cada barra representa el promedio y desviación estándar del número de pruebas involucradas para un conjunto de datos con tamaño fijo.

n de variables, mientras que CSGS toma un número de decisiones exponencialmente menor. En base a los resultados mostrados en la sección anterior, podemos decir que CSGS obtiene resultados similares a CSPC con un costo temporal significativamente menor.

5.5. Resultados en el escenario del mundo real

El Cuadro 5.5 muestra los valores de NCMLL obtenidos por todos los algoritmos de aprendizaje en los 18 conjuntos de datos. Para cada conjunto de datos, el valor en negrita indica el mejor valor de NCMLL obtenido. Las celdas vacías representan casos donde el valor de NCMLL no pudo ser computado debido a la ausencia de

una estructura. La razón de esto último fue que, en dichos casos, los algoritmos de aprendizaje consumieron una excesiva cantidad de memoria RAM (superaron el límite físico disponible de 12GB), lanzando una excepción durante su ejecución.

Por otro lado, según lo mostrado en la Sección 5.4.3, CSPC tiene una complejidad temporal exponencial con respecto al número n de variables. Debido a que los valores de n son estrictamente más elevados que los presentados en el escenario sintético, CSPC debió ser ejecutado utilizando la variable umbral $K_{\max}$ (ver Sección 2.4.3). El Cuadro 5.6 muestra los valores de $K_{\max}$ utilizados para cada conjunto de datos.

Conjunto D	CSPC	CSGS	BLM	DTSL	L1	GSSL	GSMN	HHC-MN	PC	IBMAP
NLTCS	-.3109	**-.3106**	-.3140	-.3118	-.3280	-.3156	-.3132	-.3145	-.3142	-.3110
MSNBC	-.3524	-.3420	-.3418	**-.3325**	-.3676	-.3423	-.3398	-.3475		
Abalone	-.0882	-.3065	-.2666	-.1049	**-.0639**	-.2400	-.1859	-.1733	-.3305	-.1982
Wine	-.1513	-.1851	-.5266	-.2982	**-.1430**	-.4760	-.2978	-.2992	-.3296	-.5182
KDD2000	-.0318	-.0318	-.0773	**-.0315**	-.0409	-.0321	-.0323	-.0324		-.0322
Plants	-.1431	**-.1338**	-.1426	-.1430	-.1473	-.1374	-.1439	-.1483	-.1486	-.1419
Covertype	-.1448	-.2209	-.2990	-.2841	**-.0736**	-.2202	-.3772	-.3586	-.4971	-.3350
Audio	-.3713	**-.3664**	-.4073	-.3737	-.3702	-.3684	-.3787	-.3804		-.3810
Netflix	-.5270	-.5205	-.6371	-.5309	-.5216	**-.5204**	-.5389	-.5395		-.5462
Accidents	-.2047	-.1998	-.4109	-.1888	**-.1826**	-.2690	-.2069	-.2348		-.2570
Adult	-.1325	-.1367	-.1392	-.2608	**-.0648**	-.1337	-.1826	-.1320		
Retail	-.0772	**-.0769**	-.0779	-.0773	-.0779	-.0774	-.0776	-.0778	-.0782	-.0782
Pumsb Star	-.0876	-.0913	-.4788	-.0919	**-.0787**	-.1342	-.1171	-.1315		-.1203
DNA	-.3859	-.3849	-.5535	-.3829	**-.3811**	-.5310	-.3852	-.3868	-.3885	-.4142
MSWeb	-.0313	**-.0310**	-.0317	-.0565	-.0319	-.0313	-.0315	-.0316	-.0313	
WebKB	-.1712	-.1694	-.2162	-.1784	-.1717	-.1742	**-.1690**	-.1697		
BBC	-.2261	**-.2221**	-.2594	-.2372	-.2271	-.2279	-.2341	-.2355	-.2434	
Ad	-.0087	-.0095	-.0329	-.0095	-.0085	-.0090	**-.0080**	-.0085	-.0258	

Cuadro 5.5: Resultados de NCMLL. Un valor alto indica mejor precisión predictiva. Para cada conjunto de datos, el mejor valor de NCMLL obtenido es mostrado en negrita. En las celdas en blanco, el valor de NCMLL no pudo ser computado.

Aunque muchos de los resultados individuales mostrados en el Cuadro 5.5 son interesantes en sí mismos, presentaremos una análisis general a través de dos medidas comparativas. La primera es el porcentaje de mejores/peores valores de NCMLL obtenidos entre cada par de algoritmos. La segunda es la utilización de la *prueba de los rangos con signo de Wilcoxon* (*Wilcoxon signed-rank test*) con corrección de continuidad, cuyos resultados son mostrados en el Cuadro 5.8. Esta es una prueba estadística de diferencia de pares que nos permite determinar si la diferencia entre los valores de NCMLL obtenidos por un par de algoritmos es significativa. La prueba de Wilcoxon es una prueba generalmente utilizada como una medida para comparar algoritmos de aprendizaje de estructuras [18, 57, 89, 56]. En lo que sigue, analizaremos en detalle

Conjunto D	$K_{\max}$	Conjunto D	$K_{\max}$
NLTCS	-	Accidents	-
MSNBC	-	Adult	4
Abalone	-	Retail	6
Wine	-	Pumsb Star	4
KDDCup	4	DNA	-
Plants	6	MSWeb	4
Covertype	4	WebKB	3
Audio	3	BBC	2
Netflix	5	Ad	4

Cuadro 5.6: Valores de $K_{\max}$ utilizados por CSPC. Un guión indica que CSPC no utilizó un valor de corte.

los resultados obtenidos en ambas comparaciones.

El Cuadro 5.7 muestra el número de veces que, al comparar los valores de NCMLL obtenidos por un par de algoritmos, uno obtiene mejor valor de NCMLL frente al otro (sin contar empates). Dividiendo este valor por el número de comparaciones, obtenemos un valor normalizado (entre 0 y 1). Para los valores de NCMLL perdidos (para PC y IBMAP-HC), realizamos las comparaciones únicamente teniendo en cuenta los valores disponibles. Estos valores están dispuestos en el Cuadro 5.7 de la siguiente manera. Por ejemplo, la primera fila muestra el porcentaje de veces que CSPC obtuvo mejores valores de NCMLL frente a los restantes algoritmos (uno por columna). En contraste, la primera columna muestra el porcentaje de veces que CSPC obtuvo peores valores de NCMLL frente a los restantes algoritmos (uno por fila). Para cada columna, marcamos en negrita el porcentaje más alto.

Analizando los valores en el Cuadro 5.7, CSGS muestra el promedio más bajo de peores valores de NCMLL, 19 %, seguido por CSPC y L1 con 23 % y 32 %, respectivamente. Dentro de los algoritmos basados en restricciones, el algoritmo PC es el que obtuvo, en promedio, los peores valores de NCMLL. Por el otro lado, en el caso de los algoritmos de estimación de densidad, el algoritmo BLM obtuvo, en promedio, los peores valores de NCMLL. Analizando pares de algoritmos, observamos que al comparar CSPC y CSGS con el resto de los algoritmos basados en restricciones, CSPC y CSGS obtienen mejores valores de NCMLL frente a todos los algoritmos. Por ejemplo,

	CSPC	CSGS	BLM	DTSL	L1	GSSL	GSMN	HHC	PC	IBMAP
CSPC	0.00	0.33	**0.89**	**0.72**	0.44	**0.72**	**0.78**	0.78	0.90	**0.92**
CSGS	**0.61**	0.00	**0.89**	0.67	**0.56**	**0.72**	**0.78**	**0.83**	**1.00**	**0.92**
BLM	0.11	0.11	0.00	0.17	0.22	0.11	0.17	0.22	0.50	0.17
DTSL	0.28	0.28	0.83	0.00	0.28	0.50	0.67	0.72	0.90	0.83
L1	0.56	**0.44**	0.72	**0.72**	0.00	0.56	0.56	0.61	0.80	0.75
GSSL	0.22	0.28	**0.89**	0.50	0.44	0.00	0.50	0.50	0.60	0.58
GSMN	0.22	0.22	0.83	0.33	0.44	0.50	0.00	**0.83**	0.90	0.67
HHC-MN	0.22	0.17	0.78	0.28	0.33	0.50	0.17	0.00	0.80	0.58
PC	0.00	0.00	0.50	0.10	0.20	0.30	0.10	0.20	0.00	0.29
IBMAP-HC	0.08	0.08	0.83	0.17	0.25	0.42	0.33	0.42	0.57	0.00
Promedio	0.23	**0.19**	0.72	0.37	0.32	0.43	0.41	0.51	0.70	0.57

Cuadro 5.7: Porcentajes de mejores/peores valores de NCMLL entre cada par de algoritmos. En una columna, el valor más alto es remarcado en negrita. El promedio más bajo es remarcado en negrita.

CSGS obtiene mejores resultados que PC en un 100 % de las veces. Por otro lado, los algoritmos de estimación de densidad obtienen NCMLL más competitivas. Por ejemplo, CSPC y CSGS obtienen mejores NCMLL que BLM y GSSL, en un 89 % y 72 % de las veces, respectivamente. Los algoritmos más competitivos son DTSL y L1. DTSL es superado por CSPC y CSGS sólo en un 72 % y 67 % de las veces, mientras que L1 es sólo superado un 44 % y 56 % de las veces. Por último, al comparar CSGS frente a CSPC, encontramos que CSGS obtiene mejores valores de NCMLL el 61 % de las veces.

Del análisis anterior podemos concluir lo siguiente. En promedio, CSPC y CSGS obtienen mejores valores de NCMLL frente a todos los algoritmos basados en restricciones aproximadamente el 84 % de las veces. En contraste, frente a todos los algoritmos de estimación de densidad, CSPC y CSGS obtienen, en promedio, mejores valores de NCMLL aproximadamente el 70 % de las veces. De estas conclusiones, podemos remarcar dos cosas. Primero que nuestro enfoque logra obtener redes de Markov más precisas en comparación a los algoritmos basados en restricción. Segundo, los algoritmos de estimación de densidad (tal como vimos en los resultados de KL) resultan ser más competitivos que los algoritmos basados en restricción.

Para computar la prueba de los rangos con signo de Wilcoxon, hicimos lo siguiente. Primero, hay que tener en cuenta que los 18 conjuntos de datos pueden verse como una muestra de tamaño 18 tomada desde una población desconocida de conjuntos

de datos. Segundo, cada uno de los 18 conjuntos, tiene asociado un par de valores de NCMLL, uno perteneciente a nuestros algoritmos (CSPC o CSGS) y el otro para cualquiera de los restantes algoritmos de aprendizaje. Dado que el valor de NCMLL representa la precisión de un algoritmo en un conjunto de datos en particular, estamos interesados en probar la hipótesis de que los valores de NCMLL obtenidos por nuestros algoritmos son mayores (más precisos) a los valores de NCMLL obtenidos por los restantes algoritmos con un nivel de confianza del 95 % (o un 0.05 de nivel de significancia). Para decidir si aceptar o rechazar esta hipótesis, utilizamos los valores p obtenidos por la prueba de los rangos con signo de Wilcoxon.

	CSPC	CSGS	BLM	DTSL	L1	GSSL	GSMN	HHC	PC	IBMAP
CSPC	-	0.665	9.537e-05	0.019	0.868	0.029	0.006	0.001	0.005	0.002
CSGS	0.352	-	0.0001	0.059	0.695	0.019	0.010	0.007	0.003	0.010

Cuadro 5.8: Valores p obtenidos al comparar los valores de NCMLL entre CSPC (y CSGS) frente a los restantes algoritmos de aprendizaje.

El Cuadro 5.8 muestra los valores p al comparar CSPC (y CSGS) frente a cada uno de los restantes algoritmos de aprendizaje. Desde dichos valores, podemos extraer las siguientes conclusiones. En general, frente a la mayoría de los algoritmos, CSPC y CSGS obtienen valores p menores al nivel de significancia, excepto en los siguientes casos. El primero surge del valor p asociado al par CSPC-L1, el cual no es menor que el nivel de significancia, por lo tanto, no hay suficiente evidencia para afirmar que CSGS es más preciso que L1. Los otros dos casos surgen al analizar los valores p asociados a los pares CSGS-DTSL y CSGS-L1, es decir, en dichos casos, no tenemos suficiente evidencia para afirmar que CSGS es significativamente más preciso que DTSL y L1. Sin embargo, CSPC resulta ser significativamente más preciso que DTSL.

Por otro lado, evaluamos si las diferencias en los valores de NCMLL obtenidos por CSPC y CSGS son significativas. Para tal caso, tenemos un valor p de 0.7049. Debido a que 0.7049 no es menor al nivel de significancia, entonces podemos concluir que las diferencias entre los valores de NCMLL obtenidos por CSPC y CSGS no son significativas. Este hecho, sumado a lo discutido en la Sección 5.4.3, muestran que las estructuras obtenidas por CSGS son equivalentes en precisión a las obteni-

das por CSPC, pero CSGS consume menos tiempo para construirlas. De este modo, CSGS resulta una alternativa más eficiente en comparación a CSPC para aprender la estructura de una red de Markov.

Finalmente, podemos extraer las siguientes conclusiones globales. Con respecto a los algoritmos basados en restricciones, los valores p obtenidos por cada uno de estos algoritmos y CSPC y CSGS resultan ser menores al nivel de significancia, entonces podemos concluir que nuestros algoritmos son significativamente más precisos que todos los algoritmos basados en restricciones utilizados en la experimentación. En contraste, al analizar los valores p obtenidos por los algoritmos de estimación de densidad, no podemos concluir que nuestros algoritmos son significativamente más precisos que los algoritmos de estimación de densidad. Como analizamos más arriba, los valores p asociados a los algoritmos L1 y DTSL no son menores al nivel de significancia. Sin embargo, podemos concluir que nuestros algoritmos resultan ser competitivos frente a los algoritmos de estimación de densidad.

5.6. Resumen

En este capítulo presentamos un experimento empírico para evaluar nuestra propuesta para aprender la estructura de una red de Markov. Concretamente, el experimento se diseño para evaluar el desempeño de los algoritmos CSPC y CSGS frente a otros algoritmos de aprendizaje de estructuras de redes de Markov del estado del arte.

El principal objetivo de la experimentación consistió en evaluar la exactitud de las estructuras aprendidas por los diferentes algoritmos de aprendizaje para determinar si nuestro enfoque puede obtener estructuras más exactas frente a los restantes algoritmos.

Para realizar una comparación lo más representativa posible, se seleccionaron algoritmos de aprendizaje de estructuras del enfoque basados en restricciones y del enfoque de estimación de densidad. Para el enfoque, los algoritmos PC, GSMN, IBMAP-HC, y HHC fueron seleccionados. Para el segundo enfoque, los algoritmos BLM, GSSL,

DTSL, y L1 fueron seleccionados.

Los algoritmos de aprendizaje fueron evaluados en dos escenarios de aprendizaje. Cada escenario de aprendizaje involucró conjuntos de datos originados desde dos fuentes diferentes. En un escenario, llamado el *escenario sintético*, los conjuntos de datos fueron obtenidos desde distribuciones totalmente conocidas. Particularmente, distribuciones que presentan independencias específicas del contexto. En el otro escenario, llamado el *escenario del mundo real*, los conjuntos de datos fueron obtenidos desde un amplio número de fenómenos reales cuyas distribuciones subyacentes son totalmente desconocidas.

Para evaluar la estructura aprendida por un algoritmo, se tuvo en cuenta el escenario del aprendizaje. En el escenario sintético, al ser conocida la distribución subyacente que generó lo datos, dos evaluaciones fueron realizadas. Primero se comparó una estructura aprendida con la estructura subyacente. Segundo, usando una estructura aprendida, se construyó una red de Markov (estimando parámetros). Luego, la red de Markov aprendida se comparó con la distribución subyacente.

Sin embargo, en el escenario del mundo real, debido a que la distribución subyacente es desconocida, la estructura aprendida por un algoritmo no puede ser directamente comparada con la distribución subyacente. Por esta razón, utilizamos *holdout testing*. Esta técnica consiste en dividir un conjunto de datos en dos subconjuntos disjuntos: un conjunto de entrenamiento y un conjunto de prueba. El conjunto de entrenamiento es utilizado para aprender una estructura desde la cual se construye una red de Markov. Luego, usando la red de Markov construida, se determina la probabilidad de generar el conjunto de prueba con dicha red.

En el escenario sintético, los resultados obtenidos mostraron que los algoritmos CSPC y CSGS obtuvieron estructuras más exactas en comparación a todos los restantes algoritmos. Al comparar las estructuras aprendidas con las estructuras subyacentes, los algoritmos CSPC y CSGS fueron los únicos que pudieron aprender las independencias específicas del contexto. Por otro lado, al comparar las redes de Markov aprendidas con las distribuciones subyacentes, nuestros algoritmos obtuvieron redes de Markov más similares a la distribución subyacente. Entre los restantes algo-

ritmos, los algoritmos de estimación de densidad fueron los que más se acercaron a nuestros algoritmos.

En el escenario del mundo real, los resultados obtenidos mostraron que nuestros algoritmos fueron significativamente mejores que los algoritmos basados en restricciones. Sin embargo, en comparación a los algoritmos de estimación de densidad, nuestros algoritmos no fueron significativamente mejores. Más concretamente, hubo solamente una mejor significativa frente a los algoritmos BLM y GSSL.

Los resultados obtenidos de nuestro experimento sugieren que las estructuras aprendidas por nuestro enfoque son más exactas que las obtenidas por los algoritmos basados en restricciones y resultan ser bastante competitivas frente a las estructuras obtenidas por los algoritmos de estimación de densidad.

Capítulo 6

Epílogo

La ciencia es una esfera finita que crece en el espacio infinito; cada nueva expansión le hace comprender una zona mayor de lo desconocido, pero lo desconocido es inagotable.

Jorge Luis Borges.

En un gran número de problemas en la práctica, las distribuciones de probabilidad son una excelente alternativa para modelar un amplio espectro de fenómenos físicos. Quizás, una de las razones de esto último es porque una distribución puede codificar *incertidumbre*. Consecuentemente, al definir una distribución sobre un dominio[1], esta puede ser utilizada, por ejemplo, para efectuar razonamiento bajo incertidumbre.

Una manera de representar una distribución de probabilidad es por medio de un modelo paramétrico, el cual esencialmente involucra dos elementos: una estructura y un conjunto de parámetros. De estos dos elementos, la estructura cumple un rol doble. Por un lado, la estructura restringe la dimensionalidad del espacio de parámetros; limitando el conjunto de parámetros posibles para una distribución. Por otro lado, la estructura codifica interacciones (o relaciones) estadísticas entre parámetros. Como resultado, la estructura es usualmente definida como un medio para codificar independencias estadísticas entre las variables de un dominio.

[1]Un dominio es un conjunto de eventos, el cual puede ser caracterizado a través de un conjunto de variables aleatorias.

Según el tipo de independencias asumidas, podemos encontrar, hablando aproximadamente, dos modelos paramétricos ampliamente utilizados para representar una distribución de probabilidad: las *redes de Markov* y las *redes de Bayes*. En términos generales, la diferencia crucial entre ambos modelos se encuentra en cómo es representada la estructura, debido a que dicha representación impone restricciones sobre cuáles independencias pueden ser codificadas. Por ejemplo, las redes de Markov asumen independencias que surgen desde interacciones espaciales entre elementos.

Desafortunadamente, en la gran mayoría de los fenómenos físicos, sus distribuciones de probabilidad son generalmente desconocidas, o, sumamente complejas de construir por expertos humanos. Para hacer frente a este problema, se diseñan algoritmos de aprendizaje de máquinas cuyo objetivo es aprender/construir distribuciones de probabilidad automáticamente. De esta manera, tomando un conjunto de muestras/observaciones (conjunto de datos) desde un fenómeno de interés, es posible diseñar un algoritmo que tome dichas muestras y construya una distribución, donde dicha distribución puede pensarse como una "explicación" del conjunto de muestras, y por lo tanto, del fenómeno subyacente que las generó.

Usualmente, estos algoritmos descomponen el problema de aprendizaje de distribuciones en dos subproblemas menores. Un problema es el aprendizaje de la estructura. El otro problema es la estimación de parámetros dada una estructura conocida. En comparación al primero, el segundo problema a recibido considerable interés en estadística, el cual es usualmente conocido como *point estimation*. En contraste, el primer problema a recibido más atención dentro de la comunidad de aprendizaje de máquinas, y por tanto, es un problema relativamente nuevo en comparación al primero. O, al menos, en comparación a la estimación de parámetros, el problema de aprendizaje de estructuras a recibido más interés en estos últimos años.

Dentro del aprendizaje de máquinas, el aprendizaje de estructuras es abordado por los así llamados "algoritmos de aprendizaje de estructuras". Existe una muy amplia familia de algoritmos de aprendizaje de estructuras. Quizás algunos de los más conocidos son los *algoritmos basados en restricciones*. Estos algoritmos se destacan por dos características. Primero, utilizan un grafo no dirigido como representación

de la estructura. Segundo, para construir un grafo no dirigido desde un conjunto de observaciones, utilizan como elemento clave una prueba estadística de independencia.

En términos sencillos, para obtener una estructura desde un conjunto de datos, un algoritmo basado en restricciones construye un grafo. Para esto, comenzando desde un grafo inicial, el algoritmo propone una (in)dependencia utilizando como base el grafo actual. Utilizando una prueba estadística de independencia, el algoritmo decide si la (in)dependencia está (o no) presente en el conjunto de datos. Finalmente, la presencia (o ausencia) de la (in)dependencia es entonces codificada en la topología del grafo, e.g., agregando o removiendo aristas. Hasta que no es alcanzada cierta condición de terminación, el proceso anteriormente descripto se vuelve a repetir.

Un grafo no dirigido puede verse como una potente estructura de datos que permite codificar compactamente un número exponencial de independencias, las cuales pueden ser recuperadas (desde el grafo) en tiempo polinomial. Sin embargo, un grafo no dirigido está limitado en el rango de posibles estructuras que puede codificar. En consecuencia, un grafo no dirigido se encuentra limitado en cuanto al tipo de independencias que puede codificar. Concretamente, un grafo no dirigido puede únicamente codificar las bien conocidas *independencias condicionales*.

Sin embargo, una distribución de probabilidad puede presentar diferentes tipos de independencias, no necesariamente independencias condicionales. Por ejemplo, una de tales independencias son las *independencias específicas del contexto*. Dichas independencias pueden pensarse como independencias condicionales, con la diferencia que una independencia específica del contexto es únicamente valida para determinados contextos[2]. De este modo, una independencia específica del contexto puede ser válida bajo un contexto, pero no en otro; mientras que esto no sucede con las independencias condicionales.

Debido a que un grafo no dirigido no puede codificar independencias específicas del contexto, entonces al utilizar un grafo como representación de la estructura, los algoritmos basados en restricciones están forzosamente limitados en la precisión de las estructuras que pueden aprender. En otras palabras, un algoritmo basado en

[2]Un contexto es cuando un subconjunto de las variables del dominio toma un estado.

restricciones no puede aprender estructuras que presenten independencias específicas del contexto.

No codificar independencias específicas del contexto trae aparejado dos importantes desventajas. Por un lado, dichas independencias resultan significativas en aquellos casos donde un experto humano analiza la estructura en busca de interacciones de interés en un dominio. Segundo, dichas independencias permiten reducir el número de parámetros de un modelo, mejorando así la precisión de la estimación de sus parámetros, resultando en modelos más precisos.

En este trabajo hemos discutido una propuesta para aprender estructuras de redes de Markov, la cual permite superar la limitación de los algoritmos basados en restricciones ante las independencias específicas del contexto. Para superar esta limitación, propusimos una representación alternativa para una red de Markov, la cual denominamos *modelo canónico*. Para definir un modelo canónico, utilizamos como base un modelo que en estadística es llamado *modelo CSI*[3] [43].

Adicionalmente, propusimos una forma novedosa para representar la estructura de un modelo canónico. Más concretamente, la estructura de un modelo canónico puede ser representada a través de un conjunto de *grafos canónicos*, donde un grafo canónico es un grafo no dirigido asociado a un *contexto canónico*[4]. Utilizando un conjunto de grafos canónicos, demostramos que es posible codificar independencias específicas del contexto. Por otro lado, demostramos que toda independencia condicional puede ser expresada como una conjunción de independencias específicas del contexto. En consecuencia, un conjunto de grafos canónicos pueden incluso codificar independencias condicionales.

Junto a esta nueva propuesta para representar la estructura de una red de Markov, mostramos que un grafo canónico puede ser construido de forma similar a un grafo común, es decir, utilizando una prueba estadística. Debido a que las pruebas estadísticas están originalmente diseñadas para identificar la presencia (o ausencia) de *independencias condicionales* en una muestra, propusimos un forma para modificar

[3]Los modelos CSI son una clase general de modelos gráficos que permiten representar gráficamente un modelo log-linear.

[4]Un contexto es canónico cuando todas las variables del dominio tienen un estado asignado.

una prueba estadística, permitiendo la identificación de independencias específicas del contexto. En términos muy sencillos, la modificación consiste en asociar a una prueba estadística un contexto canónico, de este modo, cada vez que la prueba estadística determinar la presencia de una independencia, en realidad está determinando la presencia de una independencia asociada al dicho contexto canónico.

Por lo tanto, los modelos canónicos presentan dos interesantes características. Primero, la estructura de un modelo canónico está formado por un conjunto de grafos no dirigidos (asociados a contextos). Segundo, estos grafos no dirigidos pueden ser construidos utilizando una prueba estadística de independencia (modificada). En base a estas dos características, propusimos un enfoque para construir un conjunto de grafos canónicos desde un conjunto de datos. Este enfoque puede ser dividido en dos partes: i) la definición de un conjunto de contextos canónicos; y ii) la construcción de un grafo canónico por cada contexto en el conjunto de contextos canónicos.

El punto clave de este enfoque es que la construcción de cada grafo canónico puede ser realizada *utilizando cualquier algoritmo basado en restricciones*. Para lograr que un algoritmo basado en restricciones construya un grafo canónico, este simplemente tiene que utilizar la prueba estadística modificada. Usando el enfoque previamente mencionado, presentamos dos algoritmos para aprender la estructura de un modelo canónico: los algoritmos CSPC y CSGS. Ambos algoritmos aprenden un conjunto de grafos canónicos, donde cada grafo canónico es construido por los algoritmos PC y GS, respectivamente; dos algoritmos basados en restricciones ampliamente conocidos en la literatura [87, 61].

Para determinar las ventajas prácticas del uso de los algoritmos CSPC y CSGS para el problema de aprendizaje de estructuras, realizamos una comparación frente a varios algoritmos del estado del arte. Esta evaluación fue realizada en dos escenarios de experimentación: un escenario sintético y un escenario del mundo real. En el escenario sintético, los resultados obtenidos por CSPC y CSGS fueron mejores que aquellos obtenidos por el resto de los algoritmos. En el escenario del mundo real, las estructuras obtenidas por los diferentes algoritmos de aprendizaje de estructuras fueron utilizadas para construir distribuciones. Luego, utilizando dichas distribuciones,

se midió su precisión por medio de *tareas predictivas*, es decir, utilizar la distribución para determinar la probabilidad de ciertos eventos. Nuestros resultados mostraron que las distribuciones obtenidas desde las estructuras aprendidas por CSPC y CSGS obtuvieron mejores precisiones en comparación a las estructuras obtenidas por todos los algoritmos basados en restricciones, alcanzando incluso resultados similares a los obtenidos por algoritmos especialmente diseñados para tareas predictivas.

Sin embargo, en base a nuestros resultados experimentales, podemos señalar dos importantes desventajas que presentan los algoritmos CSPC y CSGS. Primero, debido a que la construcción de la estructura implica la construcción de varios grafos, la complejidad temporal de ambos algoritmos es mayor a la complejidad temporal del resto de los algoritmos evaluados. Segundo, al modificar una prueba estadística para identificar independencias específicas del contexto, la prueba pierde *poder estadístico* (o *sensibilidad*), es decir, la habilidad de la prueba estadística para detectar una (in)dependencia, cuando dicha (in)dependencia realmente existe, se reduce. Esto se debe a que, al utilizar un contexto para identificar una independencia, el tamaño de la muestra se reduce.

En base a estos problemas, la próxima sección presenta algunas posibles vías para encontrar una solución.

6.1. Trabajo futuro

En esta sección mostraremos algunas direcciones de trabajo futuro para superar las desventajas que presentan los algoritmos CSPC y CSGS. Dos direcciones que resultan prometedoras son: i) la reducción de la complejidad temporal de ambos algoritmos; y ii) la mejora en el poder estadístico para identificar independencias específicas del contexto. Para reducir la complejidad temporal, una estrategia sería reducir el número de pruebas estadísticas ejecutadas durante la construcción de la estructura de un modelo canónico, debido a que justamente la operación más costosa en los algoritmos CSPC y CSGS es la ejecución de una prueba estadística.

Para tener una estimación del número de pruebas de independencias involucradas

en la ejecución de los algoritmos CSPC y CSGS, presentamos el Cuadro 6.1. Este cuadro muestra una estimación (bastante optimista) del número mínimo de pruebas de independencias ejecutadas para construir la estructura para cada uno de los conjuntos de datos utilizados en el escenario del mundo real. Esta estimación se basó en el número de aristas de un grafo (n^2) y el número de grafos canónicos que deben ser construidos. El número de grafos canónicos depende del número de contextos canónicos (columna $\mathcal{X}$ en Cuadro 6.1), el cual es siempre menor o igual al número de ejemplos en cada conjunto de entrenamiento (D_E). En base a este cuadro, por ejemplo, en el conjunto llamado *Adult*, CSPC y CSGS construyeron 25,060 grafos canónicos. Dado que cada grafo canónico requiere, al menos, la ejecución de $O(n^2)$ pruebas de independencia; entonces la construcción de la estructura involucra la ejecución de al menos 391,562,500 pruebas de independencias.

Conjunto D	n	D_E	Conjunto $\mathcal{X}$	Número de pruebas
NLTCS	16	16,181	2,671	683,776
MSNBC	17	291,326	9,228	2,666,892
Abalone	31	3,134	648	622,728
Wine	48	4,874	3,898	8,980,992
KDDCup 2000	65	180,092	6,864	29,000,400
Plants	69	17,412	8,258	39,316,338
Covertype	84	30,000	22,679	160,023,024
Audio	100	15,000	14,969	149,690,000
Netflix	100	15,000	14,998	149,980,000
Accidents	111	12,758	12,756	157,166,676
Adult	125	36,631	25,060	391,562,500
Retail	135	22,041	10,510	191,544,750
Pumsb Star	163	12,262	12,037	319,811,053
DNA	180	1,600	1,545	50,058,000
MSWeb	294	29,441	10,294	889,772,184
WebKB	839	2,803	2,754	1,938,598,434
BBC	1,058	1,670	1,585	1,774,191,940
Ad	1,556	2,461	1,573	3,808,446,928

Cuadro 6.1: Número de contextos ($\mathcal{X}$) utilizados por los algoritmos CSPC y CSGS para construir la estructura de una red de Markov para cada conjunto de entrenamiento (D_E) del escenario del mundo real.

Una estrategia para reducir el número de pruebas de independencias es evitando la

ejecución de pruebas estadísticas redundantes. Una forma de realizar esto es mediante la utilización del *Triangle Theorem* [13]. Este teorema muestra como inferir el valor de verdad de una independencia utilizando los valores de verdad de independencias que ya han sido obtenidos. Así, sólo cuando el valor de verdad no puede ser inferido, la prueba de independencia es ejecutada.

Por otro lado, para mejorar el poder estadístico, podrían utilizarse métodos alternativos para identificar independencias. Por ejemplo, algunos métodos interesantes para explorar son: el *test bayesiano* [60], la *regresión logística* [78] y la *entropía condicional* [1]. Finalmente, otra forma de mejorar el poder estadístico es explorando la utilización de otros algoritmos de aprendizaje de estructura para la construcción de los grafos canónicos. Tres algoritmos que resultan bastante interesante son: IBMAP-HC [81], L1 [78] y GSSL [89]. Como hemos mencionado, el algoritmo IBMAP-HC justamente está diseñado para reducir los errores producidos al ejecutar pruebas estadísticas con un reducido número de datos. Por otro lado, el algoritmo L1 utiliza una forma novedosa para determinar la presencia de una arista a través del uso de regresión logística. Por último, el algoritmo GSSL utiliza un enfoque aleatorio para construir la estructura.

Apéndice A

Propiedades de Markov

El más grande objetivo de toda ciencia es cubrir el mayor número de hechos empíricos por deducción lógica desde el más pequeño número de hipótesis o axiomas.

Albert Einstein.

En la literatura, la relación o propiedad $I(\cdot \perp \cdot \mid \cdot)$ está definida sobre conjuntos disjuntos de variables en X en cualquier distribución de probabilidad $p(X)$, donde $I(\cdot \perp \cdot \mid \cdot)$ satisface las siguientes propiedades (o axiomas) [71, § 3].

- *Simetría* (*Symmetry*):

  $I(X_A \perp X_B \mid X_C) \implies I(X_B \perp X_A \mid X_C)$.

- *Descomposición* (*Decomposition*):

  $I(X_A \perp X_{B \cup W} \mid X_C) \implies I(X_A \perp X_B \mid X_C) \wedge I(X_A \perp X_W \mid X_C)$.

- *Contracción* (*Contraction*):

  $I(X_A \perp X_B \mid X_C) \wedge I(X_A \perp X_W \mid X_{C \cup B}) \implies I(X_A \perp X_{B \cup W} \mid X_C)$.

- *Intersección* (*Intersection*):

  $I(X_A \perp X_B \mid X_{C \cup W}) \wedge I(X_A \perp X_W \mid X_{C \cup B}) \implies I(X_A \perp X_{B \cup W} \mid X_C)$.

- *Unión débil* (*Weak union*):

  $I(X_A \perp X_{B \cup W} \mid X_C) \implies I(X_A \perp X_B \mid X_{C \cup W})$.

- *Unión fuerte* (*Strong union*):

  $I(X_A \perp X_B \mid X_C) \implies I(X_A \perp X_B \mid X_{C \cup W})$.

- *Transitividad* (*Transitivity*):

  $I(X_A \perp X_B \mid X_C) \implies I(X_A \perp X_b \mid X_C) \vee I(X_b \perp X_A \mid X_C)$.

Las dos últimas propiedades, unión fuerte y transitividad, requieren que la distribución $p(X)$ sea positiva.

Bibliografía

[1] Pieter Abbeel, Daphne Koller, and Andrew Y Ng. Learning factor graphs in polynomial time and sample complexity. *The Journal of Machine Learning Research*, 7:1743–1788, 2006. 54, 270

[2] A. Agresti. *Categorical Data Analysis*. Wiley, 2nd edition, 2002. 80, 184

[3] Constantin F Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D Koutsoukos. Local causal and markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *The Journal of Machine Learning Research*, 11:171–234, 2010. 22, 23, 35, 72, 226

[4] Janet M Baker, Li Deng, James Glass, Sanjeev Khudanpur, Chin-Hui Lee, Nelson Morgan, and Douglas O Shaughnessy. Developments and directions in speech recognition and understanding, part 1 [dsp education]. *Signal Processing Magazine, IEEE*, 26(3):75–80, 2009. 19

[5] Yoseph Barash and Nir Friedman. Context-specific bayesian clustering for gene expression data. *Journal of Computational Biology*, 9(2):169–191, 2002. 118

[6] Csaba Benedek and Tamás Szirányi. A mixed Markov model for change detection in aerial photos with large time differences. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008. 121

[7] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986. 93

[8] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006. 74

[9] David C Blair. Language and representation in information retrieval. 1990. 73

[10] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence*, pages 115–123. Morgan Kaufmann Publishers Inc., 1996. 24, 33, 54, 98, 114, 115, 118

[11] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001. 18, 44

[12] Eliot Brenner and David Sontag. Sparsityboost: A new scoring function for learning bayesian network structure. 2013. 81

[13] Facundo Bromberg, Dimitris Margaritis, and Vasant Honavar. Efficient Markov network structure discovery using independence tests. *Journal of Artificial Intelligence Research*, 35(2):449, 2009. 22, 23, 29, 38, 72, 86, 226, 270

[14] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of The ACM*, 16:575–577, 1973. 200

[15] Yutian Chen and Max Welling. Bayesian structure learning for markov random fields with a spike and slab prior. 2012. 95

[16] D. M. Chickering, D. Heckerman, and C. Meek. A Bayesian Approach to Learning Bayesian Networks with Local Structure. In *Uncertainty in Artificial Intelligence*, pages 80–89. Morgan Kaufmann Publishers Inc., 1997. 33, 114, 115, 116, 118

[17] Thomas Cover and Joy Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991. 231, 232

[18] Jesse Davis and Pedro Domingos. Bottom-up learning of Markov network structure. In *Proceedings of the 27th International Conference on Machine Learning*, pages 271–280, 2010. 22, 30, 31, 72, 75, 144, 200, 226, 234, 255

[19] A Philip Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–31, 1979. 53, 57

[20] MH DeGroot and MJ Schervish. Probability and statistics-international edition, 2001. 81

[21] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. Biological sequence analysis: probabilistic models of proteins and nucleic acids. cambridge univ, 1998. 19

[22] Alejandro Edera and Facundo Bromberg. Aprendizaje de independencias específicas del contexto en markov random fields. In *XVII Congreso Argentino de Ciencias de la Computación*, 2011. 35

[23] Alejandro Edera, Facundo Bromberg, and Federico Schlüter. Markov random fields factorization with context-specific independences. *arXiv preprint arXiv:1306.2295*, 2013. 36, 55

[24] Alejandro Edera, Federico Schlüter, and Facundo Bromberg. Learning Markov networks with context-specific independences. In *the 25th International Conference on Tools with Artificial Intelligence*, pages 553–560. IEEE, 2013. 24, 37

[25] Alejandro Edera, Federico Schlüter, and Facundo Bromberg. Learning Markov networks structures constrained by context-specific independences. *International Journal on Artificial Intelligence Tools*, 23(06):1460030, 2014. 31, 38

[26] Alejandro Edera, Yanela Strappa, and Facundo Bromberg. The Grow-Shrink strategy for learning markov network structures constrained by context-specific independences. In *Proceedings of the 14th Ibero-American Conference on Artificial Intelligence*, pages 283–294. Springer, 2014. 38

[27] Magnus Ekeberg, Cecilia Lövkvist, Yueheng Lan, Martin Weigt, and Erik Aurell. Improved contact prediction in proteins: using pseudolikelihoods to infer potts models. *Physical Review E*, 87(1):012707, 2013. 19

[28] P Svante Eriksen. *Context specific interaction models*. Citeseer, 1999. 24, 25, 119

[29] Poul Svante Eriksen. Decomposable log-linear models. Technical report, Department of Mathematical Sciences, Aalborg University, 2005. 119

[30] D. Fierens. Context-specific independence in directed relational probabilistic models and its influence on the efficiency of Gibbs sampling. In *European Conference on Artificial Intelligence*, pages 243–248, 2010. 98

[31] Ian Foster. *Designing and building parallel programs*. Addison Wesley Publishing Company, 1995. 191

[32] Arthur Fridman. Mixed markov models. *Proceedings of the National Academy of Sciences*, 100(14):8092–8096, 2003. 24, 65, 104, 120, 121

[33] Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. In *Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence*, pages 252–262. 1996. 24, 114, 115, 118

[34] Parichey Gandhi, Facundo Bromberg, and Dimitris Margaritis. Learning Markov Network Structure using Few Independence Tests. In *SIAM International Conference on Data Mining*, pages 680–691, 2008. 219, 220

[35] D. Geiger and D. Heckerman. Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence*, 82:45–74, 1996. 55, 104

[36] Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 2013. 30, 31, 96, 223, 229, 234

[37] Walter R Gilks, Sylvia Richardson, and David J Spiegelhalter. Introducing markov chain monte carlo. In *Markov chain Monte Carlo in practice*, pages 1–19. Springer, 1996. 96

[38] Vibhav Gogate, William Webb, and Pedro Domingos. Learning efficient Markov networks. In *Advances in Neural Information Processing Systems*, pages 748–756, 2010. 55, 94, 217

[39] John Hammersley and Peter Clifford. Markov fields on finite graphs and lattices. *Unpublished manuscript*, 1971. 47, 64

[40] Hisayuki Hara, Tomonari Sei, and Akimichi Takemura. Hierarchical subspace models for contingency tables. *Journal of Multivariate Analysis*, 103(1):19–34, 2012. 119, 144

[41] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009. 17, 74, 229, 230

[42] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *The Journal of Machine Learning Research*, 1:49–75, 2001. 21

[43] Søren Højsgaard. Yggdrasil: a statistical package for learning split models. In *Proceedings of the 16th conference on Uncertainty in Artificial Intelligence*, pages 274–281. Morgan Kaufmann Publishers Inc., 2000. 24, 25, 26, 55, 104, 119, 129, 134, 140, 143, 144, 266

[44] Søren Højsgaard. Statistical inference in context specific interaction models for contingency tables. *Scandinavian journal of statistics*, 31(1):143–158, 2004. 24, 25, 26, 55, 104, 119, 120, 122, 123, 124, 125, 126, 128, 130, 144

[45] Gideon Dror Isabelle Guyon and Vincent Lemaire. *Unsupervised and Transfer Learning: Challenges in Machine Learning*, volume 7. Microtome Publishing, 2013. 18

[46] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, 2009. 19, 21, 24, 31, 48, 49, 57, 64, 81, 96, 97, 98, 108, 115, 228, 245

[47] Paul Komarek and Andrew Moore. A dynamic adaptation of ad-trees for efficient machine learning on large data sets. In *ICML*, pages 495–502. Citeseer, 2000. 197

[48] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In *Advances in neural information processing systems*, pages 785–792, 2007. 93

[49] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. 231

[50] Steffen Lauritzen. *Graphical models*. Oxford University Press, 1996. 63

[51] S. I. Lee, V. Ganapathi, and Daphne Koller. Efficient structure learning of Markov networks using L1-regularization. In *Neural Information Processing Systems*. Citeseer, 2006. 31, 69, 94, 199, 217, 229, 234

[52] M. Lichman. UCI machine learning repository, 2013. 224

[53] Dong Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. 227

[54] Daniel Lowd. Closed-form learning of markov networks from dependency networks. 2012. 21, 30, 93, 223

[55] Daniel Lowd and Jesse Davis. Learning Markov network structure with decision trees. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 334–343. IEEE, 2010. 229

[56] Daniel Lowd and Jesse Davis. Improving Markov network structure learning using decision trees. *Journal of Machine Learning Research*, 15:501–532, 2014. 22, 30, 31, 75, 94, 144, 199, 226, 227, 229, 234, 255

[57] Daniel Lowd and Amirmohammad Rooshenas. Learning Markov networks with arithmetic circuits. *The Journal of Machine Learning Research*, 31:406–414, 2013. 30, 223, 226, 229, 255

[58] Daniel Lowd and Amirmohammad Rooshenas. The libra toolkit for probabilistic models. *arXiv preprint arXiv:1504.00110*, 2015. 222, 227

[59] D. Margaritis and F. Bromberg. Efficient Markov Network Discovery Using Particle Filter. *Comp. Intel.*, 25(4):367–394, 2009. 219, 220

[60] Dimitris Margaritis. Distribution-free learning of bayesian network structure in continuous domains. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 825. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005. 270

[61] Dimitris Margaritis and Sebastian Thrun. Bayesian network induction via local neighborhoods. Technical report, DTIC Document, 2000. 22, 23, 29, 86, 87, 226, 267

[62] Andrew McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 403–410. Morgan Kaufmann Publishers Inc., 2002. 22

[63] Tom M Mitchell. Generalization as search. *Artificial intelligence*, 18(2):203–226, 1982. 71, 74

[64] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997. 17, 18, 47, 74

[65] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012. 17

[66] Andrew Moore and Mary Soon Lee. Cached Suficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998. 196, 197

[67] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012. 17, 21

[68] Henrik Nyman, Johan Pensar, Timo Koski, and Jukka Corander. Stratified graphical models-context-specific independence in graphical models. *arXiv preprint arXiv:1309.6415*, 2013. 119, 144

[69] Henrik Nyman, Johan Pensar, Timo Koski, and Jukka Corander. Context-specific independence in graphical log-linear models. *arXiv preprint arXiv:1409.2713*, 2014. 25, 119, 144

[70] Henrik Nyman, Johan Pensar, Timo Koski, Jukka Corander, et al. Stratified graphical models-context-specific independence in graphical models. *Bayesian Analysis*, 9(4):883–908, 2014. 104, 119, 144

[71] Azaria Paz, Judea Pearl, and Shmuel Ur. A new characterization of graphs based on interception relations. *Journal of Graph Theory*, 22(2):125–136, 1996. 23, 61, 271

[72] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1re edition, 1988. 20, 62, 64, 77, 78, 119

[73] Judea Pearl and Azaria Paz. GRAPHOIDS : A graph based logic for reasonning about relevance relations. Technical Report 850038 (R-53-L), Cognitive Systems Laboratory, University of California, Los Angeles, 1985. 61

[74] Karl Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900. 80

[75] S. Della Pietra, V. J. Della Pietra, and J. D. Lafferty. Inducing Features of Random Fields. *IEEE Trans. PAMI.*, 19(4):380–393, 1997. 22, 23, 72

[76] D. Poole and N. L. Zhang. Exploiting contextual independence in probabilistic inference. *J. Artif. Intell. Res. (JAIR)*, 18:263–313, 2003. 64, 98, 104

[77] Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011. 47

[78] Pradeep Ravikumar, Martin Wainwright, and John Lafferty. High-dimensional Ising model selection using L1-regularized logistic regression. *Annals of Statistics*, 38:1287–1319, 2010. 72, 94, 226, 249, 270

[79] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1):273–302, 1996. 95

[80] Karen Sachs, Omar Perez, Dana Pe'er, Douglas A Lauffenburger, and Garry P Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529, 2005. 20

[81] Federico Schlüter, Facundo Bromberg, and Alejandro Edera. The IBMAP approach for Markov network structure learning. *Annals of Mathematics and Artificial Intelligence*, pages 1–27, 2014. 22, 23, 72, 217, 219, 220, 226, 270

[82] Mark W Schmidt, Kevin P Murphy, Glenn Fung, and Rómer Rosales. Structure learning in random fields for heart motion abnormality detection. In *CVPR*, volume 1, page 2. Citeseer, 2008. 19

[83] Gabriele Schweikert, Alexander Zien, Georg Zeller, Jonas Behr, Christoph Dieterich, Cheng Soon Ong, Petra Philips, Fabio De Bona, Lisa Hartmann, Anja Bohlen, et al. mgene: accurate svm-based gene finding with an application to nematode genomes. *Genome research*, 2009. 19

[84] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, 1949. 18

[85] V Anne Smith, Jing Yu, Tom V Smulders, Alexander J Hartemink, and Erich D Jarvis. Computational inference of neural information flow networks. *PLoS computational biology*, 2(11):e161, 2006. 20

[86] Ray Solomonoff. The universal distribution and machine learning. *The Computer Journal*, 46(6):598–601, 2003. 44

[87] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991. 22, 23, 29, 36, 72, 81, 226, 267

[88] Jan Van Haaren, J Davis, GAM Lappenschaar, and AJ Hommersom. Exploring disease interactions using markov networks. 2013. 20, 94, 200, 223, 227, 229

[89] Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the 26th National Conference on Artificial Intelligence*. AAAI Press, 2012. 22, 30, 31, 72, 75, 144, 226, 227, 229, 234, 252, 255, 270

[90] Stephan Vogel, Hermann Ney, and Christoph Tillmann. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics, 1996. 19

[91] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008. 20

[92] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Ithaca: Cornell University Press, 1971. 73

[93] Stephen J Wright and Jorge Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999. 92