Inferencia causal mediante correlación sintáctica

Gabriel M. Goren

Tesis de Licenciatura en Ciencias Físicas

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Mayo de 2021

TEMA:	Inferencia causal mediante correlación sintáctica .
ALUMNO:	Gabriel M. Goren
L.U. N°:	199/15
LUGAR DE TRABAJO:	Departamento de Física, FCEN, UBA
DIRECTOR DEL TRABAJO:	Dr. Ariel Bendersky
CODIRECTOR:	Dr. Santiago Figueira
FECHA DE INICIACION:	Septiembre de 2020
FECHA DE FINALIZACION:	Mayo de 2021
FECHA DE EXAMEN:	
INFORME FINAL APROBADO POR:	

Autor	Jurado
Director	Jurado
Profesor(a) de Tesis de Licenciatura	Jurado

Resumen

Distinguir correlación de causalidad es uno de los desafíos de inferencia más importantes en la actividad científica. Alrededor de este problema se desarrolla la teoría de *modelos* o *redes causales*, la cual provee un lenguaje basado en grafos que permite razonar formalmente sobre relaciones causales y obtener conclusiones a partir de datos e hipótesis adecuadas. Un punto crucial resulta ser qué constituye una correlación o dependencia entre variables.

En este trabajo se presenta un abordaje a la inferencia de modelos causales basado en nociones *sintácticas* de correlación, en contraposición con la noción de correlación estadística usual. En este contexto, las observaciones son codificadas como cadenas de símbolos, y las dependencias entre ellas son evaluadas en términos de la *forma* de las mismas, y del grado en que pueden ser descriptas o generadas mediante reglas sucintas similares. El concepto de *información algorítmica* formaliza estas ideas, y de él se desprende una noción de correlación algorítmica universal. Sin embargo, la información algorítmica no puede ser calculada efectivamente, por lo que los tests de independencia condicional sintáctica utilizables en la práctica deben estar necesariamente basados en medidas de complejidad sintáctica más débiles.

Concretamente, se implementó un algoritmo para el descubrimiento del esqueleto de la estructura causal subyacente a un conjunto de datos simbólicos, y se desarrollaron tests de independencia condicional basados en dos medidas de complejidad sintáctica computables: la longitud de compresión mediante el compresor comercial *gzip* y la I-complejidad de Becher y Heiber. Se construyeron ciertos modelos funcionales como generadores de casos de prueba y se realizaron simulaciones en las que se buscó reconstruir la estructura causal de los modelos utilizando el algoritmo. De esta forma se identificó una clase de modelos cuya estructura causal no dirigida pudo ser inferida empleando tests estadísticos, mas no con tests sintácticos (modelos tipo XOR); una clase de modelos cuya estructura pudo ser reconstruida mediante ambos tipos de test (modelos de concatenación) y otra cuya estructura solo pudo ser reconstruida mediante tests sintácticos (modelos de concatenación con shifts).

Agradecimientos

Este trabajo representa la finalización de una carrera universitaria y una etapa de vida. Es el resultado de un contexto colectivo que lo hizo posible, tanto material como espiritualmente. Mi intención con estos agradecimientos es dejar constancia, por más breve que sea, de que haber llegado hasta aquí no es un logro individual sino el de una comunidad.

En primer lugar quiero agradecer a mis directores Ariel y Santi por haberse embarcado conmigo en el desafío de esta tesis; por guiarme con constancia y por apoyarme en los momentos difíciles. Más aún quiero agradecer el esfuerzo y el compromiso de todos los trabajadores de la facultad que hizo posible y deseable mi recorrido por esta institución.

Agradezco a mi mamá y mi papá, Hebe y Gustavo, por su cariño y su apoyo incondicional, y por su preocupación por orientarme en el tramo final.

A mis amigas y amigos de Exactas les debo la verdadera riqueza y profundidad de estos años. Gracias a ellos nuestro espacio común se tiñó para siempre de ese color vibrante que tiene la aventura compartida.

A mis amigues del secundario y a mi familia, gracias por estar en mi vida, o mejor dicho por constituirla. Con ustedes no es difícil reconocer que somos olas de un mismo océano.

Por úlitmo, por su rol crucial en los últimos meses de trabajo y de escritura, a mi prima Ana y mi novia Sofi, gracias por bancarme en el día a día, acompañarme y ayudarme a no perder de vista las cosas que importan.

Índice general

Capítulo 1

Introducción

> *The view that machines cannot give rise to surprises is due, I believe, to a fallacy to which philosophers and mathematicians are particularly subject. This is the assumption that as soon as a fact is presented to a mind all consequences of that fact spring into the mind simultaneously with it. It is a very useful assumption under many circumstances, but one too easily forgets that it is false.*
>
> Alan M. Turing [1]

El presente trabajo es una exploración de la posibilidad de aplicar ideas provenientes de la Teoría Algorítmica de la Información al contexto de la inferencia causal, entendido en particular desde el formalismo de redes causales. En esta sección, se introducen brevemente ambos tópicos y se presenta el objetivo del trabajo, enfatizando la motivación detrás de intentar combinar estas dos áreas.

1.1. Modelos causales: ¿Correlación o causalidad?

Una de las tareas fundamentales de inferencia consiste en la identificación de relaciones *causales*. Esto es, dado un conjunto de variables empíricamente observables y cierto conocimiento o teoría preexistente sobre los fenómenos que dan lugar a dichos observables, se busca distinguir cuándo las correlaciones presentes en los datos sugieren la existencia de *mecanismos* concretos de la naturaleza.

¿Es posible concluir que existe una relación causal entre dos variables y no meramente una asociación de modo *formal*, como resultado de algún tipo de cálculo? La respuesta obvia es un rotundo "no". Dejando de lado las dificultades asociadas a entender la causalidad desde un punto de vista ontológico o metafísico, la experiencia de la práctica

empírica muestra que distinguir causalidad de correlación suele requerir un entendimiento detallado del dominio específico de conocimiento involucrado. Desde este punto de vista, mientras no haya un cálculo capaz de imitar el comportamiento de la mente de les científiques, no podría haber un cálculo de la causalidad científica.

Sin embargo, insistir en una posible respuesta afirmativa a la pregunta nos conduce a considerar qué sería necesario para poder tener un cálculo semejante y en qué sentido sería posible inferir relaciones causales de manera formal. Para empezar, será necesario tener una noción formal de lo que quiere decir "relación causal". Es decir, se requiere en primer lugar de un *lenguaje* en el cual expresar afirmaciones de tipo causal. Es a este lenguaje o formalismo matemático al que nos referimos cuando hablamos de *modelos causales*.

Más aún, si disponemos de un formalismo para expresar hipótesis y premisas causales, entonces podemos en particular incorporar el conocimiento experto sobre dominios específicos en forma de premisas formuladas dentro del formalismo. Así, arribamos a una primera respuesta ante el problema recién presentado: un cálculo de la causalidad no puede buscar reemplazar el conocimiento experto, sino esclarecer las premisas que de hecho ese conocimiento experto implica, así como las consecuencias que dichas premisas conllevan. No se trata de obtener conclusiones causales a partir de correlaciones, sino a partir de dichas correlaciones sumadas a hipótesis causales (y por lo tanto extra-estadísticas) expresadas de un modo preciso.

La teoría de modelos causales permite expresar estas nociones empleando un lenguaje de grafos. La parte más importante de un modelo causal es un grafo dirigido (nodos conectados por flechas), su *estructura causal*, cuyos nodos o vértices corresponden a las variables bajo estudio, y cuyas flechas representan relaciones causales *inmediatas*. Que una relación causal sea inmediata se expresa en esta teoría por medio de una propiedad Markoviana (la *condición de Markov causal*): dadas las causas inmediatas de un cierto efecto, el efecto es independiente del resto de sus causas. Usamos grafos dirigidos porque las relaciones causales son asimétricas: de un lado tenemos las causas y del otro, los efectos. Más aún, dado que no permitimos que una variable sea causa de sí misma, en estos grafos no es posible partir de un nodo y, siguiendo las direcciones de las flechas, volver al mismo. Es decir que se trata de grafos dirigidos *acíclicos*, o *DAG*s por sus siglas en inglés.

Una observación crucial es que, en contraposición con meras correlaciones, las relaciones causales entre observables se caracterizan por ser robustas frente a variaciones en las condiciones de observación, en virtud de corresponder a mecanismos estables [2]. Así, una afirmación causal puede entenderse como un enunciado que no solamente refiere a

las correlaciones observadas, sino también a cuáles serían las correlaciones en caso de realizarse ciertas intervenciones, o en caso de variar alguna condición. Es decir que el razonamiento causal es contrafáctico. Uno de los principios fundamentales de la inferencia causal es la idea de que los mecanismos causales básicos (correspondientes a causas inmediatas) son independientes, y es posible razonar sobre la modificación de tan solo uno de ellos sin que se modifiquen los demás.

Consideremos un ejemplo proveniente de la investigación clínica [3]. Durante un ensayo clínico, a cada sujeto se le asigna un cierto tratamiento, y luego de cierto tiempo se miden o evalúan los resultados del tratamiento. Sin embargo, el tratamiento *asignado* a un sujeto puede diferir del tratamiento *efectuado*, por ejemplo porque el sujeto no cumple el tratamiento, lo cual puede conceptualizarse como un movimiento desde un grupo de tratamiento hacia otro. Una gran dificultad en el análisis de los ensayos clínicos consiste en el hecho de que existe una dependencia muy fuerte entre el grado en que los sujetos cumplen con el tratamiento asignado y el potencial beneficio del tratamiento en dicho sujeto (e.g. sujetos que reaccionan de forma adversa tienden a suspender sus tratamientos). El grafo presentado en la Figura 1.1 permite formalizar de manera abstracta esta situación. En él, el nodo A representa el tratamiento asignado, mientras que X representa el tratamiento efectivamente realizado. El nodo Y representa el resultado del tratamiento y B es una variable *latente* (i.e. no observable) que implementa la dependencia entre cumplimiento del tratamiento y potencial beneficio.

En este ejemplo, A constituye una *variable instrumental*, dado que su observación nos permite distinguir entre la dirección causal $X \to Y$ y la dirección causal $Y \to X$, incluso en presencia de "factores de confusión" inobservables (B) que afectan a ambas.[1] Por ejemplo, en física experimental, este patrón de dependencias también aparece cada vez que tenemos sistemas físicos que sufren fluctuaciones aleatorias (B) y en los cuales se desea estimar el efecto de una variable X sobre otra variable Y, pero solo se puede controlar la variable de entrada X (a través del mecanismo de control dado por A).

Sin embargo, A no tiene por qué representar un control explícito sobre el objeto de estudio, y este punto es clave para entender el potencial de la modelización causal. Podemos tomar, concretamente, el caso clásico del efecto de fumar (X) sobre el cáncer de pulmón (Y). En ausencia de A, y habiendo variables latentes, i.e. no observables (U), no es posible decidir únicamente a partir de la asociación estadística entre estas variables si una es causa de la otra, o viceversa, ni tampoco si hay o no un vínculo causal entre

[1] Pearl caracteriza el concepto de variable instrumental utilizando este lenguaje de grafos. Así, define como variable instrumental con respecto a una variable observable Y a toda variable observable A tal que todos los caminos dirigidos entre A e Y son d-separados por alguna variable observable. Esta terminología es presentada entre la Sección 1.4 y el Capítulo 2.

ellas en un primer lugar. La variable A puede ser entonces un instrumento aleatorizado que fomenta la participación de sujetos en distintos programas de estudio, en cuyo caso tenemos un ensayo clínico, o bien, quizá más éticamente en este caso, puede corresponder a la presencia o ausencia de legislación anti-tabaco a lo largo de distintas regiones. Esto último enfatiza el hecho de que muchas veces es posible incorporar variables en el análisis, quizá tan solo indirectamente relacionadas con el objeto de estudio, que permitan extraer conclusiones causales sin la necesidad de realizar experimentos controlados, los cuales pueden no ser éticos y/o factibles. Estas son las variables instrumentales. Es en este sentido también que la teoría de modelos causales está fuertemente ligada a desarrollos dentro de la tradición econométrica[2].

Dado que el grafo de la Figura 1.1 abstrae la estructura mínima presente en todos los casos mencionados, los cuales constituyen la gran mayoría de las situaciones experimentales, se le da el nombre de *proceso instrumental* o *modelo instrumental* a la estructura causal representada por este grafo, el cual denominaremos acordemente *grafo instrumental*. Retomaremos este ejemplo en los Capítulos 2 y 4.

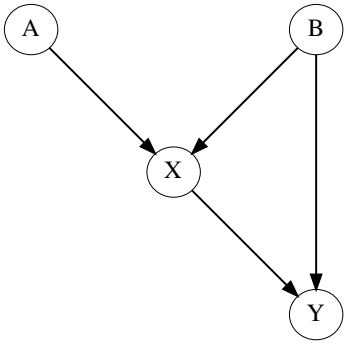


Figura 1.1: El *grafo instrumental* constituye un ejemplo de estructura causal, al interpretar sus vértices como variables y sus aristas como relaciones causales inmediatas. Esta estructura tiene la particularidad de abstraer el concepto de *variable instrumental*. Cuando se interpreta a B como un factor latente (inobservable) que afecta a dos variables de interés X e Y, la observación de la variable A es indispensable si se desea establecer si existe una relación causal entre X e Y (y si la hay, en qué dirección).

Dentro del abanico de tareas que pueden ser realizadas con la ayuda del lenguaje que nos proporcionan los modelos causales, podemos distinguir lo que Pearl llama *descubrimiento causal*. Esta tarea consiste en identificar relaciones causales a partir de datos observacionales. Por supuesto, no siempre es mucho lo que puede concluirse si no se tiene la posibilidad de realizar intervenciones o experimentos, que permitan aislar el efecto

[2]La relación entre la Econometría y la teoría de modelos causales excede el presente trabajo. Una exposición de la teoría de modelos causales para una audiencia de econometría puede encontrarse en [4].

entre variables específicas. Sin embargo, como muestra el concepto de variable instrumental, existen situaciones en las que pueden concluirse la presencia de ciertas relaciones causales entre las variables, incluso sin realizar intervenciones, si se introducen hipótesis causales adicionales. Como veremos en el Capítulo 2, asumir que los datos (y el proceso que los generó) cumplen las hipótesis de *suficiencia causal*, *minimalidad* (una variante de la navaja de Ockham) y *estabilidad* será suficiente para la justificar la existencia de un algoritmo que infiere algunas (no todas) las relaciones causales existentes entre las variables. Esto se logra analizando qué restricciones sobre las relaciones causales son implicadas por las *relaciones de independencia condicional* presentes en los datos. Por ejemplo, si tenemos dos variables dependientes A, B tales que al condicionar sobre una tercera variable C, se vuelven independientes, entonces podemos concluir[3]que, de haber una relación causal entre A y B, esta nunca puede ser *inmediata*: a lo sumo, habrá una influencia causal mediada por la variable C.

1.2. Correlación sintáctica

La teoría de modelos causales está formulada en términos estadísticos: los observables son representados por variables aleatorias y los datos consisten en estimaciones empíricas de su distribución de probabilidad conjunta. Así, dicha independencia resulta ser independencia probabilística de variables aleatorias. Sin embargo, muchas veces se ha señalado que la noción fundamental de *independencia condicional* puede expresarse axiomáticamente sin hacer referencia alguna a probabilidades o variables aleatorias [5, 6]. En términos abstractos, una expresión del tipo "U es irrelevante para V si se conoce W" (donde U, V, W son conjuntos de variables) constituye una relación ternaria, la cual debe satisfacer ciertas propiedades (ver Sección 2.1). Una notación que utilizaremos extensamente para expresar esta relación de independencia condicional es $(U \perp\!\!\!\perp V|W)$.

Siguiendo esta línea, más recientemente han sido publicados algunos trabajos que buscan desarrollar herramientas de inferencia causal basadas en una idea de correlación *algorítmica* en vez de probabilística [7, 8]. Esto permite manipular observables en términos de cualquier descripción finita de los mismos, tales como en forma de texto, tabla, grafo, o imagen digital, llevando estas descripciones a la forma de cadenas de símbolos mediante alguna codificación adecuada.

Informalmente, podemos decir que dos observables u y v son *algorítmicamente independientes* dado w si el proceso computacional más sencillo de describir que genera

[3]Estamos empleando un principio, atribuido a Reichenbach, que consiste en asumir que no puede haber una relación causal allí donde no hay correlación.

conjuntamente a u y v a partir de w consiste en primero generar u a partir de w y luego generar v a partir de w, siendo estos dos pasos totalmente independientes. La formalización de esta frase, y de la idea general de correlación algorítmica, emplea conceptos propios la Teoría Algorítmica de la Información. Esta teoría se construye alrededor de la *complejidad de Kolmogorov* o *información algorítmica*: dada una cadena de símbolos x, su información algorítmica $K(x)$ es la longitud del programa más corto que genera esa cadena.

La información algorítmica de x puede entenderse como el tamaño de su compresión algorítmica óptima. En este sentido, u y v son independientes dado w si el mejor algoritmo de compresión para comprimir conjuntamente a u y v empleando w como ayuda es el que comprime óptimamente a u y a v por separado (empleando w) y después pega o concatena sus compresiones.

1.3. Motivación y objetivos

Basar una teoría de inferencia causal en la información algorítmica resulta una idea atractiva en virtud de la rica teoría detrás de esta última y el alcance universal de la misma: mediante la observación de variaciones de información algorítmica mutua, es posible detectar cualquier mecanismo causal entre dos observables que sea simulable o implementable computacionalmente (lo cual equivale a *todos* los mecanismos causales posibles bajo la versión física de la tesis de Church-Turing). Esto incluye incluso relaciones que no pueden ser detectadas estadísticamente. Por ejemplo, si se tira una moneda n veces, y anotamos 1 si el resultado es cara, y 0 si es ceca, obtenemos una cadena binaria $x = x_1, x_2, ..., x_n$. Es claro que estos datos están totalmente correlacionados con la cadena $\bar{x} = \bar{x}_1, \bar{x}_2, ..., \bar{x}_n$, la cual corresponde a otro observador que anotó 1 cuando el resultado era ceca. Ahora bien, consideremos la cadena $y = x_2, x_3, ..., x_n, x_1$. Algorítmicamente, y se obtiene de aplicar un *shift* o permutación cíclica a los bits de x, lo cual es una operación extremadamente sencilla. Sin embargo, si el desplazamiento de los bits ocurre de forma inadvertida, al observarse las cadenas $\bar{x}, y$ como meros pares de resultados $(\bar{x}_i, y_i) = (\bar{x}_i, x_{i+1})$ (si $i \neq n$), se concluirá que no existe correlación alguna. En cambio, la información algorítmica permite detectar esta correlación dado que, una vez que se comprime o describe sucintamente la cadena x, tanto $\bar{x}$ como y pueden describirse como el resultado de aplicar a x una función sencilla, cuya especificación no depende de n (la que invierte cada bit y la que aplica el shift, respectivamente).

Más aún, la gran cantidad de resultados análogos entre la teoría de la información algorítmica y la teoría de la información clásica, así como resultados que que relacionan

directamente a la información algorítmica con la entropía o información de Shannon [9], facilita la obtención de resultados similares los de la teoría clásica de modelos causales.

Sin embargo, el uso de la información algorítmica trae aparejadas ciertas dificultades conceptuales relacionadas con el hecho de que la información algorítmica es incomputable, i.e. se puede demostrar matemáticamente que no existe un programa que la calcule. Más aún, no es posible controlar el error con el que es aproximada: no existe un programa que pueda calcular una aproximación a $K(x)$ para todo x con una cota de error dada.

Así, no es inmediatamente evidente qué rol cumple la información algorítmica en tareas reales de inferencia. Uno es el rol de "ideal platónico", inalcanzable pero motivador de definiciones y métodos de inferencia concretos y eficientemente computables relacionados con las ideas de compresión y códigos universales. Esta perspectiva enmarca la mayor parte de los trabajos que desarrollan métodos de análisis de datos basados en compresores (por ejemplo [10], [11]), así como el paradigma estadístico conocido como "longitud de descripción mínima" o MDL [12]. En otros casos, se busca desarrollar aproximaciones directas a la función K por más que el error no sea controlable [13]. También existen algunos resultados en la dirección de definir variantes computables de la información algorítmica que sean demostrablemente buenas aproximaciones a K bajo condiciones específicas, tales como el de Bloem et al. sobre información algorítmica *model-bounded* [14]. Finalmente, otra forma de estudiar la relación entre la información algorítmica y medidas de complejidad computables consiste en estudiar en qué medida estas cumplen, al menos aproximadamente, algunas propiedades teóricas de la información algorítmica [15].

* * *

El objetivo general de esta tesis de licenciatura es desarrollar herramientas para realizar inferencia causal sobre descripciones finitas de observables, codificadas como cadenas de símbolos o *strings*, apoyándonos en la noción de independencia basada en la información algorítmica y en los resultados que ya han sido obtenidos en esta dirección. Más concretamente, desarrollar e implementar un método de *descubrimiento causal*, que a partir de un conjunto de strings dado, devuelva un grafo que codifica información sobre la estructura de relaciones causales entre dichos strings.

Para esto se adaptó e implementó un algoritmo existente en la literatura de modelos causales, evaluando las dependencias entre observables mediante un análisis basado en medidas de correlación sintáctica computables. Para poner a prueba este método se construyeron ciertas clases de modelos causales funcionales, los cuales denominamos *modelos*

funcionales sintácticos, y se realizaron simulaciones en las que se ejecutó el algoritmo sobre datos generados por estos modelos.

Como se mencionó previamente, si bien la incomputabilidad de la información algorítmica no impide que esta sea útil para el desarrollo de métodos de inferencia, queda claro también que esta implica sutilezas conceptuales que deben ser abordadas, en particular la de en qué sentido un método aproxima a la información algorítmica, más allá de estar inspirado en ella, y cómo juzgar la efectividad del método, más allá de si en efecto la aproxima.

1.4. Notación y terminología

En esta sección fijamos notación que será empleada a lo largo del trabajo.

Conjuntos. Dado un conjunto S, denotamos su cardinalidad por $|S|$. El *conjunto de partes* de S se define como $\mathcal{P}(S) := \{T : T \subseteq S\}$. Como es usual, denotamos por $\mathbb{N}$, $\mathbb{Z}$ y $\mathbb{R}$ a los conjuntos de números naturales (incluyendo el cero), enteros y reales respectivamente, con sus estructuras de orden y algebraicas usuales.

Dados conjuntos S y T, una *relación binaria* R puede definirse como un subconjunto del producto cartesiano $R \times S$. Usamos como notaciones equivalentes $(s,t) \in R$, sRt y $R(s,t)$. Las relaciones *ternarias* y más en general *n-arias* para cualquier *aridad* $n \in \mathbb{N}$ se definen de forma análoga.

Probabilidades. Hablaremos de distribuciones de probabilidad conjunta p sobre variables $x_1, \ldots x_n$, entendiendo por regla general que refieren a variables aleatorias discretas. De esta manera se entiende por defecto que

$$p(x_1, \ldots x_n) = \Pr(X_1 = x_1 \wedge \ldots \wedge X_n = x_n) \tag{1.1}$$

A su vez, usamos la notación $p(x_i)$ para referirnos a la distribución marginal de X_i, y la notación $p(x_i|x_j)$ para referirnos a la distribución de probabilidad condicional de X_i dado el evento $X_j = x_j$. En general las letras latinas mayúsculas denotan variables aleatorias, mientras que las letras minúsculas correspondientes denotan realizaciones de las mismas. Utilizamos la abreviatura "i.i.d." para referirnos a conjuntos de variables aleatorias idéntica e independientemente distribuidas.

Grafos. Un *grafo* G consiste en un conjunto de *vértices* o *nodos* V y un conjunto de *aristas* E, cada una de las cuales conecta dos vértices distintos (no permitimos bucles

sobre un mismo vértice). Los grafos pueden ser *dirigidos*, si consideramos a la arista (a, b) como distinta de la arista (b, a), o *no dirigidos* en caso contrario. En el caso de grafos dirigidos, representamos a las aristas como flechas: $a \to b$ o $a \leftarrow b$ según el caso. Dos vértices conectados por una arista se dicen *adyacentes*. Más aún podemos considerar grafos *semidirigidos* en los que algunas aristas son dirigidas y otras no. Dado un grafo dirigido, si ignoramos las direcciones de las flechas podemos obtener naturalmente un grafo no dirigido subyacente, el cual denominamos el *esqueleto no dirigido* del grafo.

Un *camino* (de longitud n) en un grafo (con aristas posiblemente dirigidas) es una cadena de pares de vértices $(v_1, w_1), (v_2, w_2), \ldots (v_n, w_n)$ tal que cada par de vértices está conectado por una arista y además $v_{j+1} = w_j$ para j entre 0 y $n-1$. Nos referimos a pares de vértices en vez de aristas dado que la dirección de recorrido del camino no tiene por qué coincidir con la dirección de las aristas dirigidas que atraviesa. En el caso particular en que hay aristas dirigidas en el grafo, y la dirección del camino coincide con la dirección de todas las aristas dirigidas que atraviesa, decimos que se trata de un *camino dirigido*.

Un camino es un *ciclo* si el vértice de partida coincide con el vértice de llegada. Un grafo se dice *acíclico* si no contiene ningún ciclo dirigido. La clase de grafos que más nos interesará para la discusión de causalidad es la de los grafos dirigidos y acíclicos, los cuales denominamos *DAGs* por sus siglas en inglés.

Utilizamos terminología de parentesco para referirnos a las relaciones entre vértices conectados por aristas dirigidas. Así, los *padres* de un vértice v forman el conjunto PA_v de vértices w tales que existe una flecha $w \to v$, y los *hijos* de v son los w tales que existe una flecha $v \to w$. Un vértice se dice *raíz* si no tiene padres y *sumidero* si no tiene hijos. La clausura transitiva de la relación "ser padre de" define la relación "ser ancestro de", mientras que la clausura transitiva *y reflexiva* de "ser hijo de" define la relación "ser descendiente de". Notemos que entonces estamos definiendo a todo vértice como descendiente de sí mismo y, más importantemente, consideramos que el conjunto de *no descendientes* de un vértice v, el cual notaremos ND_v, excluye a v.

El *grado* de un vértice en un grafo es el número de *vecinos* o vértices adyacentes. En el caso de grafos con aristas dirigidas, podemos distinguir el grado *entrante* de un vértice, que equivale al número de padres, y el grado *saliente*, el cual se define como el número de hijos.

Un grafo no dirigido se dice *completo* cuando posee una arista entre cada par de vértices. Por el contrario, un grafo *discreto* es uno en el que no hay aristas.

Strings. Un *alfabeto* Σ es un conjunto finito y no vacío, cuyos elementos son denominados *símbolos*. Las *palabras* o *strings* de longitud n sobre Σ son los elementos del

conjunto $\Sigma^n = \underbrace{\Sigma \times \ldots \times \Sigma}_{n \text{ veces}}$. El conjunto Σ^0 se define como $\{\epsilon\}$, un conjunto singlete cuyo único elemento es interpretado como la palabra vacía. El conjunto de strings sobre Σ se define como $\Sigma^* := \bigcup_{n\in\mathbb{N}} \Sigma^n$, es decir las cadenas finitas de símbolos de cualquier longitud. Notamos $\Sigma^+ := \bigcup_{n\geq 1} \Sigma^n$ para referir al conjunto de strings no vacíos. Para alivianar la notación, el string $(x_1, x_2, \ldots x_n)$ es denotado por $x_1x_2 \ldots x_n$, sin paréntesis ni comas.

Dado un string x, denotamos por $|x|$ la longitud del mismo. La *concatenación* de un string $x = x_1 \ldots x_m$ con un string $y = y_1 \ldots y_n$ se define como el string $xy := x_1 \ldots x_m y_1 \ldots y_n$. La *potencia n-ésima* de x se define como $x^n = \underbrace{x \ldots x}_{n \text{ veces}}$.

En algunas definiciones utilizaremos las funciones *piso* y *techo*. Dado un número real $r \in \mathbb{R}$, el *piso* $\lfloor r \rfloor$ se define como el máximo de los números naturales menores o iguales a r. De manera dual, el *techo* $\lceil r \rceil$ se define como el mínimo de los números naturales mayores o iguales a r.

Dados dos strings x, y, decimos que x es *prefijo* de y si existe un string z tal que $y = xz$. En algunos contextos será deseable poder reemplazar un string x por otro a partir del cual pueda recuperarse x y que sea *autodelimitante*, en el sentido de que es posible distinguir dónde termina, cuando se lo considera como prefijo de un string más largo. Definimos así la *codificación autodelimitante* de x como $\bar{x} = 1^{|x|}0x$. Los primeros $|x|$ unos indican la longitud de x. Al encontrarnos un 0, sabemos que lo que sigue es el string x en sí, y como conocemos su longitud, sabemos exactamente cuándo dejar de leer.

Usamos la notación $\langle -, - \rangle$ para referirnos a distintas *funciones de formación de pares*, $\Sigma^* \times \Sigma^* \to \Sigma^*$, entendiéndose de cuál se trata a partir del contexto. Por ejemplo, la concatenación es una función de formación de pares válida. Sin embargo, no es una función biyectiva. Empleando codificaciones autodelimitantes como arriba, podemos construir a partir de dos strings x, y, un único string que contiene la información de ambos y en el cual sabemos cuándo termina uno y cuándo comienza el otro. Definimos así la *concatenación autodelimitante* de x, y como $\langle x, y \rangle = \bar{x}y = 1^{|x|}0xy$. A diferencia de la concatenación convencional, la concatenación autodelimitante sí es una función de formación de pares biyectiva.

Dada una función de formación de pares, definimos la formación de tuplas de n strings de forma inductiva según

$$\begin{aligned}\langle x_1, x_2, \ldots x_n \rangle &:= \langle x_1, \langle x_2, \ldots, x_n \rangle \rangle \\ &= \langle x_1, \langle x_2, \langle \ldots, \langle x_{n-1}, x_n \rangle \rangle \ldots \rangle \rangle\end{aligned}$$

Una función de formación de pares puede no ser asociativa (i.e. no es necesariamente cierto que $\langle a, \langle b, c\rangle\rangle = \langle\langle a, b\rangle, c\rangle$), en cuyo caso la definición dada para tuplas de $n > 2$ elementos es en cierto punto arbitraria. Por ejemplo, la concatenación autodelimitante no es asociativa, mientras que la concatenación convencional sí lo es.

Varios. Usamos el símbolo $\wedge$ para denotar la conjunción lógica ("y") y a veces abreviamos "si y solo si" por la expresión "sii".

Capítulo 2

Modelos causales y descubrimiento causal

Este capítulo presenta una descripción en mayor profundidad de los aspectos de la teoría de modelos causales que son relevantes para el trabajo realizado. El mismo está basado a grandes rasgos en los Capítulos 1 y 2 de [2].

Comenzaremos presentando el concepto de *relación de independencia condicional*, el cual es central a lo largo de todo el presente trabajo, y dando algunas propiedades abstractas de este tipo de relaciones, en anticipación del hecho de que nos interesará considerar nociones de independencia condicional no probabilísticas en los capítulos siguientes. A continuación, revisaremos la idea de *red bayesiana*, que no es más que una manera de representar mediante un DAG la información sobre las independencias condicionales contenidas en una distribución de probabilidad. En ese sentido, se buscará enfatizar de una cierta relación de *compatibilidad* entre un grafo y una distribución de probabilidad conjunta, por sobre el concepto de red bayesiana como una única entidad. La terminología de grafos y observaciones compatibles es transportable al caso en que las observaciones en base a las cuales se busca realizar una inferencia no son distribuciones de probabilidad empíricas, sino otra clase de objetos (en particular, colecciones de strings). Por el contrario, la terminología de redes bayesianas implica asumir que el tratamiento de las variables es probabilístico, lo cual no resulta un punto de vista suficientemente general. Una herramienta fundamental para estudiar la compatibilidad entre grafos y distribuciones es el *criterio de d-separación*, el cual también jugará un rol importante en el Capítulo 4.

Una vez introducida y explorada esta noción de compatibilidad entre grafos y distribuciones de probabilidad, procedemos a introducir la noción de modelo causal empleada

en este trabajo y discutir el contenido causal de estas representaciones. Luego, presentaremos la idea de causalidad *inferida* y un algoritmo para inferir relaciones causales a partir de datos observacionales en base al criterio de d-separación, tarea conocida como *descubrimiento causal*.

2.1. Independencia condicional

Uno de los conceptos fundamentales para este trabajo es el de *independencia condicional*. Intuitivamente, X es independiente de Y dado Z cuando, conociendo el valor de Z, saber Y no nos dice nada nuevo sobre X (y viceversa). Usamos el símbolo $\perp\!\!\!\perp$ para representar independencia. Así, lo que estamos describiendo es una relación abstracta de tres variables, que denotamos por $(- \perp\!\!\!\perp - | -)$.

El caso más conocido es el de la independencia condicional probabilística. Sin embargo es posible abstraer las propiedades relevantes de esta relación en términos de un conjunto de axiomas que también caracterizan otras relaciones de relevancia informacional. De hecho, anticipando la introducción del criterio de d-separación en la Sección 2.2.1, estos axiomas son satisfechos en el contexto de un DAG si se interpreta $(A \perp\!\!\!\perp B|C)$ como "todos los caminos de vértices en X a vértices en Y son d-separados por vértices en C". Importantemente, también son satisfechos de forma aproximada por las afirmaciones de independencia basadas en información mutua algorítmica [8], concepto que será introducido en el Capítulo 3.

Para poder introducir estos axiomas, extendemos la relación de independencia condicional de ser una relación de tres variables a ser una relación que toma tres *conjuntos* de variables. Es decir que X, Y, Z pasan a referir a conjuntos. Formalmente, pasamos de considerar una relación $R \subseteq V \times V \times V$ a considerar una relación $R \subseteq \mathcal{P}(V) \times \mathcal{P}(V) \times \mathcal{P}(V)$.

La idea es que estos axiomas caracterizan lo que es un *grafoide*: un conjunto de afirmaciones de la forma "el conjunto X es irrelevante para el conjunto Y dado que se sabe Z". El nombre proviene de la interpretación de la relación de independencia condicional en términos del criterio de d-separación en grafos (como veremos en la Sección 2.2.1).

Definición 2.1.1 (Axiomas de grafoide)**.** Dado un conjunto de variables V, sea R una relación ternaria $R \subseteq \mathcal{P}(V) \times \mathcal{P}(V) \times \mathcal{P}(V)$ donde $\mathcal{P}(V)$ es el conjunto de partes de V. Escribimos $(X \perp\!\!\!\perp Y|Z)$ para denotar $(X, Y, Z) \in R$.

Decimos que R satisface los *axiomas de grafoide* si cumple las siguientes propiedades:

- **Simetría**: $(X \perp\!\!\!\perp Y|Z) \implies (Y \perp\!\!\!\perp X|Z)$.
- **Descomposición**: $(X \perp\!\!\!\perp YW|Z) \implies (X \perp\!\!\!\perp Y|Z)$.

- **Unión débil**: $(X \perp\!\!\!\perp YW|Z) \implies (X \perp\!\!\!\perp Y|ZW)$.
- **Contracción**: $(X \perp\!\!\!\perp Y|Z)$ y $(X \perp\!\!\!\perp W|ZY) \implies (X \perp\!\!\!\perp YW|Z)$.
- **Intersección**: $(X \perp\!\!\!\perp W|ZY)$ y $(X \perp\!\!\!\perp Y|ZW) \implies (X \perp\!\!\!\perp YW|Z)$.

Si R satisface todas las propiedades salvo la propiedad de intersección, se dice que satisface los axiomas de *semi-grafoide*.

Si interpretamos la relación de independencia condicional de forma probabilística, tenemos que dada una distribución de probabilidad conjunta p, tenemos que $(X \perp\!\!\!\perp Y|Z)_p$ sii $\Pr(X = x|Y = y, Z = z) = \Pr(X = x|Z = z)$ para todos los valores x, y, z tales que $\Pr(Y = y, Z = z) > 0$ (el subíndice en la relación de independencia condicional denota con respecto a qué distribución se enuncia la afirmación). Así, la relación $(- \perp\!\!\!\perp -|-)_p$ satisface los axiomas de semi-grafoide, y el axioma de intersección es válido si P es estrictamente positiva. En muchos casos es sensato restringirse a las distribuciones de probabilidad estrictamente positivas dado que la aparición de probabilidades nulas responde generalmente a vínculos lógicos o definicionales que pueden tenerse en cuenta redefiniendo el conjunto de variables aleatorias o bien sus rangos.

La noción de independencia condicional incluye como caso particular a la noción de independencia no condicional, que corresponde a condicionar sobre el conjunto vacío. Llamamos *independencia marginal* a este tipo de independencia, y usamos la notación $(X \perp\!\!\!\perp Y)$ como sinónimo de $(X \perp\!\!\!\perp Y|\emptyset)$.

2.2. Compatibilidad de Markov

Dada una distribución de probabilidad conjunta p sobre n variables, siempre es posible factorizarla de distintas maneras. Por ejemplo, dado un ordenamiento arbitrario de las variables $X_1, \ldots X_n$, la regla del producto para probabilidades siempre nos permitirá escribir

$$p(x_1, \ldots x_n) = \prod_{j=1}^{n} p(x_j|x_1, \ldots x_{j-1}) \tag{2.1}$$

mientras que una expresión para una factorización genérica sería

$$p(x_1, \ldots x_n) = \prod_{j=1}^{n} p(x_j|x_{j,1}, \ldots x_{j,K_j}) \tag{2.2}$$

donde en el j-ésimo factor se condiciona sobre cierto subconjunto de variables $S_j = \{X_{j,1}, \dots X_{j,K_j}\}$. El punto clave es que estos subconjuntos pueden ser más pequeños que los de una factorización de la forma de la Ecuación 2.1, la cual siempre es válida para cualquier distribución. Más aún, es claro que no hay una factorización única que sea "la correcta".

Por otro lado, dado un DAG G con n vértices, asociando a cada vértice una variable aleatoria es posible leerlo como una *propuesta de factorización*. Así, si llamamos $PA_j = PA_j^{(1)}, \dots PA_j^{(m)}$ a la variable aleatoria compuesta por los m padres del nodo j, el cual corresponde a su vez a la variable aleatoria X_j, la propuesta de factorización según G es

$$p(x_1, \dots x_n) = \prod_{j=1}^{n} p(x_j | pa_j) \tag{2.3}$$

donde usamos la notación compacta $p(x_j|pa_j) = p(x_j|PA_j^{(1)} = pa_j^{(1)}, \dots PA_j^{(m)} = pa_j^{(m)})$. Esta propuesta de factorización puede corresponder o no a una factorización válida de p. Estamos en condiciones de definir, pues, una cierta noción de compatibilidad entre grafos y distribuciones de probabilidad.

Definición 2.2.1 (Compatibilidad de Markov)**.** Si una distribución de probabilidad p sobre n variables admite una factorización de la forma 2.3 con respecto al DAG G, decimos que p y G son *Markov-compatibles*.

A modo de ejemplo, consideremos el grafo instrumental presentado en el Capítulo 1 (Figura 1.1). En este caso, si las variables A, B, X e Y están distribuidas según una distribución p que es Markov-compatible con el grafo, tenemos que

$$p(a, b, x, y) = p(y|x, b)\ p(x|a, b)\ p(a)\ p(b). \tag{2.4}$$

El término *red bayesiana* es empleado para referirse a un DAG G Markov-compatible con una cierta distribución p, con la condición adicional de que los conjuntos PA_j sean minimales, i.e. que remover cualquier arista de G haga que el grafo deje de ser Markov-compatible con p. En el grafo de ejemplo, siguiendo alguna de las interpretaciones posibles dadas en el Capítulo 1, puede ser tentador asignar a estas aristas dirigidas un significado de causalidad. Sin embargo es importante resaltar que hasta ahora no se ha introducido ningún concepto relacionado con la causalidad propiamente, sino que la condición de compatibilidad de Markov refiere exclusivamente a si un grafo representa una factorización válida de una distribución de probabilidad.

2.2.1. d-separación

Para saber si un grafo y una distribución son Markov-compatibles, la única información que necesitamos sobre la distribución es la lista de relaciones de independencia condicional de la forma $(X_i \perp\!\!\!\perp X_j | \{Z_1, \dots Z_m\})$ que son inducidas por dicha distribución. Sería conveniente, pues, poder extraer también una lista de independencias condicionales a partir del grafo G, de forma tal que chequear si G y p son Markov-compatibles se reduzca a verificar que las independencias inducidas por G son satisfechas por p. Esto es posible a través del criterio de d-separación.

Definición 2.2.2 (d-separación)**.** Sea G un DAG. Decimos que un camino (no dirigido) es *d-separado* o *bloqueado* por un conjunto de vértices Z si y solo si se cumple por lo menos una de las siguientes condiciones:

1. el camino contiene dos aristas sucesivas de la forma $i \to m \to j$ ("cadena") o $i \leftarrow m \to j$ ("bifurcación") tales que el vértice intermedio m está en Z.

2. el camino contiene dos aristas sucesivas de la forma $i \to m \leftarrow j$ ("colisión") tales que ni el vértice intermedio m ni ninguno de sus descendientes está en Z.

Decimos que dos conjuntos de vértices X e Y son d-separados o bloqueados por Z si Z bloquea todos los caminos entre un vértice de X y uno de Y. En tal caso, notamos $(X \perp\!\!\!\perp Y | Z)_G$.

La interpretación es que el conjunto Z es un conjunto sobre el cual realizamos un condicionamiento, y el criterio de d-separación constituye una noción de independencia condicional formulada íntegramente en términos de la estructura del grafo G (lo cual fue anticipado en la Sección 2.1). En el ejemplo del grafo instrumental (Figura 1.1), $\{X, B\}$ bloquea a A de Y, pero no así el conjunto $\{X\}$. Esto es así pues, si bien condicionar sobre X bloquea el camino $A \to X \to Y$, simultáneamente "abre" o desbloquea el camino $A \to X \leftarrow B \to Y$. Probabilísticamente esto corresponde a la idea de que dos variables que son marginalmente independientes pueden correlacionarse al condicionar sobre una tercera variable. Este patrón de dependencias es conocido como la "paradoja de Berkson" [2] en estadística, y está detrás de fenómenos aparentemente paradójicos como el hecho de que si se selecciona a individuos de una población que presentan un rasgo A o bien un rasgo B (o ambos), dichos rasgos aparecerán negativamente correlacionados en la población seleccionada.

Habiendo definido una noción de independencia condicional intrínseca a un DAG, es posible establecer un vínculo entre esta y la independencia condicional probabilística:

Teorema 2.2.1. *Dados subconjuntos disjuntos X, Y, Z de los vértices de un DAG G, se tiene que*

1. $(X \perp\!\!\!\perp Y|Z)_G \implies (X \perp\!\!\!\perp Y|Z)_p$ *para toda distribución p Markov-compatible con G.*

2. *Si vale que $(X \perp\!\!\!\perp Y|Z)_p$ para toda distribución p que sea Markov-compatible con G, entonces $(X \perp\!\!\!\perp Y|Z)_G$.*

De esta manera, las relaciones de independencia inducidas por un DAG con vértices $v_1, \ldots v_n$ son precisamente las relaciones de independencia probabilística compartidas por todas las distribuciones sobre variables aleatorias $V_1, \ldots V_n$ Markov-compatibles con G.

Empleando este resultado, es posible establecer otra versión equivalente de la compatibilidad de Markov.

Teorema 2.2.2 (Condición de Markov local)**.** *Una distribución p y un DAG G son Markov-compatibles si y solo si cada variable X_j es condicionalmente independiente de sus no descendientes según G, dados sus padres PA_j (donde excluimos a X_j del conjunto de sus no descendientes).*

2.2.2. Equivalencia de Markov

Como ya se mencionó antes, una misma distribución de probabilidad puede admitir diferentes factorizaciones, y esto implica que la información sobre independencias condicionales en una distribución de probabilidad no necesariamente determina un único DAG Markov-compatible. En general, muchos DAGs pueden ser compatibles con una misma distribución.

Definición 2.2.3. Dos DAGs sobre un mismo conjunto de nodos son *observacionalmente equivalentes* o *Markov-equivalentes* si y solo si toda distribución de probabilidad Markov-compatible con uno de ellos, también lo es con el otro.

La relación entre DAGs dada por ser Markov-equivalentes es, naturalmente, una relación de equivalencia. El criterio de d-separación permite dar un criterio para la equivalencia de Markov en términos de la estructura de los grafos únicamente. Para esto se introduce la noción de v-estructura o estructura en forma de v.

Definición 2.2.4. Una *v-estructura* en un DAG G consiste en un par de aristas distintas que convergen sobre un mismo vértice de llegada y cuyos vértices de partida no están conectados por una arista.

Teorema 2.2.3. *Dos DAGs sobre un mismo conjunto de nodos son Markov-equivalentes si y solo si poseen el mismo esqueleto no dirigido y los mismos conjuntos de v-estructuras.*

Retomando el ejemplo del grafo instrumental, vemos que este DAG posee únicamente una v-estructura, $A \rightarrow X \leftarrow B$. Si consideramos un grafo modificado en el cual la arista entre X e Y se ve invertida, obtenemos una nueva v-estructura, $A \rightarrow X \leftarrow Y$. Por lo tanto, una distribución de probabilidad $p(a, b, x, y)$ Markov-compatible con el grafo instrumental no será Markov-compatible con este grafo modificado. Podríamos pensar entonces en invertir la arista $B \rightarrow Y$ obteniendo una arista $Y \rightarrow B$, pero en este caso se formaría un ciclo dirigido y el grafo resultante ya no sería un DAG. De esta forma podemos concluir que el grafo instrumental es el único DAG en su clase de equivalencia. Podríamos decir entonces que, en este caso, la información observacional sobre frecuencias de ocurrencia conjunta que poseemos sobre las variables A, B, X, e Y es suficiente para "determinar" la direccionalidad de la flecha entre Y y B.

En el caso general, una distribución de probabilidad no determina un único DAG. Pero si los DAGs Markov-compatibles con los datos son tan solo maneras distintas pero equivalentes de representar las dependencias presentes en esos datos, ¿Por qué nos importaría la pregunta sobre si estos determinan un único DAG? ¿Cuál sería la importancia de la direccionalidad de las flechas? La intuición detrás de esta preocupación consiste en interpretar las flechas en un DAG en términos causales, y no meramente asociativos. Esto es lo que discutimos a continuación.

2.3. Modelos causales

Dado un DAG Markov-compatible con una distribución, es posible darle una interpretación causal. Esto consiste en asignar a las aristas dirigidas el significado de relación causal directa: $U \rightarrow V$ indica que U es una causa directa de V, y V un efecto directo de U.

Siguiendo la presentación de Pearl, la interpretación causal de un DAG tiene dos aspectos centrales. El primero es que explica por qué nos preocupa intuitivamente distinguir entre DAGs Markov-equivalentes, y por qué algunas factorizaciones de la misma distribución de probabilidad nos parecen más naturales que otras: una factorización nos parece intuitiva cuando las flechas en el DAG correspondiente coinciden con relaciones causales en el fenómeno representado. Es decir, Pearl afirma que son las intuiciones causales las que sostienen, en verdad, nuestras intuiciones sobre cómo están o no asociadas las distintas variables, y las dependencias asociativas que no corresponden con las conexiones

que intuitivamente vemos como causales nos resultan extrañas, espurias o paradójicas. Un ejemplo sencillo es de esto es la ya mencionada "paradoja de Berkson" en la que dos variables se vuelven dependientes al condicionar sobre una tercera. Esta dependencia condicional no tiene nada de contradictorio, pero resulta paradójica en la medida en que no hay una relación causal asociada. En este sentido, los juicios requeridos para la construcción de un modelo causal, en oposición a uno meramente asociativo, son según Pearl más significativos y accesibles, y por lo tanto más confiables.

El segundo aspecto de este cambio de interpretación consiste en que un modelo causal no solo organiza afirmaciones *observacionales* sobre el sistema al ser observado, sino que también codifica información *intervencional*. Es decir, un modelo causal no solo afirma ciertas probabilidades condicionales entre las variables, sino que también afirma cuál sería la nueva distribución de probabilidad conjunta sobre las variables *si se fijara de forma arbitraria y exógena el valor de una o más variables*. Este acto de fijar el valor de una o más variables *rompiendo* la relación entre dichas variables y sus variables padre recibe el nombre de *intervención*. Cabe destacar que esta operación es conceptual y no tiene por qué corresponder a la realización de un experimento o de una perturbación real del sistema que está siendo modelado.

Introduzcamos esta idea informalmente a través de nuestro ejemplo recurrente. Si el grafo instrumental es Markov-compatible con una distribución p, tenemos entonces la factorización $p(a, b, x, y) = p(y|x, b)\ p(x|a, b)\ p(a)\ p(b)$. Si ahora dotamos a cada flecha en el DAG, por ejemplo $X \to Y$, de una interpretación causal, lo que estamos haciendo es introducir una serie de hipótesis (causales) que nos permiten tener información sobre cuál sería la distribución de probabilidad de Y si fijáramos externamente el valor de X. Así, si fijamos intervencionalmente $X = x$ (en el ejemplo clínico, fijar exógenamente el tratamiento realizado a x), la nueva distribución de probabilidad sobre las demás variables se obtiene mediante la *factorización truncada*

$$p_{X=x}(a, b, y) = \begin{cases} p(y|x, b)\ p(a)\ p(b) & \text{si } a, b, y \text{ son consistentes con } X = x \\ 0 & \text{si no} \end{cases} \tag{2.5}$$

¿Por qué el hecho de asumir que las flechas representan influencias causales implica la posibilidad de obtener estas distribuciones intervencionales? El punto clave en el razonamiento es que cada relación causal proviene de un *mecanismo* subyacente, y que los mecanismos son *estables* ante cambios de situación e *independientes* de cambios en los demás mecanismos. Es por eso que el efecto de una intervención es *local*: la intervención

"corta" el lazo entre una variable y sus variables padre, modificando el mecanismo que genera su comportamiento, pero sin modificar el resto de los mecanismos causales codificados por el DAG. G interpretado de este modo recibe el nombre de *red Bayesiana causal*.

* * *

Las intervenciones son una parte central de la teoría de modelos causales, la cual sin embargo resulta secundaria para el presente trabajo. Es por esto que preferimos no profundizar en la exposición de la definición intervencional de los modelos causales, introduciendo en cambio la versión *funcional* (o *estructural*) de los mismos. Esta será para nosotros la noción fundamental dado que es la que se generaliza de manera directa al caso algorítmico.

Definición 2.3.1 (Modelo causal (funcional))**.** Consideremos un conjunto de variables aleatorias $V = \{X_1, \dots X_n\}$. Un *modelo causal funcional* o *estructural* o simplemente *modelo causal* sobre V consiste en lo siguiente:

1. Un conjunto de variables aleatorias $\mathbb{U} = U_1, \dots U_n$ que representan términos de ruido debido a factores omitidos en el modelo y son conjuntamente independientes, i.e. $U_i \perp\!\!\!\perp \{U_j : j \neq i\}$.

2. Un DAG G, cuyos vértices corresponden a las variables V y cuyas aristas asocian a cada variable X_i un conjunto de variables padre $PA_i \subseteq V$. G recibe el nombre de *estructura causal subyacente*.

3. Una tupla de n funciones $\mathcal{F} = (f_1, \dots f_n)$, llamadas *funciones estructurales*, tales que
$$X_i = f_i(PA_i, U_i),\ i = 1, \dots n. \tag{2.6}$$

Alternativamente, podemos adoptar el punto de vista de que esta tupla $(G, \mathcal{F}, \mathbb{U})$ define las variables aleatorias V junto con su probabilidad de distribución conjunta.

La hipótesis de que las variables de ruido U_i son conjuntamente independientes recibe el nombre de *suficiencia causal*, dado que expresa que el conjunto de variables V elegido constituye un recorte válido del fenómeno bajo análisis. Si las variables de ruido no fueran independientes, entonces esto se debería a la existencia de variables adicionales que afectan simultáneamente a distintas variables de V, y por lo tanto esas variables

deberían haber sido incluidas en la descripción[1]. Esta hipótesis es fundamental para el siguiente resultado:

Teorema 2.3.1 (Condición de Markov causal)**.** *Todo modelo causal* $M = (G, \mathcal{F}, \mathbb{U})$ *induce una distribución de probabilidad sobre las mismas que es Markov-compatible con* G*.*

El teorema muestra que la compatibilidad de Markov surge como consecuencia de dos hipótesis:

1. Suficiencia causal. Toda variable que es causa de dos o más variables está incluida en el modelo.

2. Principio de Reichenbach. Si dos variables son dependientes, entonces una es causa de la otra o bien existe una tercera variable que es causa de las dos.

Estas hipótesis son ambas de naturaleza causal, y no asociativa o estadística. Por otro lado, todo par compuesto por una distribución p y un DAG G Markov-compatibles admite una representación como modelo funcional, i.e. existe un modelo funcional con diagrama causal G que induce la distribución p [16].

Si bien nos concentraremos en el aspecto observacional de los modelos causales, podemos mencionar el hecho de que las intervenciones presentadas anteriormente pueden ser expresadas de manera extremadamente natural en el contexto de modelos causales: la intervención $X_j = x$ consiste simplemente en reemplazar la ecuación estructural $X_j = f_j(PA_j, U_j)$ por la ecuación $X_j = x$. La distribución de probabilidad intervenida se obtiene simplemente como la distribución inducida por este nuevo modelo causal. Más aún, los modelos causales habilitan un tipo de razonamiento totalmente nuevo, que no puede realizarse en el formalismo de redes Bayesianas causales. Se trata del razonamiento *contrafáctico*, en el cual se busca determinar si el valor adoptado por una variable es *debido* al valor que adoptaron sus causas, *a pesar* de este o *indistintamente*. Esto implica juzgar la probabilidad de un evento bajo condiciones no observadas, como por ejemplo la probabilidad de que una persona que respondió negativamente a un tratamiento se habría recuperado si no hubiera recibido dicho tratamiento. A pesar de la riqueza de este aspecto de los modelos causales, el mismo queda fuera del alcance de nuestra exposición.

[1] El formalismo permite contemplar la existencia de variables latentes o inobservables, pero esto se hace de un modo diferente.

2.4. Descubrimiento causal

En el presente trabajo buscamos realizar inferencia a partir de datos observacionales. Se tiene un conjunto de variables V, y el objetivo es reconstruir un DAG G que represente la estructura causal de V. Esta tarea es denominada *descubrimiento causal*.

Como fue establecido anteriormente, para inferir relaciones causales es necesario partir de hipótesis causales también. Una primera hipótesis, que ya fue introducida previamente, es la de suficiencia causal: asumimos que todas las variables que son causa de dos o más variables en V, también están en V.

En un segundo lugar, asumimos una hipótesis de *minimalidad* que puede ser entendida como una variante de la Navaja de Ockham. Esta hipótesis puede ser resumida en el concepto de *causalidad inferida*, según el cual una variable C tiene una influencia causal sobre una variable E precisamente cuando existe un camino dirigido desde C hasta E en todas las estructuras causales *minimales* compatibles con los datos. Para nosotros, esto quiere decir que asumimos que la estructura causal que buscamos reconstruir es tal que si se elimina alguna arista, deja de ser Markov-compatible con los datos, i.e. se trata de una estructura minimal. Dicho de otra manera, no podremos inferir estructuras que no sean minimales. Pearl enfatiza que esta definición de causalidad inferida no tiene por qué identificar los mecanismos estables de la naturaleza que forman la estructura causal subyacente a los datos, en la medida en que dicha estructura puede no ser minimal. En cambio, esta definición de causalidad inferida caracteriza simplemente a los mecanismos que es plausible inferir a partir de información observacional.

Finalmente, la tercera hipótesis de índole causal es la hipótesis de *estabilidad* (también denominada *fidelidad*). Como fue establecido en el Teorema 2.2.1, que un DAG G sea Markov-compatible con una distribución p implica que $(X \perp\!\!\!\perp Y|Z)_G \implies (X \perp\!\!\!\perp Y|Z)_p$ para cualquier elección de conjuntos de variables disjuntos X, Y, Z. El mismo teorema muestra que la afirmación opuesta no es estrictamente cierta: $(X \perp\!\!\!\perp Y|Z)_p$ no necesariamente implica que X e Y sean d-separados por Z en G. Sin embargo, esto sí ocurre cuando la independencia $(X \perp\!\!\!\perp Y|Z)$ es válida no solo en p, sino en todas las distribuciones Markov-compatibles con el mismo DAG G.

La hipótesis de estabilidad consiste en suponer que la distribución p satisface *únicamente* las independencias condicionales implicadas por G, o equivalentemente, que las independencias en p son únicamente aquellas compartidas por todas las distribuciones Markov-compatibles con G. Expresamos esto en términos de modelos causales:

Definición 2.4.1 (Estabilidad)**.** Sea $M = (G, \mathcal{F}, \mathbb{U})$ un modelo causal y sea $p_{(G,\mathcal{F},\mathbb{U})}$ la distribución de probabilidad inducida por M. Denotemos por $I(p)$ el conjunto de todas

las independencias condicionales satisfechas por una distribución p.

Decimos que *M genera una distribución estable* si y solo si

$$I\left(p_{(G,\mathcal{F},\mathbb{U})}\right) \subseteq I\left(p_{(G,\mathcal{F}',\mathbb{U}')}\right) \tag{2.7}$$

para cualquier elección alternativa de funciones estructurales $\mathcal{F}'$ y variables de ruido $\mathbb{U}'$.

Es decir que la distribución generada por M es estable si no es posible destruir ninguna independencia condicional de dicha distribución variando los parámetros del modelo. Gracias al Teorema 2.2.1, y usando el hecho de que todo modelo causal genera una distribución Markov-compatible con su estructura causal subyacente (Teorema 2.2.1) tenemos la siguiente caracterización.

Proposición 2.4.1. *Sea $I(G)$ el conjunto de relaciones de independencia condicional inducidas por G mediante el criterio de d-separación. La distribución p generada por un modelo causal M con estructura causal G es estable si y solo si $I(G) = I(p)$.*

Al suponer que los datos observados provienen de una distribución que es estable con respecto un cierto modelo causal subyacente, estamos suponiendo que las relaciones de independencia condicional plasmadas en los datos son las mismas que las relaciones de independencia en el DAG subyacente que queremos inferir, y no hay independencias "extrañas" en los datos.

2.4.1. El algoritmo IC

Entre los distintos abordajes posibles a la tarea de descubrimiento causal, puede distinguirse la clase de métodos de optimización y la clase de métodos basados en vínculos. Un método de optimización realiza una búsqueda del DAG G (o más precisamente una clase de equivalencia de DAGs Markov-compatibles, como veremos luego) mediante minimización de alguna función de costo. En cambio, un método basado en vínculos busca construir la solución a partir de las relaciones de independencia condicional contenidas en los datos. Para esto, es necesario emplear algún método que testee dichas relaciones de independencia entre las variables, i.e. que permita a partir de los datos estimar de forma confiable las relaciones de independencia condicional satisfechas por la distribución de probabilidad a partir de la cual los datos fueron obtenidos.

El algoritmo IC pertenece a esta segunda clase de métodos[2]. Este algoritmo, presentado originalmente en [17], devuelve un grafo G consultando únicamente las relaciones de

independencia condicional entre las variables. Dado que asumimos que podemos consultar estas relaciones para todas las variables en V, asumimos en nuestro caso consideramos que no hay variables latentes[3]. Presentamos el algoritmo en forma de pseudocódigo como Algoritmo 1.

La correctitud del algoritmo asume los postulados de suficiencia causal, minimalidad y estabilidad. Concretamente, se supone que los datos provienen de un modelo causal M_0 con estructura causal G_0 que es minimal y genera una distribución estable.[4] Como en general existen múltiples estructuras causales Markov-equivalentes a G_0, y por lo tanto no pueden ser distinguidas de G_0 mediante observaciones, lo máximo a lo que puede aspirarse a partir de dicha información es a identificar la clase de equivalencia de G_0.

Algoritmo 1: Algoritmo IC

Entrada: p, una distribución estable sobre el conjunto de variables V.
Salida: Un patrón $H(p)$ Markov-compatible con p.

1.1 **para cada** $\{a, b\} \subseteq V, a \neq b$ **hacer**
  buscar $S_{ab} \subseteq V - \{a, b\}$ tales que vale $(a \perp\!\!\!\perp b | S_{ab})_p$
1.2 construir grafo no dirigido G tal que a y b están conectados sii no se encontró un conjunto S_{ab}.
2 **para cada** $\{a, b\} \subseteq V, a \neq b$ **hacer**
  si *a, b no son adyacentes en G y tienen un vecino común c* **entonces**
    si $c \notin S_{ab}$ **entonces**
      orientar las aristas $a{-}c$ y $b{-}c$ apuntando a c $(a \rightarrow c \leftarrow b)$.
3 En el grafo parcialmente dirigido resultante, orientar tantas aristas no dirigidas como sea posible, sujeto a dos condiciones:
  (i) Cualquier orientación alternativa daría lugar a una nueva v-estructura
  (ii) Cualquier orientación alternativa daría lugar a un ciclo dirigido.

Llamamos *patrón* a una representación gráfica de esta clase de equivalencia, que consiste en un DAG *semidirigido*, es decir en el cual algunas aristas son dirigidas mientras que otras no lo son. Las aristas dirigidas representan flechas que son comunes a todos los DAGs Markov-equivalentes a G_0, mientras que las aristas no dirigidas representan ambivalencia.

Así, la garantía de correctitud es que si la entrada del algoritmo es una distribución de probabilidad p estable con respecto a un modelo causal minimal con estructura causal

[2] En gran parte de la literatura, se refiere no a este algoritmo sino a un refinamiento del mismo conocido como *algoritmo PC*, el cual fue introducido en [18].
[3] Este mismo algoritmo puede ser modificado para considerar variables latentes.
[4] La hipótesis de suficiencia causal está contenida en la definición de modelo causal.

G_0, entonces la salida es un patrón que representa la clase de equivalencia de Markov de G_0.

Finalmente, notemos que en esta manera de expresar el algoritmo, la entrada es la distribución de probabilidad generada por el modelo subyacente, mientras que en la práctica lo que se posee es un conjunto de observaciones, las cuales (siempre dentro del contexto probabilístico que estamos describiendo en este capítulo) pueden sintetizarse en términos de una *distribución de probabilidad empírica* $\hat{p}$ que le asigna a cada evento la frecuencia con la que fue observado. La garantía de correctitud del algoritmo será válida en estos casos siempre y cuando esta distribución empírica aproxime "suficientemente bien" a la distribución real p. En particular es una condición suficiente pedir que $I(p) = I(\hat{p})$.

2.5. Resumen

En este capítulo presentamos algunos aspectos básicos de la teoría de modelos causales, relevantes para el desarrollo del trabajo. En primer lugar, dimos una formulación abstracta del concepto de relación de independencia condicional. Luego, realizamos un camino progresivo hasta llegar al concepto de modelo causal.

Cuando un grafo representa representa una factorización válida de una distribución de probabilidad, decimos que el grafo y la distribución son Markov-compatibles. Cuando el grafo es minimal sobre todos los grafos Markov-compatibles con la distribución, se dice que es una red Bayesiana para la distribución.

Una red bayesiana causal es una red bayesiana en la que las aristas se interpretan como relaciones de causalidad inmediata, convirtiéndose entonces en una herramienta para el cálculo de distribuciones de probabilidad intervencionales en las que se fija exógenamente el valor que adoptan una o más variables. Finalmente, un modelo causal funcional, modelo causal estructural o simplemente modelo causal, le asigna una función a cada nodo, de forma tal que cada nodo se obtiene como función de de sus padres y de una variable aleatoria de ruido. La definición de modelo causal asume implícitamente la hipótesis de suficiencia causal del conjunto de variables observado (todas las variables que son causa de dos o más variables bajo estudio, deben estar incluidas también en el modelo).

Asumiendo que un conjunto de datos dado proviene de un modelo causal subyacente, y asumiendo además las hipótesis causales de *minimalidad* y *estabilidad* de dicho modelo, presentamos un algoritmo que recupera la estructura causal subyacente a dicho modelo a partir de los datos. Por supuesto, como las relaciones de independencia condicional codificadas en un DAG lo especifican tan solo a menos de Markov-equivalencia, el resultado

de este algoritmo consiste en verdad en una clase de equivalencia de DAGs.

Capítulo 3

Información algorítmica

En este capítulo se introduce la Teoría Algorítmica de la Información como un marco teórico general para cuantificar la cantidad de información o estructura contenida en objetos finitos representados por cadenas de símbolos (*strings*). Esto da lugar a una noción sintáctica de correlación entre objetos basada en su estructura compartida.

Empleamos los adjetivos "sintáctico" o "algorítmico" para referirnos a estructura presente en objetos simbólicos que excede las propiedades estadísticas del objeto, tal como la frecuencia de aparición de distintas combinaciones de símbolos[1]. Estas palabras refieren a *reglas* según las cuales una expresión simbólica debe estar formada para ser válida, o, dicho de otra manera, a la *forma* de estas expresiones.

Para dar intuición sobre esta idea, consideremos tres strings binarios de la misma longitud. El primero es 0101010101..., mientras que el segundo es una codificación en binario del primer capítulo del Quijote. El tercero es una codificación de los *dos* primeros capítulos del Quijote, comprimidos de forma tal que forman un string de la misma longitud que los anteriores. Intuitivamente, cada uno de estos strings tiene más información que el anterior, y esta es la noción de cantidad de información formalizada por la información algorítmica[2].

Las Secciones 3.1 y 3.2 están basadas, en líneas muy generales, en el Capítulo 1 de [9]. La Sección 3.3 está basada en partes de los Capítulos 1, 2 y 3.

[1] Las propiedades estadísticas de un objeto pueden verse como un caso particular de propiedades sintácticas.

[2] Las propiedades estadísticas de estos strings son indicativas de la cantidad de información que poseen, pero la información algorítmica considera también regularidades no estadísticas.

3.1. Preliminares

¿Qué es la estructura de un conjunto de datos, o de un objeto cualquiera? ¿Podemos encontrar alguna magnitud numérica que sea indicativa de todos los patrones existentes (o encontrables) en un objeto dado? La Teoría Algorítmica de la Información responde a estas preguntas de manera sorprendentemente general, aunque la respuesta no es tan satisfactoria como podría desearse *a priori*: la magnitud por la que nos preguntamos existe matemáticamente, pero no es posible calcularla.

Una respuesta muy informal a la pregunta planteada, que nos servirá de punto de partida, puede formularse de la siguiente manera:

Postulado 3.1.1. *Un objeto simple es uno que puede ser descrito sucintamente. De forma dual, un objeto complicado/estructurado es uno cuyas descripciones son todas largas.*

Así, vinculamos la estructura presente en un objeto con la dificultad o facilidad que tenemos para describirlo.

El concepto de *información algorítmica* formaliza esta idea, y el objetivo de esta sección es motivar su definición a partir del Postulado 3.1.1.

Para formalizar el Postulado, debemos especificar un dominio de discurso (a qué nos referimos con "objeto") y una noción de qué constituye una descripción. Esto último nos conduce a preguntarnos por el *lenguaje* en que esas descripciones son expresadas.

Para que nuestra noción de descripción nos resulte útil, requerimos que las descripciones sean *unívocas* y *finitas*. Es decir, ninguna descripción puede corresponder a más de un objeto, y por otro lado debe ser posible representar cada descripción mediante un número finito de símbolos[3]. Si además insistimos en que el alfabeto del cual se toman los símbolos sea también finito, se sigue inmediatamente que el conjunto de todas las descripciones posibles es numerable[4].

Estas condiciones sobre las descripciones imponen a su vez una condición sobre el dominio de discurso: como hay tan solo una cantidad numerable de descripciones, la cantidad de objetos descriptibles es también numerable. Esto nos habilita a identificar a estos objetos con cadenas finitas de símbolos, como es habitual en computación teórica.

[3] Más en general, podríamos considerar nociones de lenguajes que no estén basadas en el símbolo como unidad sintáctica primitiva pero eso se aleja de las teorías establecidas. En cualquier caso sería esperable que cualquier noción generalizada de lenguaje admita una manera de cuantificar la extensión de las expresiones sintácticas y distinguir las finitas de las infinitas.

[4] Que el alfabeto empleado deba ser finito puede motivarse intuitivamente considerando que si dispusiéramos de un alfabeto infinito podríamos codificar descripciones enteras en un único símbolo, pero simultáneamente la "dificultad" en distinguir un símbolo de otro podría ser arbitrariamente grande. De esta manera se estaría "escondiendo" la estructura que se busca describir dentro de las entidades atómicas.

Esto se realiza mediante la elección de alguna *codificación*, que asigne los objetos a cadenas. De esta manera, tanto los objetos como sus descripciones son representados en esta teoría por cadenas finitas de símbolos, o *strings*.

Es importante distinguir la codificación de un objeto (por ejemplo, un grafo o una tabla de datos) como string, de la descripción de dicho objeto (representado por un string) mediante otro string. Es decir, la relación "A describe a B" es una relación entre strings. Así, fijando un alfabeto Σ, el proceso de descripción es una relación $D \subseteq \Sigma^* \times \Sigma^*$: si $(s, s') \in D$, entonces s describe a s'.

Las condiciones pedidas más arriba implican que cada string describe a lo sumo un otro string. En computación teórica, en este caso es habitual no hablar de D como una mera relación sino como un objeto matemático a medio camino entre una relación general y una función, el cual se denomina función parcial.

Definición 3.1.1. Una *función parcial* $f : X \to Y$ es una relación $R \subseteq X \times Y$ tal que, para cada $x \in X$, existe a lo sumo un $y \in Y$ tal que $(x, y) \in R$. En caso de que exista un y tal, escribimos $f(x) = y$. En caso contrario, escribimos $f(x) \uparrow$ y decimos que $f(x)$ *está indefinida*.

Ahora bien, ¿Cualquier función parcial $\Sigma^* \to \Sigma^*$ constituiría un método de descripción relevante? Si bien la respuesta a esta pregunta es obviamente contextual, la clase de métodos de descripción que será considerada de ahora en adelante es la de aquellos que son *efectivos* o *efectivamente calculables*[5], es decir aquellos métodos que nos permiten recuperar el objeto descrito a partir de su descripción de una manera sistemática, siguiendo alguna serie de pasos determinada. Esto responde al criterio intuitivo de que la manera de "entender" una descripción, identificando el objeto que describe, no puede ser arbitrariamente complicada y diferente para cada descripción, sino que la dificultad de asociar descripciones a objetos descriptos debe ser en algún sentido uniforme[6]. Más aún, veremos que esta clase de funciones posee propiedades fundamentales para la teoría, tales como admitir elementos universales y elementos aditivamente óptimos.

La necesidad de precisar formalmente el significado de "efectivamente calculable" nos introduce en el terreno de la Teoría de la Computabilidad. En términos de funciones parciales, los métodos de descripción efectivos corresponderán precisamente a las *funciones parciales computables*.

[5] Usando terminología contemporánea, si bien quizás ambigua, podemos adelantar cierta intuición diciendo que estos métodos son *ejecutables*.

[6] Nos referimos a una dificultad uniforme en la *descripción* del proceso involucrado, y no a sy dificultad en términos de, por ejemplo, qué tantos recursos de tiempo y espacio son necesarios para obtener el objeto descripto. En ese sentido, un procedimiento largo pero repetitivo es simple.

Antes de introducir la Teoría de la Computabilidad, y aprovechando que las intuiciones del Siglo XXI son distintas a las intuiciones de 1930, podemos adelantar una definición provisional de información algorítmica. En efecto, estos métodos de descripción efectivos son equivalentes a cualquier lenguaje de programación convencional, tal como Python, Haskell o C. En este caso, las descripciones individuales son *programas* escritos para alguno de esos lenguajes, de forma tal que el objeto que describen es precisamente la salida del programa.

Con esta idea en mente, podemos comenzar a definir de manera provisoria la información algorítmica.

Definición 3.1.2 (Información algorítmica, informal)**.** Consideremos un lenguaje de programación convencional $\mathcal{L}$ fijo. Sea $U_{\mathcal{L}} : \Sigma^* \to \Sigma^*$ una función parcial tal que $U_{\mathcal{L}}(x) = y$ sii al ejecutar el programa con código fuente x (y con entrada vacía, i.e. sin darle ninguna entrada), la salida obtenida es y. Definimos entonces la *información algorítmica* o *complejidad de Kolmogorov* como la función

$$\begin{aligned} C &: \Sigma^* \to \mathbb{N} \\ C(x) &= \min_{p \in \Sigma^*} \{|p| : U_{\mathcal{L}}(p) = x\} \end{aligned} \tag{3.1}$$

donde $|p|$ denota la longitud del string p. En palabras, la información de x es la longitud de un programa *minimal* que lo genera como salida.[7]

En primer lugar, notemos que C está bien definida puesto que siempre existe *algún programa* que compute x como salida, para cualquier x. En particular, si x es la cadena de símbolos $s_0 s_1 \dots s_n$, podemos exhibir el programa `print("`$s_0 s_1 \dots s_n$`")`, que siempre cumple con lo pedido.

Notemos también la necesidad de permitir que $U_{\mathcal{L}}$ en esta definición sea una función parcial dado que no todos los programas detienen su ejecución una vez comenzada. Es decir, el caso $U_{\mathcal{L}}(x) \uparrow$ corresponde a un programa con código fuente x cuya ejecución continúa indefinidamente, por ejemplo al entrar en un bucle infinito. Esto será analizado en detalle más adelante.

En términos intuitivos, la información algorítmica contenida en un string x es menor mientras más *compresible* es, y $C(x)$ es el tamaño de la compresión óptima de x. Por ejemplo, si consideramos $\Sigma = \{0, 1\}$, intuitivamente podemos esperar que

[7] Podría decirse en cambio "...la longitud del programa más corto que lo genera como salida." En tal caso se incurriría en un pequeño abuso de lenguaje dado que puede haber más de un programa diferente que genere x y cuya longitud sea mínima.

$$x_k = (01)^k = \underbrace{(0101\ldots01)}_{k \text{ veces}} \tag{3.2}$$

tenga una información algorítmica baja respecto a otros strings de su misma longitud, y que de hecho dicha información no crezca mucho con k. Más precisamente, como un programa posible para x_k puede expresarse informalmente como "repetir (01) k veces", podemos decir que $C(x_k)$ crece a lo sumo logarítmicamente en k (puesto que para representar k dentro del programa, requeriremos del orden de $\log k$ símbolos, donde tomamos como la base del logaritmo al número de símbolos en el alfabeto). El "a lo sumo" responde a que en principio sería posible encontrar, para k grande, programas aún más cortos que "repetir (01) k veces".

Para expresar este tipo de afirmaciones utilizamos la notación asintótica de Bachman-Landau o notación "O grande".

Definición 3.1.3 (Notación asintótica)**.** Dadas funciones $f, g : \mathbb{R} \to \mathbb{R}$, notamos $f(x) = O(g(x))$ y decimos que $f(x)$ es *de orden* $g(x)$ sii existen constantes $x_0 \in \mathbb{R}, c > 0$ tales que

$$(\forall x \geq x_0) \quad |f(x)| \leq c\ |g(x)|. \tag{3.3}$$

Así, en el ejemplo podemos decir que $C(x_k) = O(\log k)$.

La notación asintótica será útil para enunciar muchas propiedades de la información algorítmica. Un caso frecuente es el de expresiones del tipo

$$f(x) = g(x) + O(1) \tag{3.4}$$

lo cual, reordenando como $f(x) - g(x) = O(1)$ y desempaquetando la definición, es equivalente a afirmar que existe una constante c, independiente de x, tal que $|f(x) - g(x)| < c$ para todo x a partir de un x_0 en adelante.

* * *

La definición de información algorítmica es el punto de partida de toda la Teoría Algorítmica de la Información. Más adelante la reemplazaremos por una definición más formal en términos de funciones parciales computables *universales* y *aditivamente óptimas* (que juegan el rol de los lenguajes de programación ya mencionados), y consideraremos variantes de esta definición. Ampliaremos la definición a una versión *condicional*, que nos permitirá cuantificar la correlación sintáctica. Esto corresponde al caso en que las descripciones son programas que a su vez pueden variar su salida en función de la entrada

que reciben. Por otro lado, más adelante consideraremos variantes de la información algorítmica, ligeramente más complicadas pero con mejores propiedades cuantitativas. Con estas definiciones, hablaremos de información algorítmica *mutua*, que será el concepto clave para cuantificar la correlación sintáctica.

3.2. Teoría de la Computabilidad

¿Qué quiere decir que un procedimiento para obtener un una salida a partir de una entrada sea *efectivo*? La Teoría de la Computabilidad da una respuesta a esta pregunta, la cual resulta ser notablemente robusta, y explora sus consecuencias. Una de ellas es que no todas las funciones matemáticas $\Sigma^* \to \Sigma^*$ son efectivamente calculables, i.e. algunas de ellas no pueden ser calculadas por ningún programa[8]. Este resultado es particularmente contundente cuando se recuerda que dichas funciones son equivalentes a funciones $\mathbb{N} \to \mathbb{N}$.

En la década de 1930, siguiendo la dirección de investigación planteada por Hilbert y Ackerman al proponer la búsqueda de un método efectivo para calcular la validez lógica de cualquier fórmula lógica de primer orden (propuesta conocida como el *Entcheidungsproblem*), Alan Turing y Alonzo Church desarrollaron modelos matemáticos de lo que es un procedimiento efectivamente calculable. El que consideraremos es el propuesto por Turing, el cual le asigna a cada procedimiento efectivo una *máquina* que lo realiza, tal como describiremos más adelante. A estas máquinas se las conoce hoy en día como *máquinas de Turing*.

Estos modelos resultaron ser equivalentes, i.e. capturan exactamente la misma clase de funciones efectivamente calculables, y son también equivalentes a muchos otros modelos de cómputo propuestos posteriormente. Importantemente, esta noción de función efectivamente calculable coincide con la de los lenguajes de programación convencionales de uso ubicuo hoy en día. Esta robustez, observada a lo largo de los años, otorga peso intuitivo a la idea de que todos estos modelos describen una misma noción objetiva de computabilidad, y que todo lo que es calculable en un sentido intuitivo es calculable en el sentido de alguno de estos modelos (y por lo tanto, todos ellos). Esta es la tesis de Church-Turing.

[8] En verdad, este hecho en sí mismo se desprende rápidamente de un argumento de cardinalidad, una vez que aceptamos que, sea cual sea nuestra definición de "programa", solo existe una cantidad numerable de ellos (en contraposición con el conjunto de funciones $\Sigma^* \to \Sigma^*$, que es no numerable). La Teoría de la Computabilidad formaliza la noción de programa dándole rigor a este argumento, pero en verdad provee también resultados mucho más profundos y sorprendentes.

Postulado 3.2.1 (Tesis de Church-Turing)**.** *Todo proceso que pueda ser llamado de forma intuitiva "procedimiento efectivo" es realizable por una máquina de Turing (tesis de Turing). Más aún, existe una única noción objetiva de computabilidad efectiva, la cual es independiente de la formalización particular adoptada (tesis de Church). La formalización en máquinas de Turing es tan solo una de dichas formalizaciones posibles.*

Para definir, pues, qué es una función parcial computable, necesitaremos concentrarnos en un modelo de cómputo específico. Por simplicidad y adecuación a la literatura existente en Teoría de la Computabilidad, elegimos el modelo de máquinas de Turing. Debemos entonces describir qué es una máquina de Turing, en qué consiste la *ejecución* de la misma y de qué manera esta corresponde a (o *realiza* o *computa*) una función parcial computable.

3.2.1. Máquinas de Turing

Comenzamos definiendo las máquinas de Turing de manera interpretada (i.e. apelando a la intuición de un dispositivo mecánico idealizado), para dar luego una definición formal más sucinta. Hay muchas definiciones diferentes de máquina de Turing, con propiedades diferentes pero que no modifican la clase de funciones computables que es definida por el modelo. La definición que damos resulta adecuada por su simplicidad.

El paper original de Turing presenta una exposición interesante sobre las intuiciones detrás de la idea de que estas máquinas puedan realizar cualquier cómputo que podría realizar un computador humano, y que por lo tanto capturan lo que es el proceso de cómputo efectivo. El fragmento relevante puede tomarse como motivación adicional para la definición que sigue y se encuentra citado en [9], páginas 24 a 27.

Una *máquina de Turing* consiste en un mecanismo de control, cuyos estados posibles forman un conjunto *finito* Q, que interactúa con una memoria, la cual denominamos *cinta*, leyendo y escribiendo símbolos sobre la misma (ver Figura 3.1). La cinta consiste en un ordenamiento lineal y bi-infinito de *celdas*, cada una de las cuales puede contener un símbolo del alfabeto finito y no vacío Σ. En caso contrario, representamos la celda en blanco por un símbolo distinguido "_". Resulta conveniente aumentar el alfabeto Σ a $\widetilde{\Sigma} = \Sigma \cup \{\_\}$. Matemáticamente, podemos decir que cada configuración de la cinta corresponde a un elemento de $\widetilde{\Sigma}^{\mathbb{Z}} = \{\text{funciones } f : \mathbb{Z} \to \widetilde{\Sigma}\}$.

El mecanismo de control guía el movimiento de un *cabezal* a lo largo de la cinta. La posición del cabezal señala la celda que está siendo leída (lo cual influye en la próxima acción de la máquina) y cuyo símbolo actual puede ser sobrescrito por la máquina. De esta forma, el estado *total* de la máquina queda especificado por una tupla (q, m, φ),

donde $q \in Q$ es un estado interno (estado del mecanismo de control), $m \in \mathbb{Z}$ es una posición del cabezal y $\varphi \in \widetilde{\Sigma}^{\mathbb{Z}}$ una configuración de la cinta.

La evolución temporal del sistema (en tiempo discreto) se da mediante la ejecución de una acción por cada paso de tiempo. El conjunto de acciones individuales posibles es $A = \Sigma \cup \{L, R\}$, donde L, R son símbolos que no pertenecen a Σ. La acción s para algún $s \in \Sigma$ se interpreta como escribir el símbolo s en la posición actual del cabezal, mientras que la acción L se interpreta como mover el cabezal una posición a la izquierda y R, una posición a la derecha.

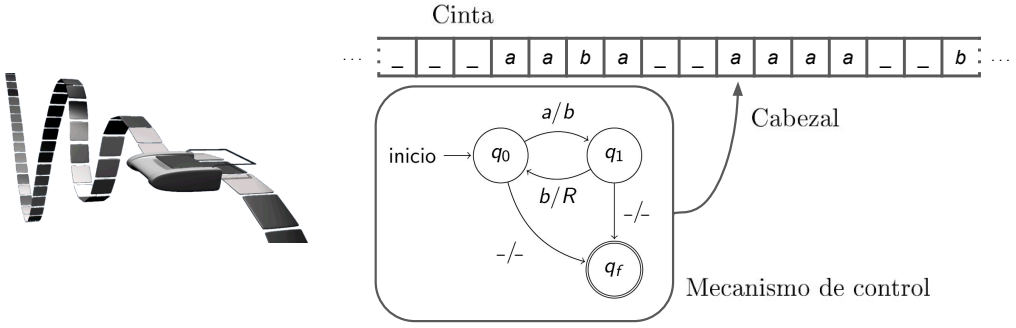


Figura 3.1: Representación artística (izq.) y esquemática (der.) de una máquina de Turing. En ambas se representa la cinta y la posición del cabezal sobre la misma, mientras que en la representación esquemática se muestra además una vista "interior" del mecanismo de control, presentando una tabla de instrucciones de ejemplo en forma de autómata. Una flecha de q a q' etiquetada con s/m representa una entrada (q, s, m, q') en la tabla de instrucciones (si la máquina está en estado interno q y el cabezal lee el símbolo s, puede realizar la acción m y cambiar al estado q'). En este ejemplo, el alfabeto es $\Sigma = \{a, b\}$, y el funcionamiento de la máquina es el siguiente: cada vez que lee una a, la sobrescribe por una b y se mueve a la derecha. Si lee una celda en blanco, se detiene instantáneamente. Finalmente, si apenas al moverse a la derecha se encuentra con una b, la ejecución se indefine (pues no hay transiciones válidas desde q_0 al leer el símbolo b.

Cada máquina de Turing está determinada por una elección de Q y de Σ, pero también y fundamentalmente por cómo el mecanismo de control está *programado*. Esta información consiste en una relación $T \subseteq Q \times \widetilde{\Sigma} \times A \times Q$ que denominamos *tabla de instrucciones*. Cada tupla $(q, s, a, q') \in T$ consiste en una instrucción para la máquina, que interpretamos de la siguiente forma: "si el estado interno de la máquina es q y el símbolo bajo el cabezal es s, entonces la máquina puede realizar la acción a y cambiar de estado interno al estado q'."

Finalmente, también requerimos que Q contenga a dos estados especiales, los cuales denotamos por q_0 y q_f, y llamamos estado *inicial* y *final*, respectivamente. La ejecución

de una máquina de Turing siempre comienza con el mecanismo de control en estado q_0, con la cinta en alguna configuración inicial $\varphi_0 \in \Sigma^{\mathbb{Z}}$ y con el cabezal en la celda de partida (que podemos identificar con $0 \in \mathbb{Z}$). La llegada al estado final q_f indica que la ejecución de la máquina ha finalizado[9].

Podemos ahora enunciar la siguiente definición sucinta.

Definición 3.2.1 (Máquina de Turing)**.** Una máquina de Turing es una tupla (Σ, Q, T, q_0, q_f), donde Σ es un conjunto finito de símbolos que no contiene a los símbolos $\{\_, L, R\}$ y Q es un conjunto finito de estados internos (ambos no vacíos), $T \subseteq Q \times \widetilde{\Sigma} \times A \times Q$ es la tabla de instrucciones de la máquina (donde $\widetilde{\Sigma} := \Sigma \cup \{\_\}$, $A := \Sigma \cup \{L, R\}$), y $q_0, q_f \in Q$ son estados internos distinguidos.

Notemos que si bien la máquina de Turing "es" (o está *especificada* por) una tupla (Σ, Q, T, q_0, q_f), lo que le da contenido a la definición es en verdad la *ejecución* de la máquina, para lo cual es indispensable la interpretación dada. La ejecución de una máquina de Turing consiste en la ejecución de instrucciones de su tabla T de manera iterativa partiendo del estado interno inicial q_0 y de la configuración de la cinta inicial φ_0: si el estado interno actual de la máquina y el símbolo bajo el cabezal están dados por $(q, s) \in Q \times \widetilde{\Sigma}$, la máquina ejecuta alguna instrucción en T que comience con (q, s) y pasa a un nuevo par de estado interno y símbolo bajo el cabezal, (q', s'), el cual depende de la instrucción ejecutada pero también de la configuración de la cinta. En el proceso, la configuración de la cinta es modificada. El proceso continúa indefinidamente, o bien hasta que la máquina arriba al estado interno q_f, en cuyo caso la ejecución *termina* y la máquina no realiza más acciones. Se obtiene en este caso una configuración final $\varphi_f \in \widetilde{\Sigma}^{\mathbb{Z}}$ de la cinta[10].

Consideremos el ejemplo dado en la Figura 3.1 y describamos cómo sería la ejecución de esa máquina. En el esquema, $\Sigma = \{a, b\}$, y el cabezal de la máquina comienza en el estado inicial q_0 sobre una celda marcada con a. La tabla de instrucciones de esta máquina contiene una instrucción (q_0, a, b, q_1), que es la única que la máquina puede ejecutar en este momento. Por lo tanto, la máquina reemplaza la a por una b y transiciona al estado interno q_1. Como siguiente paso, la máquina transiciona de nuevo al estado q_0, esta vez moviéndose una celda a la derecha (esto es, aplica la transición (q_1, b, R, q_0)). Esto

[9] Nuestra definición de máquina de Turing y de su ejecución es ligeramente distinta de la dada en [9]. Por ejemplo, en nuestra definición cada máquina tiene un estado final y la ejecución solo se detiene si la máquina entra en dicho estado; por otro lado la ejecución se indefine si no hay instrucciones que puedan ser ejecutadas desde el estado actual. En contraposición, en [9] no se define un estado final, sino que por el contrario se considera que la ejecución se detiene en el momento en que no hay instrucciones válidas que puedan ser ejecutadas. Estas diferencias no afectan el poder de cómputo del modelo resultante.

[10] La configuración final de la cinta no debe ser confundida con la *salida* de la máquina. Esta es especificada más adelante en la Definición 3.2.3.

ocurre tres veces más, hasta que finalmente el cabezal llega a una celda en blanco al mismo tiempo que el mecanismo de control vuelve a q_0. En este momento, la máquina sobreescribe _ por _ (que es lo mismo que no hacer nada) y entra al estado final q_f. El resultado neto es que el string *aaaa* es transformado en el string *bbbb*.

Notemos que si el cabezal comienza en cambio sobre el primer símbolo del string *aaba*, representado a la izquierda en la cinta del esquema, al llegar a la celda que contiene originalmente una b la máquina no tiene ninguna instrucción que pueda ejecutar (pues el estado interno es q_0). Esto es un comportamiento formalmente válido de la máquina, dado que la definición dada no implica que dado un estado interno q y un símbolo s bajo el cabezal haya siempre una instrucción que pueda ser ejecutada. De hecho, también es posible que haya más de una instrucción posible.

Si no hay instrucciones que puedan ser ejecutadas, interpretamos que la máquina "no sabe qué hacer" o "se cuelga", y la ejecución *se indefine* de modo equivalente a una ejecución consistente en una cadena infinita de acciones (y que por lo tanto nunca lleva al mecanismo de control al estado q_f).[11] Por otro lado, respecto a la existencia de múltiples instrucciones entre las que elegir, esto tan solo implica que hay múltiples ejecuciones posibles partiendo desde un mismo estado inicial del sistema control-cinta, y que por lo tanto es necesario considerar qué ocurre con cada una de ellas por separado (si la ejecución se detiene o no, y en tal caso con qué configuración final de la cinta). Esto motiva la siguiente distinción.

Definición 3.2.2 (Máquinas (no) determinísticas)**.** Una máquina de Turing con estados internos Q sobre un alfabeto Σ es *determinística* si dos instrucciones diferentes de su tabla de instrucciones T nunca comienzan con los mismos dos elementos q, s. Es decir, si T constituye efectivamente una función parcial $Q \times \widetilde{\Sigma} \to A \times Q$. En caso contrario, decimos que la máquina es *no determinística*.[12]

De ahora en adelante asumiremos que todas las máquinas de Turing son determinísticas, dado que permitir tablas de instrucciones no determinísticas no modificará la clase

[11] Esto es una cuestión de conveniencia que no tiene mayores consecuencias. Incluso aunque solo permitiéramos tablas de instrucciones que excluyen esta posibilidad, aún así serían posibles ejecuciones que no se detienen nunca, por lo cual no es posible evitar la existencia de situaciones en las que la máquina nunca arriba al estado final (que es, en definitiva, lo que importa en este contexto).

[12] En general, tanto en Computabilidad como en Complejidad Computacional y otras áreas de la Teoría de la Computación es importante no confundir un sistema no determinístico con un sistema estocástico o probabilístico. Podemos decir que un sistema no determinístico en este sentido es meramente *posibilístico*: un estado actual no determina unívocamente el estado en el siguiente instante de tiempo, pero eso no quiere decir que haya una distribución de probabilidad sobre los estados subsiguientes posibles.

de funciones efectivamente calculables[13].

3.2.2. Funciones parciales computables

Si bien no hemos dado una definición totalmente formal de la ejecución de una máquina de Turing, la descripción informal brindada es suficiente para, ahora sí, establecer el *significado* de una máquina de Turing, en términos de la función parcial que computa.

Es usual definir la noción de computabilidad para funciones parciales de números naturales. Es decir, se suele considerar la clase de funciones que pueden ser (o no) computables como la clase de funciones parciales $f : \mathbb{N}^n \to \mathbb{N}$ donde para cualquier $n \in \mathbb{N}$ (valor denominado la *aridad* de la función). En el presente trabajo nos mantendremos dentro de las funciones de strings en la medida de lo posible, de forma tal que para cada aridad n consideramos funciones n-arias $\Sigma^{*n} \to \Sigma^*$.

Frecuentemente, consideraremos strings y números naturales como equivalentes bajo la biyección $\mathbb{N} \simeq \Sigma^*$ dada por el orden lexicográfico sobre Σ^*.[14] Por ejemplo, si $\Sigma = \{0, 1\}$, tenemos que los números $0, 1, 2, 3$ y 4 corresponden bi-unívocamente a los strings $\epsilon, 0, 1, 00$ y 01. Siempre pensamos a Σ^* como un conjunto ordenado con el orden lexicográfico.

Definición 3.2.3 (Función computada por una máquina)**.** Sea $\mathcal{M} = (\Sigma, Q, T, q_0, q_f)$ una máquina de Turing y sea $n \in \mathbb{N}$. La *función parcial de aridad n computada por $\mathcal{M}$*, $f : \Sigma^{*n} \to \Sigma^*$, se define de la siguiente manera. Dados $x_1, ...x_n \in \Sigma^*$, sea $\varphi_0 \in \Sigma^{\mathbb{Z}}$ la configuración de la cinta que tiene a dichos strings escritos de izquierda a derecha comenzando en la celda de inicio de la máquina, separados por blancos (el string ϵ se representa, a su vez, por una celda en blanco). El resto de la cinta antes y después de los strings también se encuentra en blanco. Consideremos ahora la ejecución de $\mathcal{M}$ tomando a φ_0 como configuración inicial de la cinta. Si la ejecución nunca alcanza el estado q_f, entonces $f(x_1, ...x_n) \uparrow$, i.e. la función está indefinida para dichos argumentos. Si la ejecución alcanza el estado q_f, entonces examinamos la configuración final de la cinta. Si el cabezal se encuentra sobre una celda en blanco, entonces $f(x_1, ...x_n) \uparrow$. Si no, $f(x_1, ...x_n)$ se define como el string compuesto por la colección maximal de celdas contiguas no en blanco a la que pertenece la celda bajo el cabezal.

Finalmente podemos formalizar la idea de función efectivamente calculable como una

[13] No describiremos la manera en que una máquina de Turing no determinística define una función parcial computable. Esto puede encontrarse en cualquier libro de referencia tal como [19].

[14] Dado un orden total sobre Σ, (e.g. $\{0 < 1 < 2\}$ para $\Sigma = \{0, 1, 2\}$), el orden lexicográfico sobre Σ^* es aquel en el cual dados $x, y \in \Sigma^*$ distintos, si $|x| < |y|$ entonces $x < y$ y, si $|x| = |y|$, entonces $x < y$ sii $x_i < y_i$, donde $i \in \{1 \dots |x|\}$ es el índice del primer símbolo de izquierda a derecha en el que difieren los dos strings.

función que es computada por alguna máquina.

Definición 3.2.4 (Función parcial computable)**.** Una función parcial $f : \Sigma^{*n} \to \Sigma^*$ es *parcial computable* si existe una máquina de Turing $\mathcal{M}$ sobre el alfabeto Σ tal que f es la función parcial de aridad n computada por $\mathcal{M}$.

Notemos que cada máquina $\mathcal{M}$ computa infinitas funciones parciales distintas (una para cada $n \in \mathbb{N}$). A su vez, fijaremos durante el resto del capítulo un alfabeto Σ[15].

También podemos preguntarnos si es o no es computable una función *total*, es decir una función en el sentido matemático usual del término (está definida para todos los elementos de su dominio). Este es un caso particular de la definición dada arriba. Cuando decimos *función computable*, nos referimos a una función total computable. Por otro lado, a veces escribimos "función p.c." como abreviatura de "función parcial computable".

* * *

A continuación estudiamos dos aspectos fundamentales de las funciones p.c.: por un lado, la existencia de funciones p.c. universales, que en cierto sentido "incluyen" el comportamiento de todas las funciones p.c., y por otro lado el hecho de que existen funciones parciales que no son computables. En particular, existen funciones *totales* que no son computables. Si bien esta última afirmación es un caso particular de la anterior, puede argumentarse que es más sorprendente, dado que refuta una preconcepción que puede ser formulada en términos más elementales: no toda función matemática $\mathbb{N}^n \to \mathbb{N}$ admite un algoritmo que la compute[16].

3.2.3. Universalidad

La clase de funciones p.c. tiene la particularidad de admitir elementos *universales*. Esto la distingue de otras clases de funciones, como por ejemplo la de las funciones parciales (no necesariamente p.c.) o la de las funciones totales computables. Para definir funciones p.c. universales utilizaremos, como es de esperar, una noción análoga de universalidad para máquinas de Turing.

Que un elemento de una clase sea universal (con respecto a dicha clase) refiere a que dicho elemento puede simular el comportamiento observable de todos los elementos de la clase, siempre y cuando se le suministre una entrada adecuada. Esto implica que debe ser posible incluir la información que especifica un elemento de la clase dentro del argumento

[15] Dado que la cinta siempre dispone de al menos un símbolo de Σ y el símbolo adicional $\{\_\}$, no es difícil convencerse de que la elección de alfabeto no afecta la discusión de la teoría

[16] Recordar que apelamos libremente a la equivalencia $\Sigma^* \simeq \mathbb{N}$.

que recibe el elemento universal. En otras palabras, partimos de la idea de *recursividad*: los programas no son de una naturaleza diferente que las entradas que reciben, y por lo tanto un programa puede ser entrada para otro programa (o incluso para sí mismo, como veremos en la Sección 3.2.4).

En el contexto de un lenguaje de programación convencional, esto es intuitivo pues el código fuente de un programa está representado dentro del sistema operativo ambiente como un string, y podemos pasar strings como entrada a programas escritos en el mismo lenguaje. Sin embargo, para formalizar esto en el modelo de cómputo de máquinas de Turing, debemos establecer una manera de asociar cada máquina de Turing $\mathcal{M}$ a un string $E(\mathcal{M})$ que la describa[17]. Esta regla de asociación (o codificación) debe ser "efectiva" en el sentido siguiente: debe existir una máquina de Turing tal que pueda determinar si el string con el que comienza en su cinta es igual a $E(\mathcal{M})$ para algún $\mathcal{M}$, devolviendo un string de salida específico u otro según el caso (e.g. 1 si existe una tal $\mathcal{M}$ o 0 en caso contrario). Más aún, la codificación deberá permitirnos construir una máquina capaz de reproducir el comportamiento de cualquier otra máquina a partir de su codificación como string (que esto sea posible es, precisamente, afirmar la existencia de una máquina universal). A su vez, al codificar máquinas como strings obtenemos también una *enumeración efectiva* de todas las máquinas, puesto que podemos ordenarlas según el orden lexicográfico de sus strings correspondientes. Así obtenemos la sucesión $\mathcal{M}_1, \mathcal{M}_2, \ldots$ de máquinas de Turing para un alfabeto Σ fijo. Una codificación posible puede consultarse en [9].

La enumeración efectiva para máquinas de Turing da lugar a una enumeración efectiva $\phi_1^{(n)}, \phi_2^{(n)}, \ldots$ para las funciones p.c. computadas por ellas, para cada aridad $n \in \mathbb{N}$. Notemos que en general infinitos índices pueden corresponder a la misma función p.c. (es decir, infinitas máquinas diferentes computan la misma función). En este sentido, la enumeración efectiva de funciones p.c. es una enumeración con repetición, y la expresión $\phi_i^{(n)}$ debe leerse como "la función p.c. de n variables computada por la i-ésima máquina de Turing". Además, usando la equivalencia entre strings y números naturales, escribimos $\phi_x^{(n)}$, con $x \in \Sigma^*$, para referirnos a la i-ésima función p.c. donde i es el número natural correspondiente a x (i.e. x es el i-ésimo string de Σ^* en el orden lexicográfico).

Utilizamos la enumeración de máquinas de Turing (y por ende la de las funciones p.c. computadas por ellas) para definir las funciones universales. La idea es que uno de los argumentos que recibe la función universal sirve para indicar a qué máquina debe simular usando el resto de sus argumentos.

[17] Existen otros modelos de cómputo en los cuales los objetos que representan a los programas y los objetos que representan a sus entradas pertenecen exactamente a la misma clase de objetos. El ejemplo fundamental es el cálculo lambda o λ-cálculo, introducido originalmente por Church.

Definición 3.2.5 (Funciones p.c. universales)**.** Sea $\phi_1^{(n)}, \phi_2^{(n)}, \ldots$ la enumeración efectiva de referencia para funciones p.c. de n variables. Una función p.c. $\nu^{(n+1)} : \Sigma^{*n+1} \to \Sigma^*$ se dice *universal* para la clase de funciones p.c. de n variables sii para cada elección de $i, x_1, \ldots x_n \in \Sigma^*$, vale

$$\nu^{(n+1)}(i, x_1, \ldots x_n) = \phi_i^{(n)}(x_1, \ldots, x_n) \tag{3.5}$$

si $\phi_i^{(n)}(x_1, \ldots, x_n)$ está definido, y $\nu^{(n+1)}(i, x_1, \ldots x_n) \uparrow$ en caso contrario.

A las funciones p.c. universales las denominamos también *intérpretes universales*.

Veamos ahora la existencia de estas funciones. En verdad, existen infinitas máquinas de Turing universales que puedan computarlas. Más aún existe un interés en encontrar las máquinas universales más "sencillas", es decir con menor número de estados internos y de símbolos en su alfabeto, así como en encontrar universalidad en instancias simples de otros modelos de cómputo, tales como autómatas celulares [20]. Como no nos interesa controlar el número de estados internos o instrucciones de la máquina, podemos mantener fijado el alfabeto Σ y discutir informalmente la construcción de una máquina de Turing universal $\mathcal{U}$ particular que computa una función universal $\nu^{(n+1)}$ para cada n. Esta construcción es esencialmente independiente del tamaño del alfabeto.

La definición formal de $\mathcal{U}$ podría tomar páginas enteras, por lo que se da una descripción intuitiva del funcionamiento de la misma, sabiendo que es posible formalizarla de ser necesario. Esta informalidad también está habilitada por el hecho de que no nos interesa controlar el número de estados internos o instrucciones necesarias.

Siguiendo la Definición 3.2.3, asumimos que nuestra máquina recibe de entrada alguna cantidad m de argumentos $x_1, \ldots x_m \in \Sigma^*$ separados por blancos hacia la derecha de la posición inicial del cabezal. El string en la primera posición de entrada, o primera entrada de izquierda a derecha, será interpretado como el número natural i que le corresponde lexicográficamente. La máquina comienza su ejecución generando todos los strings de Σ^* en orden lexicográfico hacia la *izquierda* de la posición inicial, y chequeando para cada string si es el caso que codifique una máquina de Turing (según la codificación efectiva de referencia fija). Si efectivamente codifica una máquina, lee el primer string hacia la derecha de la posición inicial (el que originalmente es i) y lo sobrescribe por el string que le precede lexicográficamente (completando con blancos las celdas liberadas). De esta forma, el string a la izquierda de la posición inicial que hace que el string en la primera posición de entrada sea ϵ es la codificación de la i-ésima máquina de Turing. A continuación, la máquina ejecuta las instrucciones de dicha máquina, leyéndolas del string que quedó a la izquierda de la posición inicia, usando como entradas los $m-1$ strings

que quedan a la derecha de la posición inicial del cabezal. Al terminar de reproducir la ejecución de T_i, nos aseguramos de que el cabezal quede en la posición en la que habría quedado el cabezal de T_i. Por supuesto, si en algún momento la ejecución de T_i se indefine, será inmediato que la de $\mathcal{U}$ también, y del mismo modo la ejecución de $\mathcal{U}$ continuará indefinidamente si la de T_i lo hace.

Si nos convencemos de que esta construcción funciona, habremos demostrado el siguiente resultado.

Teorema 3.2.1. *Para cada $n \in \mathbb{N}$ existe una función p.c. $\nu^{(n+1)}$ que es universal para la clase de funciones p.c. de n variables.*

* * *

Este es un buen momento para preguntarse ¿Por qué venimos insistiendo en que las funciones computables pueden ser parciales? Hemos insistido en que la ejecución de una máquina de Turing puede indefinirse, pero no hemos dado un argumento concreto sobre por qué las máquinas que se indefinen deberían importarnos. ¿No podríamos, acaso, restringirnos a considerar las máquinas de Turing que *sí* se detienen para cualquier entrada?

La existencia de intérpretes universales nos mostrará que es imposible realizar esa restricción de manera efectiva, pues no existe una máquina de Turing que pueda distinguir a las máquinas que siempre se detienen. Esto puede demostrarse como corolario de la incomputabilidad del *halting problem* o "problema de la detención". En el camino, comenzaremos a entender los problemas conceptuales que trae la incomputabilidad de la información algorítmica.

3.2.4. Incomputabilidad

A continuación nos será útil hablar de funciones cuyo codominio es binario, indicando si se cumple o no una determinada propiedad del argumento. A este tipo de funciones las llamamos *predicados*, y a los dos valores que pueden adoptar los llamamos genéricamente 0 y 1.

Definición 3.2.6. Un *predicado* n-ario es una función $p : \Sigma^{*n} \to \{0, 1\}$, donde identificamos $\{0, 1\}$ con algún subconjunto distinguido de Σ^*.

A continuación enunciamos y demostramos la incomputabilidad del *halting problem*.

Teorema 3.2.2. *Sea* $g : \Sigma^{*2} \to \{0,1\}$ *el predicado total definido por* $g(x,y) = 1$ *si* $\phi_x^{(1)}(y)$ *está definido y* $g(x,y) = 0$ *en caso contrario (donde identificamos* x *con el número natural al que corresponde). Entonces* g *no es computable.*

Demostración. Para demostrar la negación de que g es computable, asumimos que sí lo es y derivamos una contradicción[18]. Supongamos que g es computable. Entonces existe una máquina de Turing $\mathcal{M}$ que lo computa, y que se detiene para cualquier entrada (pues g es total). Así, podemos construir una máquina de Turing $\mathcal{M}'$ que, usando a $\mathcal{M}$ como "subrutina", computa la siguiente función:

$$\psi : \Sigma^* \to \Sigma^*$$
$$\psi(x) = \begin{cases} 1 & \text{si } g(x,x) = 0 \\ \uparrow & \text{si } g(x,x) = 1 \end{cases} \tag{3.6}$$

Analicemos esto en palabras. Hemos construido un "programa rebelde" que, dado x, hace "lo opuesto" a lo que haría x al ser ejecutado pasándole su propio código fuente como argumento.

Ahora, ¿Qué pasa si a ese programa le damos su propio código fuente como argumento? Es decir, como ψ es p.c. existe un string y (su código fuente) tal que $\psi = \phi_y^{(1)}$, y podemos preguntarnos por el valor de $\phi_y^{(1)}(y)$.

Según la definición de ψ, $\phi_y^{(1)}(y)$ está definido sii $g(y,y) = 0$. Pero esto es absurdo, pues $g(y,y) = 0$ si y solo si $\phi_y^{(1)}(y)$ está indefinido. La contradicción proviene de haber asumido que g es computable. □

Vemos así que la existencia de un intérprete universal nos permite demostrar que ciertos problemas de decisión, como el de decidir si una máquina de Turing se indefine, no son *decidibles*, o lo que es lo mismo, sus funciones predicado correspondientes no son computables. De una manera análoga puede demostrarse que el predicado binario halt, definido por $\text{halt}(x,y)$ sii $\phi_x^{(1)}(y)$ está definido, tampoco es computable. La técnica general de demostración que empleamos se denomina *diagonalización* y es una herramienta fundamental para el estudio de la Computabilidad.

Es importante distinguir la imposibilidad de un algoritmo general para computar halt, del hecho de que sí es posible demostrar, para programas específicos, que van a detenerse

[18] En matemática constructiva es interesante distinguir entre este procedimiento y una demostración por el absurdo, en la cual se niega una afirmación, para luego encontrar una contradicción y concluir la negación de la negación. Esto no es constructivamente equivalente a la afirmación original. En cambio, la demostración de la incomputabilidad del *halting problem* no es una demostración por el absurdo y por lo tanto es constructiva.

(o no hacerlo) al ser ejecutados. Como ejemplo intuitivo, es obvio que un programa del estilo "`while True: skip`", que lo único que hace es ejecutar un bucle infinito sin hacer nada, no se detiene. Sin embargo, este tipo de comportamientos *sí* es fácil de detectar, dado que en este bucle el programa siempre vuelve a un mismo estado luego de una cierta cantidad de pasos[19]. Un programa que no se detiene no necesariamente tiene un comportamiento periódico, sino que puede estar realizando algo más sofisticado, como por ejemplo buscar un contraejemplo a alguna conjetura matemática famosa. Es decir que si uno pudiera determinar si el programa se va a detener o no sin ejecutarlo, tendría una manera automática de demostrar o refutar conjeturas matemáticas. Más aún, existen infinitos programas que no se detienen, para los cuales es imposible encontrar una demostración formal de que no lo hacen.

Como corolario de esta discusión, vemos que no es posible determinar de manera efectiva si la función computada por una máquina de Turing es total. Esto justifica nuestra elección de extender el universo de discurso a las funciones parciales desde un primer momento.

* * *

Si bien la discusión precedente puede resultar abstracta y técnica, la incomputabilidad del *halting problem* no solo es un resultado sorprendente desde el punto de vista conceptual sino que tiene consecuencias importantes en la práctica. Por ejemplo, en el contexto de verificación de software, en el cual se busca demostrar matemáticamente que determinado programa satisface una especificación de diseño, la incomputabilidad impone limitaciones muy fuertes a la manera en que este problema puede ser encarado. Esto se debe a que todas las propiedades *semánticas* de los programas (o máquinas de Turing), es decir todas aquellas que refieren únicamente a propiedades de la función parcial que computan, son o bien triviales, o bien indecidibles. Este es el contenido del Teorema de Rice [19].

Las consecuencias de la incomputabilidad del *halting problem* pueden verse en la experiencia cotidiana del uso de una computadora, cada vez que el sistema operativo informa al usuario que un programa no responde, y pregunta si desea cerrarlo. El sistema operativo recurre en algún sentido a la intuición o las necesidades pragmáticas del usuario debido a que no puede saber en general si un programa dado eventualmente va a responder o no.

[19] En términos de máquinas de Turing, se trataría de una máquina que luego de una cierta cantidad de pasos, vuelve al mismo estado interno en la misma posición y con la misma configuración de la cinta. En términos de sistemas dinámicos, se trata de una trayectoria periódica y para detectarla basta identificar la primera vez que ocurre el retorno a un mismo estado total previo, pues el sistema es determinístico.

Por último, más directamente relevante para el presente trabajo es el hecho de que la información algorítmica es, como veremos en la Sección 3.3.1, una función *incomputable* (i.e. no es computable).

3.2.5. Variantes de máquinas de Turing

Como se mencionó antes, existen muchas definiciones ligeramente diferentes de máquina de Turing. Puede haber múltiples cintas, múltiples cabezales, múltiples alfabetos diferentes (e.g. uno para entradas y otro para salidas), o distintas definiciones de qué constituye una acción básica (por ejemplo, permitiendo un movimiento del cabezal simultáneamente con la escritura de un símbolo). Incluso puede considerarse una "cinta" de dimensionalidad diferente (e.g. que la memoria esté estructurada como grilla bidimensional). Lo importante es que todas estas variantes pueden *simularse* entre sí[20], en el sentido de que para cada máquina de Turing de tipo A siempre puede construirse una máquina de tipo B tal que computa la misma función parcial que la primera, y viceversa. Por lo tanto, la clase de las funciones computables no depende de estos aspectos de la definición.

* * *

Habiendo recorrido esta breve introducción a la Computabilidad, estamos en condiciones de introducir de manera rigurosa el concepto de información algorítmica, formalizando las intuiciones presentadas anteriormente.

3.3. Información algorítmica

Retomando la discusión en 3.1, para definir la información algorítmica necesitaremos especificar un método de descripción, y una manera en la que los objetos descriptos pueden ser recuperados a partir de sus descripciones.

En principio, cualquier función parcial computable ϕ de una variable constituye un método válido: dado $x \in \Sigma^*$, decimos que x es una *ϕ-descripción de y* si $\phi(x) = y$. El hecho de que ϕ sea p.c. nos indica que el objeto puede ser recuperado a partir de su descripción de modo efectivo.

[20] De hecho, muchas de ellas pueden hacerlo de manera *eficiente* en términos de recursos computacionales, si bien eso excede las preocupaciones de la Computabilidad para adentrarse en la Complejidad Computacional.

Así, para cada objeto en un conjunto numerable S al que le asignamos un string en Σ^*, o equivalentemente, para cada string $x \in \Sigma^*$, podemos definir su *complejidad de descripción* con respecto al método de descripción ϕ según

$$\begin{aligned} C_\phi &: \Sigma^* \to \mathbb{N} \cup \{\infty\} \\ C_\phi(x) &= \min_{p \in \Sigma^*} \{|p| : \phi(p) = x\} \end{aligned} \tag{3.7}$$

donde interpretamos que $C_\phi(x) = \infty$ si no existe una descripción p para x.

Si ahora elegimos una función p.c. universal como nuestro método de descripción, tendremos garantizada la existencia de al menos una descripción para cada string (e.g. `print(`x`)`, donde x debe ser reemplazado por su valor como string) y así la complejidad de descripción será siempre finita. Notemos que en este caso estaríamos usando una función p.c. universal $\nu^{(1)}$ tal que $\nu^{(1)}(x)$ corresponde al comportamiento de la máquina x-ésima[21] al ser ejecutada *sin ninguna entrada* en la cinta.

En contraposición, si pudiéramos suministrarle una entrada y a la máquina como ayuda, la complejidad de descripción de x podría resultar ser menor. En vez de considerar funciones p.c. de 2 variables o más, resultará más conveniente utilizar una función de formación de pares $\langle -, - \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ biyectiva y computable. De esta manera podemos utilizar funciones p.c. únicamente de una variable. De ahora en adelante, asumimos que las funciones p.c. son de una sola variable a menos que se diga lo contrario.

Definición 3.3.1 (Complejidad de descripción condicional)**.** Sean $x, y, p \in \Sigma^*$. Cualquier función p.c. ϕ junto con p e y, tales que $\phi(\langle y, p \rangle) = x$, es una *descripción* de x. La *complejidad* C_ϕ de x *condicional* a y se define como

$$\begin{aligned} C_\phi &: \Sigma^* \times \Sigma^* \to \mathbb{N} \cup \{\infty\} \\ C_\phi(x|y) &= \min_{p \in \Sigma^*} \{|p| : \phi(\langle p, y \rangle) = x\} \end{aligned} \tag{3.8}$$

y $C_\phi(x|y) = \infty$ si no hay un p tal. Decimos que p es un *programa* para computar x según ϕ, dado y (o a partir de y). Equivalentemente, decimos que p es una *ϕ-descripción de x dado y*.

La clase de funciones p.c. nos ofrece un tipo de universalidad más específico, que es el de las funciones aditivamente óptimas. Esta propiedad es un poco técnica, en el sentido de que las funciones universales más intuitivas, que son las instanciadas por los

[21] Realizamos un abuso de lenguaje identificando x con el número que representa. Más correctamente debería hablarse de la máquina $\mathcal{M}$ tal que su codificación simbólica es $E(\mathcal{M}) = x$.

lenguajes de programación convencionales, corresponden a funciones universales aditivamente óptimas, e inversamente resulta extraño imaginar un lenguaje de programación que *no* corresponda a una función aditivamente óptima. Sin embargo, es la propiedad matemática que necesitamos para capturar este aspecto de los lenguajes. Intuitivamente, se relaciona con el hecho de que es posible escribir un programa "compilador" o "intérprete" para un lenguaje $\mathcal{L}$ escrito en otro lenguaje $\mathcal{L}'$, de forma tal que código fuente escrito para el lenguaje $\mathcal{L}$ pueda ser ejecutado en el lenguaje $\mathcal{L}'$, al dárselo como entrada a dicho intérprete. La observación clave es que esto permite convertir cualquier descripción en el lenguaje $\mathcal{L}$ en una descripción en el lenguaje $\mathcal{L}'$, *pagando tan solo un incremento fijo en la longitud de la descripción* (dado por la longitud del intérprete).

Definición 3.3.2 (Función aditivamente óptima)**.** Sea $\mathcal{C}$ una subclase de funciones parciales $\Sigma^* \to \Sigma^*$ (no necesariamente p.c.). Una función $f \in \mathcal{C}$ se dice *aditivamente óptima* para la clase $\mathcal{C}$ sii para cada función $g \in \mathcal{C}$ existe una constante $c_{f,g} \in \mathbb{N}$ tal que $C_f(x|y) \leq C_g(x|y) + c_{f,g}$ para todo x y para todo y.

Adicionalmente, decimos que una función f cualquiera *minoriza* a g si existe una constante $c_{f,g}$ como arriba para ese par de funciones particular. Es decir que f es aditivamente óptima si minoriza a todas las funciones de la clase.

Un "corolario de la definición" es el siguiente.

Proposición 3.3.1 (Invariancia)**.** *Sean f, g funciones parciales aditivamente óptimas para una clase $\mathcal{C}$ como arriba. Entonces existe una constante $c_{f,g}$ tal que*

$$|C_f(x|y) - C_g(x|y)| \leq c_{f,g} \tag{3.9}$$

para todo x y para todo y.

Ahora, veamos que efectivamente la clase $\mathcal{C}$ de funciones parciales computables admite este tipo de universalidad.

Teorema 3.3.1. *Existe una función parcial computable universal que es aditivamente óptima para la clase de las funciones parciales computables.*

Demostración. Mediante una construcción similar a la de la máquina universal $\mathcal{U}$, podemos definir una máquina universal $\mathcal{U}'$ tal que, al ser inicializada con entrada $\langle y, \langle n, p\rangle\rangle$, simula a la máquina T_n con entrada $\langle y, p\rangle$. Sea ψ la función p.c. computada por $\mathcal{U}'$. Entonces esto es decir que $\psi(\langle y, \langle n, p\rangle\rangle) = \phi_n(\langle y, p\rangle)$.

Consideremos una función p.c. ϕ_n arbitraria. Sean $x, y \in \Sigma^*$ y sea p un programa para x según ϕ_n dado y, es decir $\phi_n(\langle y, p\rangle) = x$. Queremos obtener a partir de esta

información un programa para computar x a partir de y según ψ, cuya longitud no pueda ser arbitrariamente más larga que la de p. En efecto, el párrafo precedente nos muestra que $\langle n, p\rangle$ es un programa tal, y tenemos que $|\langle n, p\rangle| \leq |p| + c_{\phi_n}$ donde c_{ϕ_n} es una constante que depende solo de n.

Así, vemos que para $x, y \in \Sigma^*$ cualesquiera,

$$C_\psi(x|y) \leq C_{\phi_n}(x|y) + c_{\phi_n}. \tag{3.10}$$

Para la función de formación de pares dada por la concatenación autodelimitante (ver Sección 1.4), podemos en efecto tomar $c_{\phi_n} = 2|n| + 1$. □

Ahora fijamos una función pc universal y aditivamente óptima, y definimos la información algorítmica como la complejidad relativa a ese método de descripción. Definimos la información algorítmica no condicional como caso particular de la condicional.

Definición 3.3.3 (Información algorítmica)**.** Sea ψ una función p.c. universal aditivamente óptima fija, que llamaremos *función de referencia*. Definimos entonces la *información algorítmica* (o *complejidad de Kolmogorov*) *condicional* según

$$\begin{aligned} C &: \Sigma^* \times \Sigma^* \to \mathbb{N} \\ C(x|y) &= C_\psi(x|y). \end{aligned} \tag{3.11}$$

La *información algorítmica* (o *complejidad de Kolmogorov no condicional* se define como

$$\begin{aligned} C &: \Sigma^* \to \mathbb{N} \\ C(x) &= C(x|\epsilon). \end{aligned} \tag{3.12}$$

¿Por qué es tan importante haber elegido una función p.c. universal y aditivamente óptima? Precisamente porque entonces tenemos el resultado de la Proposición 3.3.1. Este resultado, denominado *Teorema de invariancia* cuando se aplica a la información algorítmica, nos indica que en cierto sentido la información algorítmica es una medida "absoluta" o "universal" de la cantidad de estructura presente en un objeto cualquiera, y que por lo tanto es una propiedad intrínseca *del objeto*, más que una propiedad relacional entre el objeto y el método de descripción.

Sin embargo, cabe enfatizar que esta propiedad intrínseca de los objetos no es estrictamente hablando un *número*, puesto que el valor concreto de $C(x)$ depende de la función de referencia. De hecho, cambiar de función de referencia puede modificar la relación de orden entre las informaciones de dos strings distintos, $C(x)$ y $C(y)$. Más aún las cons-

tantes de "traducción" $c_{f,g}$ en la Proposición 3.3.1 pueden ser arbitrariamente grandes. En términos de lenguajes de programación, es intuitivo que sea posible escribir, en un lenguaje universal cualquiera, un compilador o intérprete para otro lenguaje universal, pero simultáneamente es posible imaginar lenguajes de programación universales pero fuertemente optimizados para representar sucintamente determinado tipo de programas, que trabajen con determinado tipo de entradas.

Como ejemplo de aplicación del hecho de que usamos un intérprete universal aditivamente óptimo (que es la propiedad matemática detrás del teorema de invariancia), formalicemos la idea presentada anteriormente de que los programas de la forma `print(x)` dan una cota para la información algorítmica de cualquier string.

Proposición 3.3.2. *Existe una constante $c \in \mathbb{N}$ tal que $C(x) \leq |x| + c$ para todo $x \in \Sigma^*$.*

Demostración. Consideremos la función identidad id_{Σ^*}, dada por $\mathrm{id}_{\Sigma^*}(x) = x$. Esta función es claramente computable, pues una máquina de Turing que la computa es aquella que no hace más que detenerse apenas comienza su ejecución. Por otro lado, considerada como método de descripción, tenemos que la longitud de descripción de cualquier string según id_{Σ^*} es precisamente la longitud del string original.

Por lo tanto, como el intérprete de referencia para C minoriza a id_{Σ^*}, existe una constante c tal que $C(x) \leq |x| + c$, que es lo que se quería demostrar. □

* * *

Si bien el teorema de invariancia es una piedra angular para la Teoría Algorítmica de la Información, dependiendo del contexto puede resultar más adecuado enfatizar este grado de independencia de la información algorítmica respecto del lenguaje, máquina o intérprete de referencia empleado, o bien por el contrario enfatizar la manera en que esta *sí* depende de dicha elección. Una perspectiva útil en este sentido es la de pensar a la información algorítmica no como una medida absoluta sino como una manera de comparar la estructura de distintos objetos. El caso idóneo sería aquel en que se analizan familias infinitas de objetos, dado que entonces las constantes aditivas efectivamente se vuelven despreciables. Es interesante notar que la dificultad conceptual de aplicar la información algorítmica al análisis de objetos específicos y finitos es diferente y en cierto sentido independiente de las que surgen de la incomputabilidad de la información algorítmica, asunto que trataremos en la próxima sección.

3.3.1. Incomputabilidad de la informacion algorítmica

Como adelantamos previamente, la información algorítmica es incomputable. Para entender esto, consideremos la *paradoja de Berry*, que plantea la siguiente pregunta: ¿Cuál es el primer número natural que no puede ser definido en menos de, por ejemplo, 20 palabras? Si asumimos que cada expresión lingüística con menos de 20 palabras define a lo sumo un único número (lo cual parece ser necesario para cualquier acepción de la palabra "definir"), entonces como la cantidad de estas expresiones es finita, se sigue que debe haber algún número que no puede ser definido en menos de 20 palabras. Por lo tanto, debería haber un primero.[22]

Ahora bien, si existe ese primer número, llamémoslo m, entonces podemos definir m como "el primer natural que no puede ser definido con menos de veinte palabras". Lo cual a todas luces parece una descripción de m que toma menos de veinte palabras. ¿Qué está ocurriendo?

La paradoja surge de la ambigüedad en afirmar que un número "pueda ser definido": es necesario especificar un lenguaje que esté siendo empleado para definir, y qué constituye una definición dentro de ese lenguaje. Más aún, cuando formalizamos esta situación en términos de información algorítmica, lo que obtenemos es no una paradoja sino un resultado de incomputabilidad.

La formalización consiste en asumir que "x puede ser definido en menos de n símbolos" corresponde a la afirmación "$C(x) < n$". Así, "el primer string que no puede ser definido en menos de n símbolos" sería el primer elemento del conjunto $S_n = \{x \in \Sigma^* : C(x) \geq n\}$ (con el orden inducido por el orden en Σ^*). Llamemos s_n a ese elemento problemático.

El conjunto S_n está bien definido; no hay nada de "malo" con él. El problema viene de que este conjunto en general no es computable (si bien puede serlo para una cantidad finita de valores de n). En otras palabras, el predicado $\Sigma^* \to \{0, 1\}$ que vale 1 para strings que pertenecen S_n y 0 para strings que no, no es computable. Y si nuestra noción de "definición de un número" es un *programa* que lo genera, si no existe un programa para computar S_n, no podemos usar la pertenencia a S_n para definir a s_n realmente.

Veamos ahora una demostración sencilla de la incomputabilidad de C usando estas ideas, que puede ser encontrada en [21].

Teorema 3.3.2. *La información algorítmica vista como función total $C : \Sigma^* \to \Sigma^*$ es incomputable.*

Demostración. Para demostrar la negación de que C es computable, asumimos que sí lo es y derivamos una contradicción. Supongamos pues que C es computable. Entonces

[22] Esto es una propiedad de los números naturales, la cual se denomina "buen orden".

existe una máquina de Turing $\mathcal{M}$ que la computa. Usándola podemos construir una nueva máquina $\mathcal{M}'$, que espera como entrada un string n que interpretamos como número natural. Esta máquina itera sobre todos los strings de longitud n en orden lexicográfico, y para cada uno computa su información algorítmica (simulando a $\mathcal{M}$), hasta que encuentra un x con $C(x) \geq |x| = n$ y lo devuelve como salida.

Esta definición está bien planteada pues siempre existe un x de longitud n con complejidad mayor o igual a n. Para verlo, basta un argumento de conteo: existen $\sum_{i=0}^{n-1} |\Sigma|^i = 2^n - 1$ strings de longitud menor a n, mientras que el número de strings de longitud n es $|\Sigma|^n$.

Sea ϕ la función computada por $\mathcal{M}'$. Por la construcción de $\mathcal{M}'$, es una función total computable. Luego, como el intérprete universal de referencia minoriza a ϕ, existe una constante c tal que para todo x vale

$$C(x) \leq C_\phi + c. \tag{3.13}$$

Por cómo definimos ϕ, no todos los strings pueden ser obtenidos como salida, luego $C_\phi(x)$ será infinito para muchos x. Sin embargo, concentrémonos en la sucesión de strings $(x_n)_{n\in\mathbb{N}}$ dada por $x_n = \phi(n)$. Para estos strings, vale que su complejidad según ϕ es la longitud del string que codifica el número natural n. Esa longitud es aproximadamente $\log n$ (donde como siempre, el logaritmo es en base $|\Sigma|$). De esta forma, vemos que

$$n = |x_n| \leq C(x) \log n + c \tag{3.14}$$

lo cual es falso eligiendo n suficientemente grande. □

Como vemos, la demostración está relacionada con la velocidad a la que crecen distintas magnitudes. Así también se demuestra un resultado más fuerte cuya demostración puede consultarse en [9]: ninguna función p.c. definida en una cantidad infinita de strings puede coincidir con C sobre todo su dominio de definición. Esto nos dice que cualquier programa que busque aproximar C (computando alguna función parcial) erra respecto del valor correcto en todos salvo finitos de los valores que devuelve.

De esta manera, la información algorítmica no solo es incomputable sino que es "inaproximable" en ciertos sentidos específicos. En particular, podemos obtener funciones computables, quizá cada vez más sofisticadas, que coincidan con C en dominios cada vez más grandes, pero siempre existirán infinitos strings para los que el valor computado es incorrecto, y no podemos saber de cuáles strings se trata.

Como comentario, el método de conteo sencillo que usamos en la demostración pre-

cedente para mostrar que siempre existe algún string de longitud n cuya información es mayor o igual a n, permite también mostrar que, en general, la mayoría de los strings tiene complejidad aproximadamente igual a su longitud. Es decir, la mayoría de los strings son *incompresibles*. Por distintas razones, tiene sentido llamar también *aleatorios* a los strings incompresibles. Si bien no entramos en detalle en este aspecto de la teoría, esto es intuitivo si se considera que el hecho de no poder comprimir un string indica que este no tiene ninguna estructura que pueda ser aprovechada con ese fin.

3.3.2. Información mutua

Dado que nuestro objetivo es estudiar la correlación entre objetos simbólicos, el concepto fundamental que necesitamos de esta teoría de la información es el de *información mutua*. Así como la información mutua clásica mide la totalidad de la correlación entre dos variables aleatorias (a diferencia de magnitudes como el famoso coeficiente de correlación de Pearson, que solo cuantifica la correlación *lineal*), la información mutua algorítmica también cuantifica la correlación entre strings en un sentido total –al menos en lo relativo a estructura que pueda ser descrita de manera computable.

La información algorítmica posee muchas propiedades análogas a las de la información de Shannon [22]. Es por esto que puede construirse una Teoría Algorítmica de la Información que guarda una relación estrecha con la Teoría de la Información clásica. Sin embargo, para maximizar el paralelismo entre estas dos teorías resulta conveniente adoptar una variante diferente de la información algorítmica, que denotaremos con la letra K para distinguirla de C y que llamaremos *información algorítmica autodelimitante*. Esta presenta mejores propiedades cuantitativas que C. Por ejemplo, es subaditiva (a menos de la "típica" constante aditiva): $K(\langle x, y\rangle) \leq K(x) + K(y) + O(1)$. En cambio, para C, que en este contexto suele denominarse información algorítmica *plana*, esta propiedad vale tan solo a menos de un término logarítmico en las complejidades de los strings: $C(\langle x, y\rangle) \leq C(x) + C(y) + O(\log(\text{mín}\{C(x), C(y)\}))$.

Como antes, consideramos funciones p.c. de una sola variable, e interpretamos análogamente a las máquinas de Turing como sistemas que reciben un único string como entrada. La modificación consiste en considerar solo máquinas de Turing tales que el conjunto de strings que son válidos como entrada forma un *código libre de prefijos* o *autodelimitante*. Esto quiere decir que si p y q son dos entradas válidas distintas para una de estas máquinas de Turing, entonces ninguna es un prefijo de la otra.

Definición 3.3.4. Una *función parcial computable autodelimitante* $\phi : \Sigma^* \to \Sigma^*$ es una función parcial computable tal que si tanto $\phi(p)$ como $\phi(q)$ están definidos, para strings

$p \neq q$, entonces p no es un prefijo de q (ni viceversa).

Existe una construcción sencilla mediante la cual podemos reemplazar cada máquina de Turing $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3 \ldots$ en la enumeración efectiva por una máquina $\mathcal{M}'_1, \mathcal{M}'_2, \mathcal{M}'_3 \ldots$ que computa una función p.c. autodelimitante. Esta construcción puede encontrarse en [9]. La clase de funciones p.c. autodelimitantes también admite una función universal aditivamente óptima. Así, la información algorítmica autodelimitante es simplemente la complejidad definida con respecto a alguna función universal aditivamente óptima, intepretada como método de descripción.

Definición 3.3.5 (Información algorítmica autodelimitante)**.** Sea ψ_K una función p.c. autodelimitante, universal y aditivamente óptima (para la clase de funciones p.c. autodelimitantes) fija. Definimos entonces la *información algorítmica autodelimitante condicional* según

$$\begin{aligned} K &: \Sigma^* \times \Sigma^* \to \mathbb{N} \\ K(x|y) &= C_{\psi_K}(x|y). \end{aligned} \tag{3.15}$$

La *información algorítmica no condicional* se define como

$$\begin{aligned} K &: \Sigma^* \to \mathbb{N} \\ K(x) &= K(x|\epsilon). \end{aligned} \tag{3.16}$$

Ahora podemos definir la información mutua algorítmica utilizando C o K según se desee. En principio utilizamos K.

Definición 3.3.6 (Información mutua algorítmica)**.** La *información algorítmica sobre y* contenida en x se define según

$$I_K(x : y) = K(y) - K(y|x) \tag{3.17}$$

Una consecuencia inmediata de la definición es que, como $K(x|x) = O(1)$, se tiene que $I_K(x : x) = K(x) + O(1)$[23]. Notemos también que I es "no negativa": como $K(y) \geq K(x|y) + O(1)$, se sigue que $I_K(x : y) \geq 0$ a menos de una constante aditiva.[24]

La información mutua en la teoría clásica de la información es una cantidad simétrica. Esto tiene sentido dado que la idea intuitiva de "estructura común entre dos objetos" está desprovista de direccionalidad. Sería deseable entonces que I_K fuera una función

[23] Podemos suprimir los términos $O(1)$ si elegimos un intérprete de referencia ψ_K tal que $\psi_K(x, \epsilon) = x$

[24] Condicionar sobre x no puede aumentar la dificultad en describir y; a lo sumo, la "ayuda" constituida por x será ignorada.

simétrica de sus dos variables, aunque sea a menos de una constante aditiva. Sin embargo, la simetría se cumple a menos de un término *logarítmico*, y este problema no desaparece al emplear K en vez de C. Si se desea tener lo que en [9] se denomina *simetría exacta* de la información algorítmica, es necesario introducir una nueva variante de la información algorítmica, que denotamos por Kc. Para definirla, primero introducimos la notación x^* para referirnos a un programa minimal canónico para x (damos la definición con respecto a la información algorítmica autodelimitante).

Definición 3.3.7. Dado un string x denotamos por x^* a un programa minimal para x (es decir que $K(x) = |x^*|$), que es el "primero" en el sentido siguiente. Fijamos un programa de referencia tal que, a partir del string de entrada $\langle x, K(x)\rangle$, enumera[25] todos los programas para x de longitud $K(x)$. Definimos entonces a x^* como el primero en dicha enumeración.

La definición de x^* garantiza que la información contenida en x^* sea la misma que la contenida en el par $\langle x, K(x)\rangle$, i.e. un string puede obtenerse del otro a través de un programa fijo y viceversa.

Definición 3.3.8. Llamamos *complejidad condicional* Kc a la función $Kc : \Sigma^* \times \Sigma^* \to \mathbb{N}$ definida por $Kc(x|y) := K(x|y^*)$. La complejidad Kc no condicional coincide con K: $Kc(x) = K(x) \forall x$. Análogamente $Kc(x|y, z)$ se define como $K(x|\langle y, z\rangle^*)$, etc.

Usando esta definición, la información mutua $I_{Kc}(x : y) = Kc(y) - Kc(y|x) = K(y) - K(y|x^*)$ sí resulta simétrica a menos de una constante aditiva. Sin embargo, el uso de Kc en vez de K trae ciertas desventajas. Una de ellas es que, a diferencia de K, la complejidad Kc no es semicomputable por arriba[26]. Otra desventaja ligada a la anterior es que la expresión $K(x|y^*)$ es mucho menos intuitiva que $K(x|y)$, siendo más difícil establecer conceptualmente un vínculo entre la complejidad Kc y las aproximaciones computables a la información algorítmica.

* * *

Finalmente, damos las definiciones de información mutua *condicional*, tanto para el caso de K como el de Kc, las cuales son fundamentales en el contexto de este trabajo.

[25] La manera estándar de construir un programa para este fin es a través de la técnica de *dovetailing*, que consiste en ejecutar intercaladamente pasos correspondientes a la ejecución de distintos programas. Esto impide que la ejecución del programa quede bloqueada por un único programa que no se detiene, no pudiendo comenzar entonces la ejecución de los programas subsiguientes.

[26] Por cuestiones de espacio, omitimos discutir este aspecto en el presente trabajo, pero el mismo puede ser consultado en [9]

Definición 3.3.9 (Información mutua condicional)**.** Dados strings x, y, z, definimos la *información algorítmica (autodelimitante) sobre y contenida en x dado z* según

$$I_K(x : y|z) = K(y|z) - K(y|x, z) \tag{3.18}$$

Para el caso de Kc, la definición análoga es

$$I_{Kc}(x : y|z) = K(x|z) - K(x|y, K(y|z), z) \tag{3.19}$$

Si bien la definición de I_{Kc} puede no ser intuitiva, esta garantiza el siguiente resultado:

Proposición 3.3.3 (Simetría exacta de la información mutua algorítmica)**.** *La información mutua condicional algorítmica basada en la variante de complejidad Kc satisface*

$$I_{Kc}(x : y|z) = K(x|z) + K(y|z) - K(x, y|z) + O(1)$$

para todo x, y, y z.

3.4. Resumen

En este Capítulo presentamos el concepto de información algorítmica y la medida de correlación algorítmica universal asociada a ella, la información mutua algorítmica, en sus versiones marginal y condicional. Comenzamos con una exposición informal de la información algorítmica, pero para poder precisar este concepto debimos antes recorrer una breve introducción a la teoría de la Computabilidad. Así, vimos cómo formalizar el concepto de función efectivamente calculable, y la incomputabilidad del *halting problem* nos mostró por qué es necesario "convivir" con el hecho de que las funciones computadas por programas o máquinas son, genéricamente, funciones parciales. En el camino, vislumbramos el fenómeno de *universalidad computacional*, que puede expresarse como el hecho de que existen máquinas de Turing que pueden simular la ejecución de cualquier otra máquina, e incluso de sí mismas.

Habiendo hecho esto, pasamos a definir formalmente la noción de complejidad de descripción, basándonos siempre en que las descripciones deben ser efectivas. También usando la noción formal de computabilidad pudimos entender qué quiere decir que la información algorítmica C, que es la complejidad de descripción respecto de un intérprete universal aditivamente óptimo, sea incomputable, y por qué eso también representa una imposibilidad de *aproximarla* controladamente. Finalmente, introdujimos las variantes

K y Kc de la información algorítmica, las cuales presentan algunas propiedades cuantitativas diferentes, y con ellas definimos la información mutua algorítmica, concepto que jugará un rol importante en el capítulo siguiente.

Capítulo 4

Descubrimiento causal sintáctico

En el Capítulo 2 vimos que a partir de una distribución de probabilidad es posible estimar una clase de DAGs correspondiente a las estructuras causales minimales Markov-compatibles con los datos. Cuando no hay variables latentes, esta tarea de descubrimiento causal puede ser realizada mediante el algoritmo presentado en la Sección 2.4.1. Por otro lado, en el Capítulo 3 vimos cómo podemos poner sobre bases firmes la idea de estructura y correlación sintáctica, lo cual nos permite concebir una tarea de descubrimiento causal que opere sobre objetos con una estructura sintáctica que vaya más allá del paradigma de muestreo de variables aleatorias independientes e idénticamente distribuidas.

En este capítulo se presenta el trabajo original realizado para esta tesis. Este consiste en una implementación de la primera fase del algoritmo recién mencionado, adaptada para recibir como entrada no una distribución de probabilidad empírica sino una colección de strings, y devuelve un grafo no dirigido en el cual los nodos corresponden a los strings de entrada y las aristas corresponden a relaciones de causalidad directa inferidas entre los mismos. A su vez, se implementaron tests de independencia condicional de tipo sintáctico y otro, más convencional, de tipo estadístico, y se realizaron simulaciones cuyos resultados permiten comparar los dos tipos de tests. Para esto, se eligieron ciertas clases de modelos causales funcionales como modelos generadores de casos de prueba, los cuales admiten análisis mediante las dos variantes del mismo. Así, los resultados principales comparan estos dos métodos de análisis sobre distintas clases de modelos generadores.

Comenzamos presentando algunos antecedentes de tipo teórico, los cuales sugieren que esta adaptación del algoritmo a un contexto sintáctico puede ser fructífera.

4.1. Antecedentes

En [7], Janzing y Schölkopf proponen reescribir la teoría de modelos causales presentada en el Capítulo 2 reemplazando variables aleatorias por strings y la independencia condicional probabilística entre ellas por un criterio de información mutua algorítmica, concepto que introdujimos en la Sección 3.3.2. Una primera forma de entender el pasaje del primer escenario al segundo consiste en notar que la información mutua probabilística entre dos variables aleatorias es nula si y solo si dichas variables son probabilísticamente independientes. De esta manera, y aprovechando los paralelismos entre la Teoría de la Información clásica o de Shannon y su versión algorítmica[1], un candidato natural a definición algorítmica de independencia consiste en afirmar que dos strings, x e y, son independientes dado un tercer string z si y solo si $I(x : y|z) \equiv 0$. Típicamente, la teoría de la información algorítmica nos provee definiciones y resultados involucrando constantes aditivas desconocidas, con lo cual una definición posible de independencia sería pedir $I(x : y|z) = O(1)$. Sin embargo, los autores notan que una afirmación asintótica como esta solo podría ser interpretada en términos de sucesiones de strings $(x_n)_{n\in\mathbb{N}}, (y_n)_{n\in\mathbb{N}}, (z_n)_{n\in\mathbb{N}}$ de longitud creciente; en cambio, si se desea hablar de tres strings fijos y específicos que representan observaciones sobre el mundo, es necesario reemplazar el lenguaje asintótico por algún umbral concreto por debajo del cual se considera que los strings son condicionalmente independientes, y este valor de umbral puede en principio depender fuertemente del contexto de aplicación[2].

Los autores utilizan la variante Kc de la información algorítmica, la cual como se mencionó previamente difiere de la complejidad autodelimitante K en que $Kc(x|y) = Kc(x|y^*)$ y garantiza que la información mutua algorítmica sea simétrica a menos de una constante aditiva. Utilizando la información mutua condicional $I(- : -|-)$ correspondiente a esta variante, definen la relación de independencia condicional algorítmica para tres colecciones de strings X, Y, Z (posiblemente con elementos repetidos) como "$(X \perp\!\!\!\perp Y|Z)$ si y solo si $I(x : y|z) < t$" donde t es algún valor de umbral sin especificar y

[1] Una presentación en paralelo de ambas teorías puede encontrarse en [22].

[2] Si se fija un intérprete de referencia concreto, el orden de magnitud de dicho umbral puede ser estimado, teniendo en cuenta cuántos símbolos adicionales son necesarios para "ensamblar" los programas minimales para x e y en un único programa que produzca la tupla deseada. Sin embargo, los autores argumentan que para poder hacer sentido de los valores de información mutua en un determinado contexto de aplicación, es necesario introducir cierta información de fondo, codificada en un string w, interpretando la expresión $K(x|y)$ como $K(x|y, w)$. Por ejemplo, dados los genomas de dos personas s_1 y s_2, claramente se tendrá que $I(s_1 : s_2) \gg 0$ puesto que las dos personas son humanas, pero si lo que se desea es determinar posibles similitudes más allá de este hecho lo que debe ser evaluado es $I(s_1 : s_2|w)$ donde w es tal que la información algorítmica $K(s|w)$ es en general baja para genomas humanos s. El valor de umbral para determinar si $I(s_1 : s_2|w)$ es suficientemente bajo como para señalar independencia sería distinto dependiendo de w.

x, y, z son las concatenaciones de los strings en cada conjunto respectivamente, en algún orden prefijado para las variables[3]. Cuando nos referimos a esta noción de independencia algorítmica específica, utilizamos la notación $(X \perp\!\!\!\perp Y|Z)_{Kc,t}$.

Habiendo dicho esto, parafraseamos el Postulado 5 de [7] en términos de una definición de compatibilidad de Markov algorítmica, que es paralela a la versión local de la condición de Markov dada para el contexto probabilístico en el Teorema 2.2.2.

Definición 4.1.1 (Compatibilidad de Markov algorítmica)**.** Sean $x_1, \dots x_n$ strings que representan observaciones (no necesariamente de la misma longitud). Sea G un DAG cuyos vértices son dichos strings. Sea pa_j la colección de los padres de x_j según G y nd_j la colección de todos sus no-descendientes (excluyendo a x_j) $(j = 1 \dots n)$. Entonces decimos que G y el conjunto de strings $x_1, \dots x_n$ son *Markov-compatibles* si y solo si $(x_j \perp\!\!\!\perp nd_j|pa_j)_{Kc,t}$ para todo j. Más explícitamente, para cada j se tiene que

$$I(x_j : nd_j|pa_j) < t \tag{4.1}$$

donde identificamos a nd_j y pa_j con la concatenación de sus elementos (en algún orden prefijado) y t es una constante de umbral especificada por el contexto.

Los autores definen también una noción de *modelo causal algorítmico*, y demuestran que la colección de strings generada por un modelo cuyo DAG subyacente es G es Markov-compatible (en el sentido algorítmico) con G (resultado análogo al Teorema 2.3.1). Damos una definición de modelo causal algorítmico parafraseando el Postulado 6 de [7].

Definición 4.1.2 (Modelo causal algorítmico)**.** Consideremos una colección de strings $x_1, \dots x_n$ indexados[4]por números naturales $1, \dots n$. Un *modelo causal algorítmico* para dichos strings consiste en lo siguiente:

1. Una colección de de strings de ruido $(u_j)_{j=1\dots n}$ conjuntamente algorítmicamente independientes, i.e. $(u_i \perp\!\!\!\perp \{u_j : j \neq i\})_{Kc,t}$ para todo i.

2. Una *estructura causal subyacente*, i.e. un DAG G que especifica para cada i una tupla de padres pa_i.

3. n strings $q_1, \dots q_n$ que interpretamos como programas para máquina de Turing

[3] Distintos órdenes en la concatenación de los strings pueden dar lugar a valores distintos de información mutua y cabe imaginar una situación en la cual dependiendo de esta elección el valor de información mutua condicional caiga por arriba o por debajo del umbral. Esto ejemplifica las sutilezas involucradas en la adaptación de la Teoría de la Información Algorítmica a un contexto más aplicado.

universal y aditivamente óptima de referencia, tales que

$$x_i = q_i(\langle pa_i, u_i \rangle),\ i = 1, \dots n \tag{4.2}$$

donde identificamos dichos programas con la función parcial $\Sigma^* \to \Sigma^*$ que computan, $\langle -, - \rangle$ denota la concatenación de strings, e identificamos pa_i con la concatenación de los strings padre correspondientes.

Alternativamente, podemos adoptar el punto de vista de que esta tupla $(G, (q_i)_i, (u_i)_i)$ *genera* los strings $x_1, \dots x_n$.

Teorema 4.1.1. *Si $x_1, \dots x_n$ son los strings generados por un modelo causal algorítmico con DAG subyacente G, entonces son Markov-compatibles con G.*

Uno de los teoremas principales de [7] consiste en establecer una equivalencia entre distintas condiciones de compatibilidad de Markov algorítmica. Para esto, no utilizan la noción de independencia algorítmica basada en un umbral t sino que "revierten" a la "concepción asintótica" de la información algorítmica, interpretando a los símbolos x y, z, etc. como variables que pueden adoptar cualquier longitud y definiendo independencia condicional entre x e y dado z como $I(x : y|z) = O(1)$. Denotamos por $(x \perp\!\!\!\perp y|z)_{Kc,O(1)}$ esta noción de independencia algorítmica.

Teorema 4.1.2. *Dados strings $x_1, \dots x_n$ y un DAG G, las tres condiciones siguientes son equivalentes:*

1. ***Forma recursiva**:*

$$K(x_1, \dots x_n) = \sum_{j=1}^{n} K(x_j|pa_j^*) + O(1). \tag{4.3}$$

2. ***Forma local**: $x_1, \dots x_n$ y G son Markov-compatibles (con respecto a la noción de independencia algorítmica $(- \perp\!\!\!\perp - | -)_{Kc,O(1)}$).*

3. ***Forma global**: dadas colecciones de strings X, Y, Z en $x_1, \dots x_n$,*

$$(X \perp\!\!\!\perp Y|Z)_G \implies (X \perp\!\!\!\perp Y|Z)_{Kc,O(1)}. \tag{4.4}$$

[4] A diferencia del caso de variables aleatorias, debemos distinguir entre el string x_i y el vértice i debido a que podemos tener $x_i = x_j$ para $i \neq j$.

La equivalencia entre condiciones (2) y (3) conduce a pensar que podría ser fructífero adaptar un algoritmo de descubrimiento causal basado en el criterio de d-separación para operar sobre strings utilizando una noción de independencia definida a través de la información mutua algorítmica.

* * *

En [8], se desarrolla un formalismo más general que contempla el uso de distintas "medidas de información", entre las cuales pueden contarse la información de Shannon (con la cual se recupera la teoría de modelos causales probabilísticos) y la información algorítmica (con la cual se recupera la variante algorítmica desarrollada en [7]).

En este trabajo, se observa el hecho crucial de que *la noción de estabilidad para modelos probabilísticos puede no ser adecuada para medidas de dependencia basadas en longitud de descripción*, dado que estas métricas poseen una propiedad que Steudel et al. denominan *monotonicidad*. Bajo esta definición, una medida numérica de dependencia condicional es *monótona* si al condicionar sobre más variables nunca puede aumentar el valor de dependencia. Esto claramente no ocurre para la información mutua probabilística, dado que existe la paradoja de Berkson; de lo contrario, condicionar sobre una tercera variable nunca podría volver dependientes a dos variables que eran independientes marginalmente. Al emplear una medida de dependencia monótona, no puede tener lugar este tipo de situación. Steudel et al. observan empíricamente esta monotonicidad para medidas de información mutua I_Z donde Z es la complejidad de Lempel-Ziv (introducida en [23]) o bien una complejidad de descripción basada en gramáticas libres de contexto.

La propiedad de monotonicidad representa un problema para la aplicabilidad de la hipótesis de estabilidad, lo cual es claro si se considera una v-estructura $a \to b \leftarrow c$. Por d-separación, tenemos que $a \perp\!\!\!\perp c$, luego por monotonicidad de la medida de dependencia tenemos $a \perp\!\!\!\perp c|b$, pero esta independencia *no* es inducida por el criterio de d-separación. Afortunadamente, los autores presentan una noción alternativa de estabilidad, la *estabilidad monótona*[5], junto con una demostración de que el algoritmo PC de Spirtes y Glymour presentado en [18] es correcto también en un sentido modificado acordemente.

Definición 4.1.3 (Estabilidad monótona)**.** Un DAG G *representa* una lista de dependencias condicionales L *de forma monótonamente estable* si la siguiente condición es válida para toda elección de subconjuntos disjuntos X, Y, Z: si $(X \perp\!\!\!\perp Y|Z) \in L$ y Z es minimal sobre todos los conjuntos que hacen independientes a X e Y, entonces $(X \perp\!\!\!\perp Y|Z)_G$.

[5] Los trabajos [7] y [8] utilizan la expresión alternativa "fidelidad" para referirse a lo que aquí llamamos estabilidad.

Con esta definición, los autores muestran que si la lista de independencias condicionales L obtenida a partir del conjunto de observaciones de entrada es representada de forma monótonamente estable por un DAG G, entonces la salida del algoritmo PC es una representación monótonamente estable de L.

* * *

Los resultados mencionados aportan evidencia a favor de que una adaptación del algoritmo IC para trabajar con medidas de información computables que aproximen en algún sentido a la información algorítmica puede ser exitosa. Sin embargo, también podemos preguntarnos por cuál es la potencial utilidad de un método de análisis de este tipo.

Es posible dar múltiples respuestas, y de hecho en [7] se argumenta elocuentemente a favor de la utilidad de expandir los métodos de descubrimiento causal en esta dirección. Inversamente, podemos preguntarnos qué aporta la perspectiva de modelos causales a los trabajos ya existentes que estudian el uso de compresores como aproximaciones de la información algorítmica para realizar tareas de inferencia. Una referencia clásica en este sentido es [10], trabajo en el cual se utiliza la longitud de compresión para definir una *distancia* entre strings, la *distancia de compresión normalizada* o NCD por sus siglas en inglés. Esta métrica es empleada para realizar detección de comunidades (o *clustering*) jerárquica, y se basa en la noción de *compresor normal* introducida en ese mismo paper.

Esto difiere de la línea en la que se inscribe el presente trabajo, en la cual se contemplan métricas de similaridad *condicional*. Tal como se menciona en [7] y [8], la tarea de detección de comunidades jerárquica puede ser entendida como una tarea de reconstrucción de árboles, y una propiedad fundamental de los árboles es que solo existe un único camino (sin repetir vértices) entre cualquier par de vértices. Ahora bien, si en la estructura del proceso generador dos vértices están conectados por más de un camino, entonces será necesaria una métrica de similaridad condicional para poder reconstruir dicha estructura.[6]

[6] Un trabajo que sigue esta línea de razonamiento, reemplazando la métrica NCD por una métrica de distancia condicional, es [11]. El tipo de compresor modificado empleado para la definición de esta métrica resulta de interés para un posible trabajo a futuro tal como se detalla en la Sección 5.1.1.

4.2. Implementación de un algoritmo de descubrimiento causal sintáctico

Ahora procedemos a describir el principal aporte original de este trabajo.

La implementación fue desarrollada en el lenguaje de programación Python[7]. El trabajo realizado se concentró en el análisis de la primera fase del algoritmo IC, tal cual fue descrito en el Capítulo 2. Esta fase consiste en identificar el *esqueleto no dirigido* que comparten todos los DAGs compatibles con los datos. Respecto de las fases subsiguientes, veremos que el hecho de que el algoritmo permita identificar correctamente el esqueleto no dirigido del modelo que generó los datos no implica que el direccionamiento de las aristas en la fase 2 sea correcto.

El algoritmo fue implementado modularmente de manera tal de poder utilizar distintos tipos de tests. En otras palabras, permitimos que los juicios de independencia condicional que requiere el algoritmo provengan de distintos tipos de tests. Se implementó un test estadístico convencional, basado en *tablas de contingencia* (como veremos en la Sección 4.2.2), y un test sintáctico que utiliza la métrica de información mutua condicional $I_Z(x : y|z) = Z(x|z) + Z(y|z) - Z(x, y|z)$ para alguna métrica de complejidad sintáctica computable Z. Describimos estos tests con detalle en las secciones siguientes.

Los datos que recibe el algoritmo son conjuntos de strings *de igual longitud*. Esta restricción se debe a la forma en que está implementado el test estadístico, y podría ser relajada en un trabajo futuro.

4.2.1. El algoritmo

El algoritmo consiste en la primera fase del algoritmo IC, presentado en la Sección 2.4.1, adaptada de forma tal de recibir como entrada n strings de longitud ℓ en vez de una distribución de probabilidad (ver Algoritmo 2). Esta fase consistía en buscar, para cada par de variables, un conjunto "bloqueador" tal que dicho par de variables se vuelve independiente al condicionar sobre el conjunto, y agregar una arista no dirigida entre los nodos correspondientes a esas dos variables si no es posible encontrar ningún conjunto bloqueador. En este caso, las variables son strings, y los juicios de independencia condicional podrán ser realizados o bien con el test estadístico o bien con alguno de los tests sintácticos.

El algoritmo devuelve, entonces, un grafo no dirigido G. Siguiendo al algoritmo original, presentado en la Sección 2.4, interpretamos este grafo como una estimación del esque-

[7] La implementación puede encontrarse en https://github.com/GRGab/tesislic-public.

leto no dirigido de un *patrón* o grafo semidirigido, el cual a su vez representa una cierta clase de DAGs Markov-equivalentes (es decir, indistinguibles a través de d-separación).

Los vértices del grafo que debe ser construido no representan ya variables aleatorias sino, en cambio, variables valuadas en Σ^*, i.e. "registros" que contienen un string. Enumeramos arbitrariamente a estos registros como $1, \dots n$ y los distinguimos de los strings $x_1, \dots x_n$ por el simple hecho de que permitimos que haya strings repetidos. Decimos "registro", "variable" o "nodo" indistintamente.

Para poder hablar de independencia condicional entre estas variables o registros, necesitamos poder combinarlas de algún modo, de la misma manera que cuando hablamos de variables aleatorias necesitamos poder condicionar sobre *conjuntos* de variables aleatorias, e.g. $\{A, B, C\} = \{A\} \cup \{B\} \cup \{C\}$. Abstractamente, requerimos poder hablar de una unión, supremo o *join* de las mismas. Una diferencia con el presente caso, sin embargo, es que la unión de variables aleatorias es idempotente (e.g. $\{A\} \cup \{A\} = \{A\}$), mientras que para combinar strings x e y, debemos utilizar alguna función de formación de pares $x, y \mapsto \langle x, y \rangle$. En general, por ejemplo en el caso de la concatenación, no es cierto que $\langle x, x \rangle = x$. De esta manera, condicionar sobre strings $x_{i_1}, \dots x_{i_k}$ se interpretará como condicionar sobre el string

$$\langle x_{i_1}, x_{i_2}, \dots x_{i_k} \rangle = \langle x_{i_1}, \langle x_{i_2}, \langle \dots, \langle x_{i_{k-1}}, x_{i_k} \rangle \rangle \dots \rangle \rangle$$

que denotamos alternativamente como $\langle \{i_1, \dots i_k\} \rangle$ a través del conjunto de registros correspondientes[8].

Algoritmo 2: Descubrimiento causal sintáctico

Entrada: Strings $x_1, \dots x_n \in \Sigma^*$ y una relación de independencia condicional $R = (- \perp\!\!\!\perp - | -)$ computable.

Salida: Un grafo no dirigido G con vértices $x_1, \dots x_n$.

para cada $\{i, j\} \subseteq \{1 \dots n\}, i \neq j$ **hacer**
  buscar $S_{ij} \subseteq \{1 \dots n\}$ tales que vale $(x_i \perp\!\!\!\perp x_j | \langle S_{ij} \rangle)$ según R.

Construir grafo no dirigido G tal que i y j están conectados sii no se encontró un conjunto S_{ij}.

Tal como se presentó en la Sección 2.4, la primera fase del algoritmo IC consiste en identificar un *conjunto bloqueador* S_{ij} para cada par de nodos $i, j \in V, i \neq j$, si es que existe. Los detalles de este paso no están especificados por el algoritmo. Se implementó una búsqueda de fuerza bruta, como un *loop* sobre $\mathcal{P}(V - \{i, j\})$, comenzando por $\emptyset$ y

[8] Tal como se mencionó en la Sección 1.4, esta definición de formación de tuplas de longitud mayor a 2 incurre en cierta arbitrariedad para las funciones $\langle -, - \rangle$ no asociativas.

siguiendo en orden ascendiente de cardinalidad. Para cada subconjunto de nodos S, si $(x_i \perp\!\!\!\perp x_j | \langle S \rangle)$, se marca ese conjunto como S_{ij} y se detiene la búsqueda. Si no se encuentra ningún S tal. Esta implementación tiene una complejidad exponencial, en vez de la complejidad polinómica que puede alcanzarse (para grafos con grado acotado) mediante el refinamiento presentado en [18]; sin embargo en nuestro caso esta optimización no habría tenido un efecto observable debido a que en casi todos los grafos empleados, todo par de nodos tiene un vecino común.

Es importante notar que dependiendo del origen de los strings de entrada y de la manera en que se determinan los juicios de independencia condicional, el grafo resultante G puede o no ser una estimación (en algún sentido razonable) de la clase de DAGs Markov-equivalentes a un cierto DAG subyacente D_0. En nuestro caso, testeamos el algoritmo contra strings generados a través de modelos causales algorítmicos con un DAG subyacente D_0, y buscamos determinar si el grafo resultante aproxima a la clase correspondiente.

Ahora pasamos a describir los tests utilizados para determinar juicios de independencia condicional.

4.2.2. Test estadístico

El test estadístico implementado se basa fundamentalmente en la hipótesis de que los n strings de entrada pueden ser considerados *posición a posición*. Es decir que para determinar la correlación entre dos strings x e y, se compara el i-ésimo símbolo de x con el i-ésimo de y. En este contexto, tiene sentido pensar a cada posición como una *ronda* de un experimento. Es por esto que requerimos que todos los strings de entrada tengan la misma longitud. Por otro lado, para emplear este test como estándar de comparación, las pruebas realizadas parten del caso en que los strings de entrada están correlacionados precisamente de esta manera. Es decir que los modelos causales con los que generamos nuestros datos realizan operaciones posición a posición sobre los strings. Partiendo de este caso, añadimos luego variaciones que rompen esta estructura.

Así, el primer paso conceptual para entender el test estadístico consiste en interpretar a cada posición de cada string como si fuera una variable aleatoria, cuya distribución de probabilidad conjunta estimamos a partir de los datos provistos. Llamemos x_k^i al símbolo en la i-ésima posición del k-ésimo string de entrada.[9] Interpretamos a este símbolo como una realización de la variable aleatoria X_k^i. Para cada i consideramos a la tupla $\mathbb{X}^i = (X_1^i, \dots X_n^i)$ como una variable aleatoria compuesta. De esta manera podemos expresar la

[9] En esta sección reservamos los subíndices para indicar el nodo y los supraíndices para indicar la posición en el string, o la variable aleatoria correspondiente a dicha posición.

hipótesis de que la correlación ocurre únicamente símbolo a símbolo como la afirmación de que las variables aleatorias $\{\mathbb{X}^i\}_{i=1\ldots n}$ son independientes e idénticamente distribuidas. De esta manera, cada tupla $x^i = (x_1^i, \ldots x_n^i)$ puede considerarse como una realización de una *unica* variable aleatoria $\mathbb{X} = (X_1, \ldots X_n)$, y las frecuencias de aparición de cada tupla nos permiten estimar su distribución. Cada variable aleatoria X_i corresponde a un nodo diferente. Así, recuperamos la versión probabilística de los modelos causales, en la cual cada nodo tiene asignada una variable aleatoria. Estas variables aleatorias adoptan valores en Σ.

Independencia marginal

Para facilitar la exposición, describimos en primer lugar el test estadístico empleado para juzgar la independencia *no condicional* (o *marginal*) entre las variables aleatorias $X, Y \in \{X_1, \ldots X_n\}$ $(X \neq Y)$. Llamemos x, y a los strings que corresponden a esos mismos nodos; podemos reinterpretar a cada string como la secuencia de realizaciones observadas de la variable aleatoria correspondiente.

La idea básica consiste en aplicar un *test de contingencia*. En primer lugar, se construye una tabla de tamaño $|\Sigma| \times |\Sigma|$ cuya entrada (i, j) corresponde al número de veces en que aparecen simultáneamente el i-ésimo símbolo de Σ en el string x y el j-ésimo símbolo de Σ en y. La tabla constituye una estimación a la distribución de probabilidad conjunta de (X, Y), que denominamos *distribución empírica*. La idea del test es evaluar qué tanto difiere esta distribución estimada de la *distribución producto*, la cual posee las mismas distribuciones marginales para X y para Y que la distribución empírica y tiene como distribución conjunta al producto de dichas distribuciones marginales. Si las proporciones de casos en cada fila varían significativamente con la columna, entonces las dos variables son dependientes.

Para esto, se realiza una estimación puntual de las distribuciones marginales de X e Y a través de las distribuciones marginales empíricas, que se obtienen de sumar las filas y las columnas de la tabla (y dividir por el número total de elementos de la tabla). La hipótesis nula consiste en asumir que la distribución de (X, Y) es el producto de estas distribuciones marginales empíricas. La distribución de probabilidad sobre tablas de contingencia posibles, bajo la hipótesis nula, se calcula de manera exacta como una distribución hipergeométrica (test exacto de Fisher) o bien mediante una aproximación gaussiana a través de un test χ^2 de bondad de ajuste entre la distribución empírica observada y la distribución producto.

En nuestro caso, si bien se realizaron pruebas con ambos casos, terminamos utilizando

un test de contingencia con aproximación chi-cuadrado, implementado por la función `chi2_contingency` del paquete de software estadístico `scipy.stats`. Esta elección se fundamenta en el hecho de que no fue de particular interés el trabajar con strings de longitud pequeña. Por el contrario, se eligió la longitud de los strings empleados de forma tal de garantizar buenas condiciones de aplicación tanto para el test estadístico como para los tests sintácticos. Así, la diferencia en el resultado entre un test de Fisher y un test chi-cuadrado se vuelve despreciable, por lo cual se prefiere la eficiencia de cómputo del test chi-cuadrado.

De esta manera, se realiza el test de contingencia chi-cuadrado sobre los datos provenientes de los strings x, y, fijando previamente una significancia α para el test, y se rechaza la hipótesis de independencia si el p-valor obtenido p es menor a α. Es decir que se afirma (o, en términos del paradigma de tests de hipótesis, *no se puede rechazar*) $(X \perp\!\!\!\perp Y|\emptyset)$ si y solo si $p \geq \alpha$.

* * *

Existe una dificultad adicional para la aplicación del test de contingencia si los datos analizados permiten la posibilidad de aparición de ceros en la tabla de contingencia. Por un lado, la validez del test chi-cuadrado requiere un conteo alto en cada entrada de la tabla. Más aún, si una fila o columna de la tabla consiste enteramente de ceros, esto se traduce en un cero en la distribución conjunta bajo la hipótesis nula e implica una división por cero. Incluso para el test exacto de Fisher, en el cual no ocurre la división por cero, si únicamente una fila o columna de la tabla posee entradas no nulas, el p-valor es automáticamente 1, pues los valores observados son entonces los únicos posibles según la hipótesis nula. Esto en principio no es problemático dado que esta distribución "degenerada" bajo la hipótesis nula técnicamente *es* una distribución producto[10], con lo cual no sería deseable que el test rechace la hipótesis de independencia. Sin embargo se trata de un comportamiento singular del test.

La contemplación de estas dificultades técnicas llevó a identificar, tangencialmente, un aspecto fundamental del algoritmo IC que había pasado desapercibido. La aparición de ceros en una tabla de contingencia está relacionada con la dependencia determinística entre variables (por ejemplo, que si X vale 0, Y deba valer 1). Sin embargo, el criterio de d-separación no es válido para modelos causales con este tipo de dependencia [24]. Es por esto que los modelos causales empleados para estudiar el algoritmo son tales que las relaciones de dependencia entre las variables son no determinísticas (ver Sección 4.2.4). Esto elimina simultáneamente las dificultades en la aplicación del test de contingencia.

[10] Como ejemplo, podemos considerar $\Sigma = \{0, 1\}$ y la distribución conjunta $p(x, y) = \delta_{x,0}\ \delta_{y,1}$.

Independencia condicional

Ahora describimos la adaptación del test de contingencia al caso condicional.

Mantenemos las variables aleatorias X, Y como arriba y llamamos $Z = (Z_1, \ldots, Z_m)$ a la variable aleatoria compuesta por las m variables aleatorias sobre las cuales se desea condicionar ($Z_j \in \{X_1, \ldots X_n\}$, $Z_j \neq X, Y$ para todo j en rango). Notamos por z_j a los strings de datos correspondientes a dichos nodos.

Si se fija un valor simbólico α_j para cada Z_j, entonces la independencia entre X e Y condicional al evento $(Z_1 = \alpha_1, \ldots Z_m = \alpha_m)$ puede testearse restringiendo las muestras de X e Y que son contadas en la tabla de contingencia a aquellas que ocurren simultáneamente con la aparición del símbolo α_1 en el string z_1, el símbolo α_2 en el string z_2, etc. En otras palabras, miramos la posición i-ésima de los strings x e y (es decir los símbolos x^i, y^i) únicamente si $z_j^i = \alpha_j$ para todo j.

Concretamente, representamos al conjunto de strings de interés $(x, y, z_1, \ldots z_m)$ como una tabla de $m + 2$ columnas y ℓ filas, donde cada columna corresponde a un string, siendo las primeras dos columnas x e y, y cada fila corresponde a una posición en los strings. Sobre dicha representación, realizamos esta operación de condicionamiento de la siguiente manera: primero identificamos las filas de la forma $(\beta, \gamma, \alpha_1, \alpha_2, \ldots \alpha_m)$ (para algunos $\beta, \gamma \in \Sigma^*$) y eliminamos todas las filas que no son de esa forma. Luego eliminamos todas las columnas de la tabla salvo las primeras dos. El resultado es una serie de pares de símbolos (x_i, y_i) con los cuales se confecciona la tabla de contingencia.

Ahora bien, lo que se desea evaluar es la independencia condicional entre X e Y dado Z como variable aleatoria, lo cual implica condicionar sobre cada uno de los valores $(\alpha_1, \ldots \alpha_m)$ que la variable Z puede adoptar. Es decir que el test debe incorporar la información de todos los condicionamientos sobre los $|\Sigma|^m$ valores posibles para Z.

Con este fin, realizamos un test de contingencia distinto condicionando sobre el evento $(Z_1 = \alpha_1, \ldots Z_m = \alpha_m)$ para cada valor posible, obteniendo una lista de p-valores $p_1, \ldots p_m$. Asumiendo la hipótesis nula, estos p-valores deben estar distribuidos uniformemente en el intervalo $[0, 1]$. Aplicamos a continuación un test de Kolmogorov-Smirnov, o test KS, para comparar la distribución empírica acumulada de p-valores con la distribución acumulada uniforme sobre $[0, 1]$, obteniendo un nuevo p-valor p que denominamos p-valor agregado. Finalmente, rechazamos la hipótesis nula de independencia cuando $p < \alpha$, o, dicho de modo inverso, afirmamos que $(X \perp\!\!\!\perp Y|Z)$ si y solo si $p \geq \alpha$. Para el caso en que no se condiciona sobre ninguna variable, el test de independencia marginal se realiza según el procedimiento detallado en la sección precedente.

Para todas las pruebas realizadas, se utilizó el valor de significancia $\alpha = 0{,}05$.

4.2.3. Tests sintácticos

Para poder calcular afirmaciones de independencia condicional sintáctica, requerimos una métrica de complejidad computable $Z : \Sigma^* \to \mathbb{N}$, y por otro lado una función de formación de pares computable $\langle -, - \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$. Empleamos la notación

$$Z(x_1, x_2, \ldots x_n) := Z(\langle x_1, x_2, \ldots x_n \rangle) = Z(\langle x_1, \langle x_2, \langle \ldots, \langle x_{n-1}, x_n \rangle \rangle \ldots \rangle \rangle).$$

En el caso de la información algorítmica, la elección de la función $\langle -, - \rangle$ no modifica sustancialmente la información del par (x, y) sino que tan solo modifica una constante aditiva. Esto se debe a que siempre existe un programa (de longitud constante) que transforma una codificación de la tupla en otra y viceversa. En cambio, al usar aproximaciones computables a la información algorítmica esto no tiene por qué ser el caso.

De entre las distintas métricas de complejidad contempladas, se seleccionó la longitud de compresión dada por el compresor comercial *gzip* (versión 1.6), y la *I-complejidad* introducida por Becher y Heiber [25].[11]

El compresor gzip utiliza el algoritmo de compresión *Deflate*. Este consiste en una combinación de, por un lado, un algoritmo derivado del algoritmo LZ77 introducido en [28] (el cual se basa a su vez en la complejidad de Lempel-Ziv), y por el otro una codificación de Huffman [29]. En esencia, el primer paso consiste en aplicar el algoritmo LZ77 al string de entrada x. Este intenta reemplazar la aparición de un string y en x por una referencia a una aparición previa del mismo string en x. Más precisamente, si el algoritmo está analizando la posición i de x y encuentra que el string y, con $|y| = m$, que comienza en i (i.e. $x_i = y_1, \ldots x_{i+m-1} = y_m$) aparece previamente en x en una posición $k < i$, entonces reemplaza la descripción literal de y por una referencia o *pointer* a la posición j, junto con la indicación de que deben leerse m símbolos a partir de dicha posición. Así se obtiene un string consistente en una concatenación tanto de símbolos literales como de pointers (los cuales en principio son considerados como símbolos adicionales). Finalmente, este string resultante es comprimido aún más mediante la aplicación de un código de Huffman, el cual reemplaza a cada símbolo por un código autodelimitante de forma tal que los símbolos de mayor frecuencia son asignados códigos más cortos.

Por otro lado, la I-complejidad no es estrictamente hablando una complejidad de descripción, sino que es una métrica de la estructura combinatoria del string de entrada,

[11] Otras métricas consideradas fueron la complejidad de Lempel-Ziv [23], la métrica *Effort to Compress* introducida en [26] y complejidades de descripción basadas en autómatas de estados finitos [27].

lo cual facilita el análisis matemático de la métrica. Esto permite demostrar que esta posee una serie de propiedades básicas de la información algorítmica [25].

Describamos informalmente la definición. Para calcular la I-complejidad de x, se le asigna un puntaje a cada posición en el string de la siguiente manera. Primero, se computa el valor máximo de m tal que el string y, de longitud m, que comienza en i, aparece también "más adelante" en x, es decir que es también un sub-string de la cola de x que comienza en la posición $i+1$. El puntaje asociado a la posición i es aproximadamente inversamente proporcional a este valor máximo de m. Finalmente, la I-complejidad de x se define como la suma de los puntajes asignados a cada posición.

Tomando Z como una de estas dos métricas, aún queda por determinar la elección de función de pares $\langle -, - \rangle$. En un primer momento se estudiaron dos funciones diferentes: la concatenación, $\langle x, y \rangle = xy$, $x = x_1 \dots x_m$, y la *intercalación*, que, asumiendo $m < n$, manda $x, y \mapsto x_1 y_1 x_2 y_2 \dots x_m y_m y_{m+1} \dots y_n$ (los casos $m = n$ y $m < n$ son análogos).

Estas dos funciones empleadas no son biyectivas, a diferencia de las funciones de pares utilizadas en la teoría, pero esto no es en sí mismo un problema dado que trabajamos con métricas mucho más débiles que la información algorítmica, y por lo tanto nuestras preocupaciones son de una índole diferente[12]. Por ejemplo, como se mencionó antes, no esperamos que el comportamiento de la función compuesta $Z(\langle -, - \rangle | \langle -, - \rangle)$ sea similar al usar una u otra función de pares, sino que en cada caso debemos analizar con cuidado cómo el procedimiento especificado por Z interactúa con el especificado por $\langle -, - \rangle$. Más aún, dependiendo del tipo de proceso que genere los datos, puede resultar más adecuado usar una función de pares u otra.

En este sentido, preferimos concentrar el análisis en la función de concatenación, puesto que los resultados obtenidos con ella fueron más consistentes e interpretables, y tiene una mayor relación con el tipo de funciones estructurales que se usaron en este trabajo. Por lo tanto, en la Sección 4.3.3 se presentan simulaciones realizadas empleando únicamente la concatenación como función de pares.

* * *

Ahora estamos en condiciones de especificar los tests sintácticos utilizados. Fijando una función Z y una función $\langle -, - \rangle$, definimos la *complejidad condicional* de x dado y como

$$Z(x|y) := Z(x, y) - Z(y) \tag{4.5}$$

[12] El hecho de que estas codificaciones de pares no sean biyectivas tampoco es importante en el sentido de que pueden ser adaptadas a codificaciones biyectivas con un *overhead* pequeño.

y la información mutua condicional asociada según

$$I_Z(x : y|z) := Z(x|z) + Z(y|z) - Z(x, y|z) \tag{4.6}$$

con la convención de que $I_Z(x : y) := I_Z(x : y|\epsilon)$. Notemos que entonces

$$I_Z(x : y|z) = Z(x, z) + Z(y, z) - Z(x, y, z) - Z(z). \tag{4.7}$$

Estas definiciones no coinciden exactamente con las definiciones dadas para la información algorítmica, sino que en ese caso se trata de igualdades que se deducen de las definiciones, y valen solo a menos de ciertos términos constantes o incluso logarítmicos. Nos vemos sin embargo forzados a adoptar estas definiciones dado que las métricas de complejidad empleadas no poseen una versión condicional de manera *intrínseca*. En otras palabras, los programas que implementan estas métricas no pueden recibir entradas y de "ayuda", que modifiquen explícitamente la manera en que sería procesado el string x que constituye la entrada principal[13]. Tomando a gzip como ejemplo, lo mejor que podemos hacer es ver qué tan bien comprime x, qué tan bien comprime y, y qué tan bien comprime alguna combinación de x e y (concatenación, concatenación con símbolos delimitantes, concatenación autodelimitante, intercalado, etc.). La única manera que tenemos de permitir que la estructura de y modifique el procesamiento de x es permitir que el programa procese a ambos simultáneamente.

Sean entonces x, y los strings cuya relación de independencia queremos juzgar, condicional a los strings $z_1, \ldots z_m$. Eligiendo un valor de *umbral* $u \in \mathbb{Z}$ podemos realizar el siguiente juicio:

$$(x \perp\!\!\!\perp y|z_1, \ldots z_m) \quad \text{sii} \quad I_Z(x : y|\langle z_1, \ldots z_m\rangle) < u \tag{4.8}$$

Pensamos esta elección como análoga a la elección de un nivel de significancia para el test estadístico, en el sentido de que es un parámetro adicional para el test.

Como se mencionó en la Sección 4.1, en el caso en que $Z = Kc$ y el modelo causal subyacente que se desea estimar es un modelo causal algorítmico en el sentido de [7], se tiene la equivalencia entre las versiones local, global y recursiva de la condición de Markov usando, para definir la independencia condicional de strings, el criterio $(x \perp\!\!\!\perp y|\{z_1, \ldots z_m\})$ sii $I_{Kc}(x : y|\langle z_1, \ldots z_m\rangle) = O(1)$. Esta es la razón detrás de nuestra

[13] Yendo más lejos aún, para la información algorítmica, el argumento y con respecto al cual se condiciona puede ser *ejecutado* en el afán de encontrar una compresión óptima del argumento principal x: es una herramienta que puede ser usada en cualquier modo por cualquier programa p de forma tal que $\psi(p, y) = x$, y p puede ser tal que incluso ejecute y como programa en sí mismo.

definición del test sintáctico (si bien no implica una garantía de correctitud para otras elecciones de Z).

4.2.4. Modelos funcionales sintácticos

Siguiendo a [2], la elección de un modelo causal consiste en determinar una estructura causal o DAG subyacente G, una colección de funciones estructurales f_j, una para cada nodo del DAG, y un conjunto de variables aleatorias U_j de ruido, de forma tal que $X_j = f_j(PA_j, U_j)$ donde PA_j son los padres de X_j. Para el caso de los nodos raíz, la función f_j es supérflua y podemos considerarlos como meramente variables aleatorias (independientes). En esta sección se describe cómo se realiza la elección de estos ítems, dando lugar a ciertas clases de modelos funcionales, y por ende cómo se construyen los casos de prueba.

En el contexto de este trabajo, llamaremos *modelo funcional sintáctico* a los modelos causales utilizados. En estos modelos, las variables adoptan valores en strings, y las funciones estructurales son funciones de strings. Estos modelos pueden ser pensados como modelos probabilísticos, ya que generaremos strings pseudoaleatoriamente en los nodos raíz de los modelos, y que permitiremos que la relación entre un string y sus strings padre sea estocástica. Sin embargo, es conveniente realizar dos advertencias.

En primer lugar, no debe confundirse este hecho con la "interpretación estadística" de un conjunto de strings *fijo* presentada en la Sección 4.2.2 al introducir el test estadístico empleado. Cada modelo genera una distribución de probabilidad conjunta sobre n variables aleatorias valuadas en strings, tal como se presentó en el Capítulo 2. Ahora bien, para cada *realización* de ese conjunto, que consiste en n strings específicos, puede aplicarse el test estadístico, asumiendo que cada *posición* de string corresponde a una variable aleatoria independiente de las variables correspondientes a las demás posiciones, o bien considerar al string como un objeto fijo, o bien aplicarse un test sintáctico sobre los strings completos. Por supuesto, un tercer abordaje en este caso sería el de aplicar tests estadísticos que *no* asuman que cada posición es i.i.d. No considerar este caso no es cometer una "injusticia" para con los métodos de análisis estadísticos en general, sino que simplemente responde a nuestro objetivo, que es el de estudiar el desempeño de los tests sintácticos.

En segundo lugar, nuestro interés principal no es el analizar la distribución sobre strings generada por el modelo, sino analizar cada realización de los strings en sí misma. Si observamos una única realización de los n strings, podemos intentar entenderlos como el resultado de un modelo causal algorítmico. En este sentido, al generar múltiples reali-

zaciones de los strings, en algún sentido lo que estamos haciendo es generar una familia de modelos causales algorítmicos muy similares, que difieren únicamente en los valores específicos que adoptan los strings de ruido.

Respecto del tipo de strings considerado, el alfabeto empleado fue $\Sigma = \{0, 1\}$, salvo en algunos casos, que serán debidamente aclarados, en los cuales se utilizó $\{0, 1, 2\}$ o bien $\{0, 1, 2, 3, 4\}$. Por otro lado, todos los strings provenientes de un mismo modelo funcional sintáctico tendrán la misma longitud ℓ, y más aún mantenemos constante esta longitud a lo largo de todas las simulaciones presentadas. La potencia de los tests de independencia depende de esta magnitud dado que es la que determina la cantidad de datos de entrada para los mismos. Teniendo esto en cuenta, se realizaron pruebas con longitudes de strings cada vez mayores buscando que, al repetir el proceso de inferencia múltiples veces utilizando el test de independencia estadístico, hubiera una clara distinción entre aristas inferidas con baja probabilidad (falsos positivos) y aristas elegidas con alta probabilidad (verdaderos positivos). Así, se eligió $\ell = 20000$ como valor de trabajo.

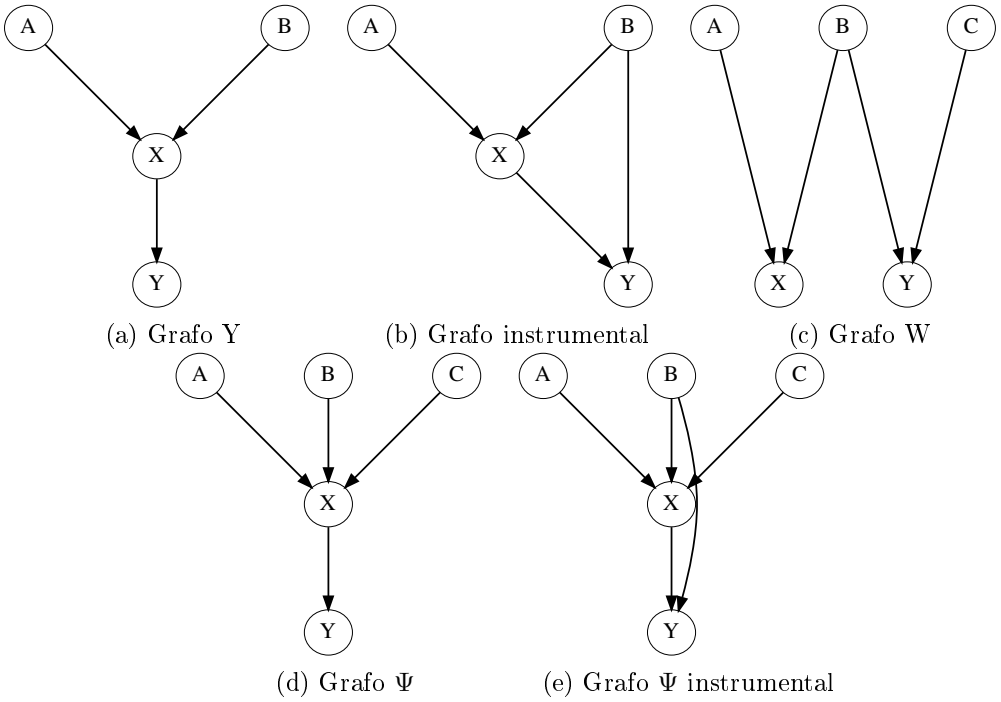
Estructuras causales

Del mismo modo que ocurre para modelos causales convencionales, cada modelo causal sintáctico tiene asociado un DAG subyacente, y no es necesario realizar ninguna adaptación de la teoría. En la Figura 4.1 se presentan los cinco DAGs que serán utilizados para la generación de los casos de prueba.

Los grafos elegidos poseen una cantidad de nodos baja, para mantener acotados tanto el costo computacional de las simulaciones como la dificultad de interpretación de los resultados. El grafo 4.1b es el grafo instrumental, que sirvió de ejemplo en los Capítulos 1 y 2, y fue aquel con el que se comenzó la mayoría de los análisis realizados. Los grafos 4.1a, 4.1d y 4.1e pueden pensarse como modificaciones simples a partir del grafo instrumental. El grafo 4.1c presenta otro patrón que no se encuentra en los demás grafos (un mismo nodo que es padre de nodos independientes).

Nodos raíz

Los nodos raíz de un modelo causal sintáctico serán strings de longitud ℓ fija obtenidos a partir de un generador de números aleatorios (esta elección de longitud fija la longitud de todos los strings subsiguientes, en la medida en que las funciones estructurales preserven las longitudes). Para cada nodo raíz j se elige una distribución de probabilidad sobre símbolos, $p_j : \Sigma \to [0, 1]$, y cada símbolo del string x_j se elige de manera i.i.d. a partir de dicha distribución.



(a) Grafo Y (b) Grafo instrumental (c) Grafo W

(d) Grafo Ψ (e) Grafo Ψ instrumental

Figura 4.1: Se presentan las cinco estructuras causales empleadas para testear el algoritmo de descubrimiento causal. Estas estructuras fueron combinadas con distintas familias de funciones estructurales dando lugar a un repertorio de modelos causales funcionales sintácticos.

Esta elección de strings para los nodos raíz es posiblemente la traducción más inmediata del caso de modelos funcionales convencionales a modelos causales sintácticos. Esta elección garantiza simultáneamente que los strings de entrada no tienen ninguna estructura interna y que no están correlacionados entre sí salvo, por supuesto, a través de Ía estructura correspondiente al generador de números pseudoaleatorios. Adoptamos la suposición razonable de que ninguno de los métodos empleados para inferir dependencia es capaz de detectar esta estructura.

De esta forma, tenemos como parámetros del modelo a Σ, G, ℓ y las distribuciones p_j asociadas a cada nodo raíz. A estos parámetros se suman las funciones estructurales en los nodos que no son raíz.

Funciones estructurales

En términos intrínsecos, cada nodo no raíz de un modelo causal sintáctico contiene una función de strings. En términos extrínsecos, las *observaciones* generadas por el modelo

consisten en un string asociado a cada uno de los nodos. Estas observaciones son el punto de partida para el proceso de inferencia, mientras que la estructura funcional permanece oculta.

Podemos describir la elección de las funciones estructurales de la siguiente manera. Consideremos el nodo j-ésimo en el DAG subyacente G. Si este nodo tiene m padres, seleccionamos una función f_j de m argumentos. Exigimos que las funciones tomen argumentos de longitud ℓ y devuelvan a su vez un string de la misma longitud, y por otro lado permitimos que sean estocásticas (i.e. que el resultado dependa de las entradas de manera probabilística). Así, podemos decir estas funciones son de tipo $\Sigma^{\ell m} \to \mathcal{D}(\Sigma^{\ell})$ donde $\mathcal{D}(\Sigma^{\ell})$ es el conjunto de distribuciones de probabilidad sobre (Σ^{ℓ}). Sin embargo seremos poco formales en este aspecto y a veces nos referiremos a las funciones estructurales como funciones determinísticas $\Sigma^{\ell m} \to \Sigma^{\ell}$. Omitimos también la formulación equivalente en términos de funciones determinísticas que toman como variable adicional una variable aleatoria de ruido, pero recordando que la estocasticidad en cada función estructural debe ser independiente de la presente en las demás.

Estas funciones fueron elegidas conjuntamente para todos los nodos, de manera paramétrica en la aridad. Es decir, para elegir la totalidad de las funciones estructurales de un modelo causal, se elige una clase de funciones $f^m : \Sigma^{*m} \to \Sigma^*$, $m = 1, 2, \ldots$, las cuales realizan alguna operación sobre sus argumentos que puede definirse de manera independiente de cuántos argumentos son exactamente. Por razones obvias, las funciones elegidas son computables, y más aún se buscó que fueran particularmente sencillas[14]. Se estudió también una variante de cada una de estas familias paramétricas, en la que son modificadas las funciones estructurales correspondientes a los nodos X e Y, introduciendo un *shift* en el string correspondiente al nodo (ver más adelante).

A continuación presentamos las familias paramétricas que fueron empleadas para los casos de prueba.

1. **Funciones XOR**. Dados dos bits $x, y \in \{0, 1\}$, denotamos por $x \oplus y$ a la *disyunción exclusiva* o *XOR* de los mismos, definida por $x \oplus y = 1$ si $x \neq y$ y 0 si $x = y$. En primer lugar extendemos esta operación a una operación entre una cantidad arbitraria de bits, aplicando iteradamente la operación $\oplus$:

$$XOR(x_1, \ldots x_m) := x_1 \oplus x_2 \oplus \ldots \oplus x_m \tag{4.9}$$

[14] Podría resultar deseable caracterizar las funciones estructurales utilizadas de forma más precisa en términos de clases de complejidad computacional o modelos de cómputo débiles que puedan implementarlas.

En segundo lugar, extendemos XOR a una función que opera símbolo a símbolo sobre strings $x_1, \ldots x_m \in \{0,1\}^*$. Así obtenemos una primera clase de funciones estructurales que denominamos *funciones XOR*.

2. **Suma módulo** n. Una variante de las funciones XOR que opera sobre alfabetos más grandes puede obtenerse considerando al XOR de dos bits como un caso particular de la suma módulo n, para el caso $n = 2$. En este caso se utiliza como alfabeto $\Sigma = \{0, 1, \ldots n-1\}$. $(a + b)$ mod n se define como el resto de dividir $a + b$ por n (así, por ejemplo, $(2 + 2)$ mod $3 = 1$). Las funciones se definen posición a posición y para una cantidad arbitraria de strings de entrada al igual que en el caso previo.

3. **Funciones de concatenación truncada**. Dados strings $x_1, \ldots x_m$, definimos la *concatenación truncada* $\mathrm{ct}(x_1, \ldots x_m)$ concatenando un fragmento de cada string. El primer fragmento se toma como los primeros ℓ/m símbolos de x_1. El segundo fragmento, que es colocado a continuación del primero, corresponde a los *segundos* ℓ/m símbolos de x_2, y así consecutivamente. El resultado es un string $\mathrm{ct}(x_1, \ldots x_m)$ que coincide con x_j entre las posiciones $(j-1)(\ell/m)$ y $(j)(\ell/m)$ (adoptando la convención de que el índice de la primera posición de un string es 0). Toda esta descripción es exacta en el caso en que ℓ/m es entero; en el caso contrario el primer fragmento se toma de longitud $\lceil \ell/m \rceil$ y los $m-1$ fragmentos restantes se toman de longitud $\lfloor \ell/m \rfloor$.

4. **Funciones de concatenación errática**. Dados $x_1, \ldots x_m$, su concatenación errática se obtiene concatenando fragmentos de longitud aleatoria según el siguiente procedimiento. En primer lugar, cada posición j entre 0 y $\ell - 1$ es marcada como "posición de cambio de string" con probabilidad p de manera independiente. Luego, para cada posición así marcada, se elige un string a partir de $x_1, \ldots x_n$ de manera equiprobable. Finalmente, el string de salida se forma colocando entre una posición marcada y la siguiente el fragmento del string correspondiente. Es decir que el string de salida coincide entre una posición marcada, j, y la siguiente posición marcada, llamémosla k, con el string asignado a la posición j. En todos los casos presentados se utilizó $p = 0{,}01$.

Cabe destacar que, para las dos clases de funciones de concatenación introducidas, el string de salida de la función coincide con cada uno de los strings de entrada en uno o más fragmentos. Esto nos permite esperar que la relación entre el string de salida y los strings de entrada sea detectable tanto mediante el test estadístico (dado que los símbolos del

string mostrarán una tendencia a coincidir con los símbolos de cualquiera de sus padres) como mediante tests sintácticos (dado que se repiten fragmentos extensos).

En los modelos considerados en este trabajo, se realizó una modificación de las funciones estructurales tal como han sido presentadas, agregando un ruido uniforme a su resultado. El propósito de este ruido no es añadir un desafío al proceso de inferencia sino simplemente romper las relaciones determinísticas entre entradas y salidas[15], tal como se detalla en la subsección siguiente.

Ruido

Debido a que el algoritmo IC se basa en el criterio de d-separación, es correcto únicamente para redes no determinísticas.[16] Es decir, si bien técnicamente los modelos causales determinísticos son un caso particular de los modelos causales estocásticos en general, en aquel caso las relaciones de independencia condicional implicadas por un DAG son diferentes: cuando un nodo depende de manera determinística de sus padres, al condicionar sobre estos el nodo se vuelve no solo independiente del resto de sus antecesores, sino también de los valores de sus descendientes. Para entender esto consideremos un ejemplo con funciones numéricas básicas. Supongamos que tenemos cuatro variables aleatorias reales X, Y, Z, W, con X e Y i.i.d. (por ejemplo gaussianas), $Z = X + Y$ y $W = 2Z$. En este caso, condicionar sobre X e Y hace que Z sea también independiente, como variable aleatoria, de W, lo cual no ocurriría si, por ejemplo, hubiera un cierto ruido sumado a Z: $Z = X + Y + R$ con R independiente. En estos casos debe reemplazarse el criterio de d-separación por uno diferente, denominado *D-separación* en [24].

En un principio, se consideró utilizar modelos funcionales totalmente determinísticos, para enfatizar el cambio de contexto de una teoría de la causalidad probabilística a una teoría de causalidad puramente algorítmica. Sin embargo, frente a esta dificultad, y para solucionar simultáneamente los detalles mencionados respecto de la aplicación del test estadístico, se decidió incorporar ruido a todas las funciones utilizadas en los modelos causales. El ruido consiste en la siguiente modificación al valor $f(x_1, \dots x_n) = s_1 s_2 \dots s_m$ de cualquier función estructural dada más arriba: para cada símbolo s_i, con probabilidad p_{ruido} el símbolo es modificado por algún otro símbolo, es decir un elemento de $\Sigma - \{s_i\}$,

[15] Una manera de decir esto es que, si no se incluye un ruido adicional, las distribuciones de probabilidad $f_j(x_1, \dots x_n) \in \mathcal{D}(\Sigma^\ell)$ serán muy singulares, asignándole probabilidad 0 a una gran cantidad de strings. Esto es el caso incluso para la concatenación errática, dado que, por dar un ejemplo, si el número total de posiciones en las que alguno de los argumentos vale 0 es N, entonces la probabilidad de que la salida tenga más de N ceros es nula.

[16] Por "no determinístico" referimos a que necesariamente haya una variabilidad del string resultante para un valor fijo de los argumentos, lo cual en este contexto implica estocasticidad a diferencia del uso que le dimos en el Capítulo 3.

según una distribución equiprobable. Los valores de ruido empleados durante las distintas pruebas oscilaron entre $p_{\text{ruido}} = 0{,}05$ y $0{,}2$.

Modelos con *shifts*

Recordemos el ejemplo de los ensayos clínicos, presentado en el Capítulo 1, como una interpretación posible del grafo instrumental. El nodo A corresponde a la asignación de sujetos a tratamientos, X corresponde al tratamiento efectivamente realizado por cada sujeto, Y es el resultado del tratamiento y U representa qué tan potencialmente beneficioso es cada tratamiento para cada sujeto. Considerando a B como variable *observable* (que es lo que venimos haciendo en nuestro caso, a diferencia del ejemplo en el cual B era latente), podríamos aplicar el algoritmo IC sobre nuestros datos para determinar si estos son compatibles con la estructura causal dada por el grafo instrumental. Ahora bien, nuestra elección de representar los datos como strings, y no como tablas, señala nuestro interés en apartarnos de los casos en los que toda la correlación existente se produce "ronda a ronda" en los datos, y considerar correlaciones entre distintas posiciones de los strings. Supongamos, por más inverosímil o caricaturesco que parezca, que durante el desarrollo del ensayo se pierde cierta información sobre la asignación de tratamientos a sujetos: se sabe el orden en que fueron convocados los sujetos hasta el momento, y el orden en que fueron asignados los tratamientos, pero algunos sujetos no fueron registrados, y algunos tratamientos no fueron registrados, de forma tal que las dos listas no están "sincronizadas" sino que existen desplazamientos entre distintos fragmentos de las mismas. En este caso se pierde la estructura "ronda a ronda" del conjunto de datos.

Podemos aventurar otra interpretación que quizá sea más intuitiva. Consideremos que se quiere evaluar el beneficio económico de emplear un sistema de riego automático sobre un terreno de cultivo. Se instala el sistema de riego, con una programación que fija una tabla de horarios, la cual nos dice si el riego se enciende o no cada día. Esta tabla nos da un string x. Por otro lado, una persona realiza una medición de la humedad de la tierra cada día, obteniéndose así un string y. El tiempo meteorológico actúa como factor de confusión dado que puede humedecer la tierra a través de lluvias pero también puede generar fallas temporarias en el sistema de riego. Consultando el registro meteorológico se obtiene un string b que representa el estado diario del tiempo. En este contexto, una variable instrumental que puede ayudar a determinar el efecto del sistema de riego sobre la humedad de la tierra es, por ejemplo, la presencia de cortes en el suministro de electricidad. Así, registrando qué días hay electricidad en el terreno, obtenemos un string a.

En esta situación, si no se registra día a día si el sistema de riego funciona o no, sino que se depende de la tabla de la programación del sistema, pueden ocurrir problemas como que, por ejemplo, el funcionamiento del sistema se desfase progresivamente respecto de su programación original. En este caso, nuevamente se pierde la correlación "ronda a ronda" en los datos.

¿Quiere decir esto que entonces las observaciones realizadas no son útiles? Por supuesto que no: siempre es posible realizar preprocesamientos de los datos intentando anular las distorsiones ocurridas. Por ejemplo, podemos realizar distintos desplazamientos (o *shifts*) de un string respecto de los demás y efectuar, en cada caso, un análisis estadístico ronda a ronda. Si todos los análisis nos muestran ausencia de correlación salvo por uno, para una cierta magnitud de desplazamiento, entonces podemos concluir que ese desplazamiento es el que debe ser revertido para "corregir" el error en los datos. Más aún, para un ejemplo como este podrían utilizarse también técnicas de series temporales como por ejemplo un análisis de correlación cruzada. Sin embargo está claro que es necesario realizar un análisis de los datos, ya sea como strings o como series temporales, que vaya más allá de la correlación ronda a ronda. El beneficio potencial de tratar a los datos como strings y analizar su correlación en términos sintácticos es que, en la medida en que las métricas de complejidad de strings empleadas sean suficientemente poderosas, un mismo método de análisis puede capturar, de manera uniforme, desviaciones muy diversas respecto de la estructura ronda a ronda.

En el presente trabajo, elegimos estudiar el efecto del desplazamiento entre strings, debido a que es un candidato natural a "desviación respecto de la correlación ronda a ronda". Dado que trabajamos con strings de longitud fija y queremos mantener siempre esta longitud, utilizamos un desplazamiento cíclico de los símbolos en un string que denominamos *shift*. Formalmente, si comenzamos con un string $x = x_1 x_2 \dots x_\ell$, definimos el shift (hacia izquierda) de magnitud k como $\text{shift}_k(x) := x_k x_{k+1} \dots x_\ell x_1 \dots x_{k_1}$.

En los casos que se presentan a continuación, cada vez que se introducen shifts en un modelo, esto se realiza modificando las funciones estructurales en los nodos X e Y. En particular se aplica un shift de magnitud 1 al string en el nodo X y un shift de magnitud 2 al string en el nodo Y (salvo para el caso de aplicación de tests sintácticos con gzip, en el cual estos valores son reemplazados por 8 y 16 respectivamente, por razones que serán explicadas más adelante). Denominamos "modelos funcionales con shift" a estos nuevos modelos modificados.

4.3. Simulaciones y resultados

Habiendo especficado un modelo causal como generador de los datos y un test de independencia a utilizar, lo cual determina unívocamente el procedimiento de análisis, la ejecución del código desarrollado devuelve un grafo no dirigido G para cada conjunto de strings pseudoaleatorios que generado en los nodos raíz. Para poner a prueba el procedimiento de inferencia mismo, se realizaron simulaciones en las que este procedimiento se repite una cierta cantidad de veces $n_{\text{repeticiones}}$ sobre strings generados a partir de un mismo modelo causal. En cada iteración, se generan nuevos strings en los nodos raíz de forma pseudoaleatoria y se recalculan los demás strings acorde a las funciones del modelo.

El resultado de cada una de estas simulaciones se presenta como un grafo no dirigido con aristas etiquetadas por fracciones f con $0 < f \leq 1$, las cuales indican la frecuencia con la que dicha arista fue inferida. Para facilitar la visualización, las aristas con frecuencia de aparición menor o igual a 0,1 se dibujan con una línea más suave, mientras que las aristas que no son inferidas en en ninguna iteración pero sí están presentes en el modelo del que provienen los datos se representan con líneas punteadas. Las aristas que no fueron inferidas en ninguna iteración ni pertenecen a la estructura del modelo generador no son representadas.

Por simplicidad, nos referiremos a las aristas que pertencen a la estructura del modelo generador como aristas *correctas*, y a las que no, como *incorrectas*. En la Sección 4.4 se discutirá qué tan razonable es, en cada caso, esperar que el algoritmo infiera exactamente las aristas correctas.

4.3.1. Primeras observaciones

Un parámetro a determinar para especificar los modelos causales es el de las distribuciones de probabilidad sobre símbolos empleadas en los nodos raíz. Durante las primeras pruebas con el test estadístico, los cuales se realizaron con funciones de tipo XOR y suma modular, se observó que el programa no infería ninguna arista cuando la distribución empleada en los nodos era la que asigna probabilidades iguales a todos los símbolos. Esto es totalmente esperable dado que, en tales modelos funcionales, los nodos hijo son marginalmente independientes de sus padres, aunque no sean *condicionalmente* independientes. Este fenómeno, denominado *sinergia informacional* en [30], representa una dificultad para el algoritmo IC que reaparecerá también más adelante.

Para entender esto, consideremos dos variables aleatorias U, V con valores en $\{0, 1\}$ y con probabilidad 0,5 para cada resultado; en otras palabras, dos monedas no sesgadas. Consideremos ahora la variable $W = U \oplus V$. Computando la distribución de probabilidad

condicional $\Pr(W = w|X = x)$, puede verse que U y W son marginalmente independientes, i.e. independientes cuando no se condiciona sobre ninguna otra variable. Esto es razonable intuitivamente puesto que, si uno desconoce el valor de V, W se comporta como una nueva moneda no sesgada. Por el contrario, si uno conoce el valor de dos de las tres variables, el valor de la tercera queda fijado de forma unívoca. Este comportamiento es paradigmático del fenómeno de sinergia informacional entre variables aleatorias.

¿Por qué representa esto un problema para el algoritmo? Sencillamente porque si U y W son marginalmente independientes, entonces el conjunto vacío es un conjunto bloqueador para esos nodos, y por lo tanto el algoritmo no asigna una arista entre ellos. En otras palabras, las influencias causales entre variables aleatorias que dan lugar a correlaciones sinérgicas, que solo aparecen al considerar tres o más variables simultáneamente, escapan al alcance del criterio de d-separación. En efecto, en [2] Pearl introduce el ejemplo del XOR de dos monedas no sesgadas como un ejemplo de distribución inestable, para las cuales el algoritmo no es capaz de reconstruir la estructura causal.

En vista de que el algoritmo no es correcto para este tipo de entradas, se consideraron únicamente distribuciones no equiprobables para los símbolos en los nodos raíz. Más aún, se observó una dependencia en las probabilidades de inferencia de aristas, empleando el test estadístico, con respecto a qué tanto se aleja la distribución de ser equiprobable. Nuevamente esto es esperable dado que mientras más cercana a ser equiprobable es la distribución en los nodos, menor es la dependencia marginal entre padres e hijos (cuantificada por ejemplo a través de la información mutua probabilística).

Por otro lado, para evitar introducir asimetrías en las probabilidades de inferencia de las distintas aristas que no respeten la simetría de las estructuras causales empleadas, se decidió utilizar una misma distribución de probabilidad en todos los nodos raíz. Así, en todas las simulaciones presentadas que usan un alfabeto binario se utilizó la distribución que le asigna probabilidad $p_1 = 0{,}4$ a la aparición de un 1 en cada posición de los strings raíz.

* * *

Previamente a añadir ruido a las funciones estructurales, se realizaron pruebas con el grafo subyacente 4.1b y funciones de tipo XOR. En este caso se observó que, debido a la estructura específica del grafo, $y = x \oplus u = z \oplus u \oplus u = z$ donde se usó en la última igualdad que $w \oplus w = 0^{|w|}$ (el string compuesto por $|w|$ ceros) para cualquier string binario w. En este sentido el modelo causal es "degenerado", dado que los dos caminos mediante los cuales U influye causalmente sobre Y (el camino directo y el camino a través de X) generan efectos que se cancelan perfectamente, dando como resultado una

influencia causal nula. Desde ya, el criterio de d-separación no es capaz de identificar este fenómeno, el cual constituye un ejemplo de independencia condicional inestable.

Esta observación motivó dos modificaciones. En primer lugar, motivó estudiar el mismo modelo pero utilizando alfabetos más grandes y aritmética módulo n con $n > 2$. En efecto, para este grafo particular, si $n > 2$ entonces las influencias causales de U sobre Y no se cancelan, con lo cual el modelo deja de ser degenerado en este sentido. Sin embargo, se siguió observando que el algoritmo identificaba siempre una arista entre Z e Y, i.e. no era capaz de encontrar un conjunto bloqueador para estas variables. Esto, de nuevo, tiene sentido dado que fijar el valor de X hace que el valor de Z determine el de U y viceversa; por lo tanto, la independencia $(Z \perp\!\!\!\perp Y|X)$ *no* está presente en la distribución de probabilidad empírica. El problema se encontraba en el hecho de que las relaciones de independencia que tiene sentido leer a partir de un DAG, sabiendo que se trata de variables determinísticas, no son las dadas por el criterio de d-separación, sino que corresponden a un criterio gráfico diferente tal como se mencionó en la Sección 4.2.4.

* * *

En todas las simulaciones que se presentan a continuación, se agregó ruido a las funciones estructurales, eligiendo $p_{\text{ruido}} = 0{,}1$. Como se mencionó anteriormente, por defecto el alfabeto usado es $\Sigma = \{0, 1\}$ y la distribución de probabilidad empleada sobre $\{0, 1\}$ es la que asigna a 1 la probabilidad $p_1 = 0{,}4$. La significancia empleada para el test estadístico es $\alpha = 0{,}05$ y la longitud de todos los strings es $\ell = 20000$.

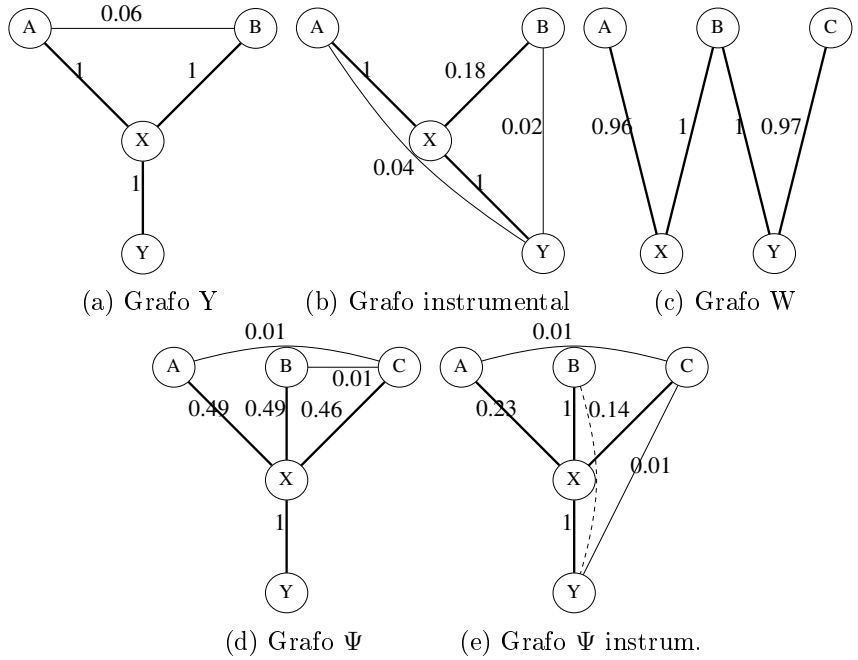
4.3.2. Simulaciones con test estadístico

Al realizar simulaciones empleando el test de independencia condicional estadístico expuesto en 4.2.2, se busca establecer un caso base con el cual contrastar los resultados obtenidos al utilizar tests sintácticos. Sobre cada modelo causal estudiado en esta sección, se presentan los resultados para $n_{\text{repeticiones}} = 100$ repeticiones del procedimiento.

Modelos XOR (y suma modular)

Comencemos considerando los modelos con funciones de tipo XOR sobre las cinco estructuras causales ya presentadas. En la Figura 4.2 se presentan los grafos inferidos.

En primer lugar podemos observar buenos resultados para los modelos con grafos subyacentes Y (4.2a) y W (4.2c): todas las aristas inferidas son correctas, salvo por la arista (A, B) en el modelo con grafo Y, cuya frecuencia de observación es cercana a la probabilidad de error tipo I del test estadístico dada por su significancia $\alpha = 0{,}05$. Por



(a) Grafo Y (b) Grafo instrumental (c) Grafo W

(d) Grafo Ψ (e) Grafo Ψ instrum.

Figura 4.2: Resultados de simulaciones sobre modelos de tipo XOR con test estadístico ($n_{\text{repeticiones}} = 100$). Las aristas están etiquetadas con la fracción de repeticiones en las que fueron inferidas. Se observa que las estructuras causales Y, W fueron inferidas correctamente en la mayoría de las repeticiones, mientras que en los demás casos se observan frecuencias bajas o nulas (líneas punteadas) para aristas correctas. Estos efectos están relacionados con la interacción entre las funciones XOR y la topología específica de estos grafos.

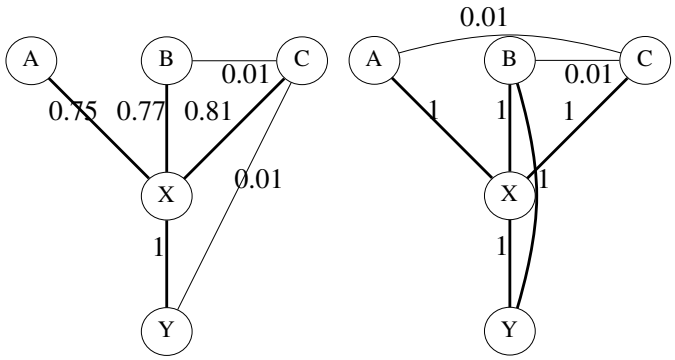
otro lado, en tan solo 4 de 100 iteraciones hay falsos negativos en el modelo con grafo W, lo cual también consideramos como un comportamiento esperado dado que todo test estadístico posee también una probabilidad de error tipo II.

En segundo lugar, consideremos los grafos instrumental y Ψ instrumental. Aquí se observa el problema ya mencionado con la aplicación iterada de las funciones de tipo XOR: en ambos grafos hay dos caminos de B hasta Y, y despreciendo el ruido el valor correspondiente a Y es el string $y = a \oplus b \oplus b = a$. Esto implicaría que el modelo instrumental colapsa al modelo Y. Como sí hay ruido en este modelo, el razonamiento no es exacto y, en efecto, no solo se observa una frecuencia de 0,02 para la arista (B, Y) sino que también se observa una frecuencia baja, de 0,18, para la arista (B, X), lo cual no ocurre para el modelo con grafo Y

Veamos ahora que este problema desaparece al reemplazar las funciones de tipo XOR por funciones de suma modular sobre un alfabeto más grande. Si reemplazamos los XOR

por suma módulo 3, utilizando el alfabeto $\Sigma = \{0, 1, 2\}$, se resuelve el problema con el nodo Y, pero simultáneamente aparece, para el modelo con grafo Ψ instrumental, un problema en el nodo X, puesto que este nodo tiene exactamente tres padres, y por lo tanto las frecuencias de aparición de todas las aristas salvo (X, Y) bajan al orden de 0,5[17]. En cambio, al considerar modelos funcionales de suma módulo 5 (con alfabeto $\{0, 1, 2, 3, 4\}$), sobre los grafos instrumental y Ψ instrumental, vemos en la Figura 4.3 que la estructura causal no dirigida es inferida correctamente en ambos casos.[18]

Por supuesto, cambiar las funciones estructurales XOR por suma módulo 5 es algo que podemos hacer para la generación de los casos de prueba, pero en un escenario real no es posible cambiar cómo se comportan las variables medidas (a menos que se pueda realizar intervenciones). Cabe enfatizar, entonces, que cambiar las funciones estructurales no resuelve el problema que tiene el algoritmo de inferencia en este tipo de modelos funcionales (porque cambia el modelo bajo estudio), sino que tan solo funciona como herramienta para entender mejor las causas de dicho problema.



(a) Grafo instrumental (b) Grafo Ψ instrumental

Figura 4.3: Resultados de simulaciones con test estadístico sobre modelos de suma módulo 5 y grafos subyacentes instrumental y Ψ instrumental ($n_{\text{repeticiones}} = 100$). Se observa que estas estructuras son inferidas con una frecuencia mucho más alta que la observada con funciones estructurales de tipo XOR, evidenciando el impacto de la interacción entre estructura causal y funciones estructurales.

Finalmente, volviendo a la Figura 4.2, podemos observar los resultados para el modelo XOR con grafo Ψ (4.2d). En este caso las aristas inferidas con una frecuencia sustancial son las correctas, pero las frecuencias de observación de las aristas que conectan a X con

[17] Esto es un ejemplo más, al igual que la situación ya discutida con el XOR, de cómo en los modelos estudiados puede haber interacciones muy fuertes entre las funciones estructurales elegidas y la estructura causal subyacente.

[18] Para el alfabeto de 3 símbolos, se asignaron las probabilidades (0,4, 0,35, 0,25) a los símbolos $0, 1, 2$ respectivamente. Para el alfabeto de 5 símbolos, se utilizaron las probabilidades (0,3, 0,15, 0,1, 0,15, 0,3)

sus padres son de aproximadamente 0,5. Ejecutando la simulación sobre el modelo de suma módulo 3 para este mismo grafo, estas frecuencias caen a aproximadamente 0, lo cual es esperable dada la topología, como se mencionó previamente, y para el modelo de suma módulo 5 se observó que dichas frecuencias suben hasta aproximadamente 0,8.

Modelos de concatenación

Consideremos ahora las dos familias de funciones estructurales que concatenan fragmentos de sus argumentos, las cuales denominamos previamente concatenación truncada y concatenación errática.

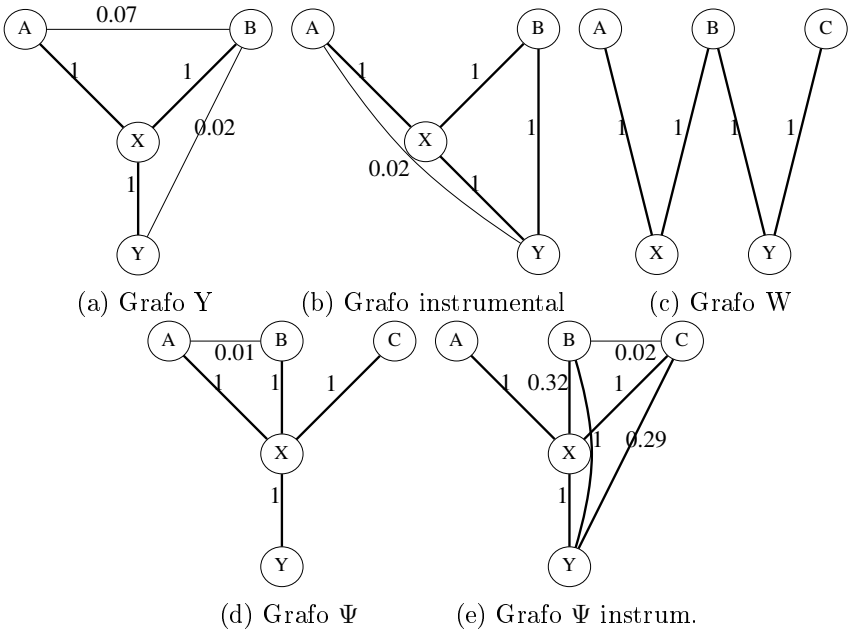
Según puede verse en la Figura 4.4, las simulaciones realizadas sobre modelos de concatenación truncada muestran que la implementación infiere correctamente el grafo subyacente: infiere con frecuencia 1 las aristas correctas mientras que las aristas incorrectas son inferidas con probabilidad menor a 0,1.

Sin embargo hay dos excepciones, las cuales se producen en el grafo Ψ instrumental (4.4e). Allí se tienen la arista correcta (B, X) y la arista incorrecta (C, Y), ambas inferidas con probabilidad de aproximadamente 0,3. Más aún, cabe preguntarse por qué no aparece también la arista incorrecta (A, Y) con probabilidad 0,3, i.e. por qué los resultados no poseen la simetría del grafo subyacente.

Para entender esto, debemos recordar el funcionamiento de las funciones de concatenación truncada. Ignorando por un momento el ruido, el string x en el nodo X contiene un tercio de cada uno de los strings a, b, c en A, B y C respectivamente. El orden asignado a los argumentos de la función estructural en X es A, B, C, por lo que el primer tercio de x coincide con a, el segundo tercio con b y el tercer tercio con c. Por otro lado, la primera mitad del string y en Y coincide con b, mientras que la segunda mitad coincide con x. Es decir que los primeros dos tercios de y coinciden con b, mientras el último tercio coincide con c.

Una primera observación es que el string y no depende de a. Esto explica la asimetría entre las frecuencias de aparición de la arista (C, Y) (0,3) y (A, Y) (cero). Contrariamente a lo que se inferiría mediante el criterio de d-separación a partir del grafo Ψ instrumental (4.1e), los nodos A e Y en este modelo son *marginalmente* independientes. Usando el lenguaje de redes causales probabilísticas, esto nos dice que la distribución de probabilidad conjunta sobre los cinco *strings* que es inducida por este modelo funcional es inestable (con respecto a la estructura causal del modelo): la relación de independencia $(Y \perp\!\!\!\perp A|\emptyset)$ es válida en esa distribución pero no es implicada por el criterio de d-separación en el grafo 4.1e.

Esta observación es interesante en sí misma ya que resalta el hecho de que *el grafo utilizado para definir el modelo funcional que genera los datos no necesariamente coincide con la estructura causal de esos datos generados*, y que lo haga o no depende de cómo interactúan las funciones estructurales elegidas con la topología del grafo. Volveremos sobre esta observación en la Sección 4.4.



(a) Grafo Y (b) Grafo instrumental (c) Grafo W

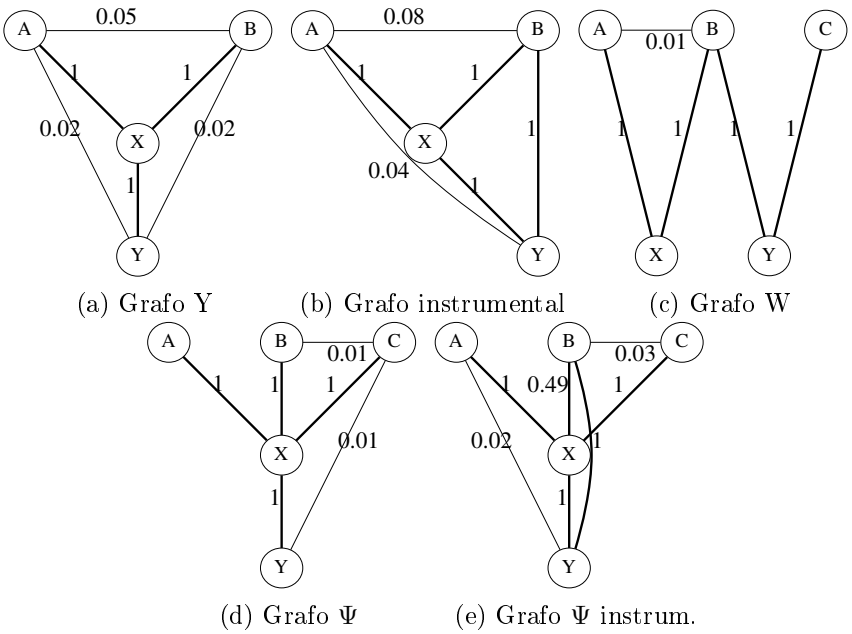
(d) Grafo Ψ (e) Grafo Ψ instrum.

Figura 4.4: Resultados de simulaciones sobre modelos de concatenación truncada con test estadístico ($n_{\text{repeticiones}} = 100$). Se observa que las estructuras causales Y, instrumental, W y Ψ fueron inferidas correctamente en la mayoría de los casos. Para el caso del grafo Ψ instrumental, se observa que tanto la arista correcta (B, X) como la arista incorrecta (C, Y) son inferidas aproximadamente el 30 % de las veces.

Ahora bien, ¿Por qué se infiere una arista entre C e Y el 30 % de las veces? El conjunto bloqueador identificado por el algoritmo para este par es $\{B, X\}$, en concordancia con lo implicado por el criterio de d-separación, pero el 30 % de las veces el algoritmo no identifica este conjunto sino que concluye que no existe un conjunto bloqueador. Esto está relacionado con el hecho de que incluso si se condiciona sobre B y X, seleccionando las posiciones de los strings en las que B y X adquieren un cierto valor fijo, en aquellas de esas posiciones que caigan en el último tercio de los strings, los strings c e y coinciden. Es decir que la dependencia estadística entre C e Y al condicionar sobre $\{B, X\}$ es pequeña, pero no nula, nuevamente a diferencia de lo que indicaría el criterio de d-separación sobre el grafo 4.1e.

Finalmente, respecto de la frecuencia de aparición relativamente baja de la arista (B, X), esto ocurre pues frecuentemente el algoritmo identifica a $\{Y\}$ como conjunto bloqueador para este par de nodos. Esto se puede entender mediante un razonamiento análogo a los ya dados.

Pasando ahora a las simulaciones realizadas con funciones de concatenación errática, vemos resultados completamente análogos, los cuales se presentan en la Figura 4.5. La única diferencia que cabe mencionar es que, sobre el grafo Ψ instrumental, desaparece la arista (C, Y). En este caso, dado que ya no está fija la elección de qué fragmentos de cada padre son concatenados para producir al hijo, ya no es cierto que a e y sean marginalmente independientes, y por lo tanto condicionar sobre A tiene un impacto en la dependencia entre C e Y. Así, en las iteraciones en las que el conjunto $\{B, X\}$ no logra bloquear a C de Y, sí se encuentra algún conjunto bloqueador que incluye a A. De esta manera, y dado que la función de concatenación errática sí es simétrica ante permutación de sus argumentos (salvo por un pequeño detalle de implementación que será explicado más adelante en la Sección 4.3.3), también desaparece la asimetría en las frecuencias de detección de aristas.



(a) Grafo Y (b) Grafo instrumental (c) Grafo W

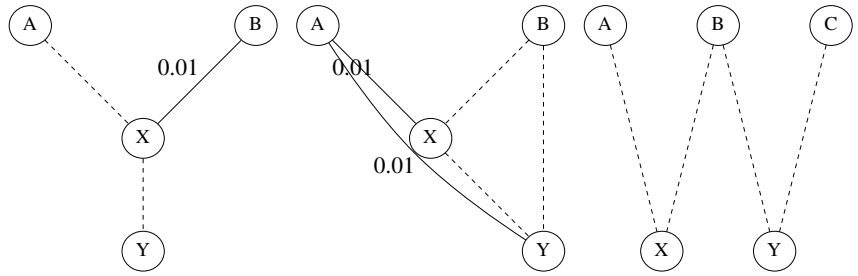
(d) Grafo Ψ (e) Grafo Ψ instrum.

Figura 4.5: Resultados de simulaciones sobre modelos de concatenación errática con test estadístico ($n_{\text{repeticiones}} = 100$). Se observa que todas las estructuras causales fueron inferidas correctamente en la mayoría de los casos, salvo por la arista correcta (B, X) en el grafo Ψ instrumental, la cual es inferida tan solo la mitad de las veces.

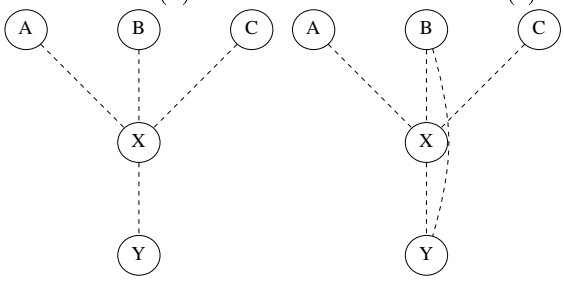
Introducción de shifts

Tal como se presentó en la Sección 4.2.4, se introdujo una modificación en todos los modelos funcionales presentados hasta el momento, según la cual se aplica un shift de magnitud 1 al string en el nodo X y un shift de magnitud 2 al string en el nodo Y.

Tal como era de esperar, al utilizar datos de entrada generados por los modelos funcionales con shift, el algoritmo devuelve grafos discretos al utilizar el test de independencia estadístico, dado que no es capaz de encontrar correlaciones entre los strings. Como ejemplo, podemos ver en la Figura 4.6 los resultados de las simulaciones para modelos de tipo XOR con shift. Los resultados para los demás modelos son análogos. Podemos preguntarnos si los tests sintácticos presentados en 4.2.3 sí son capaces de inferir la estructura de este tipo de modelos.



(a) Grafo Y (b) Grafo instrumental (c) Grafo W



(d) Grafo Ψ (e) Grafo Ψ instrum.

Figura 4.6: Resultados de simulaciones sobre modelos de tipo XOR con shifts (de magnitud 1 en X y 2 en Y), con test estadístico ($n_{\text{repeticiones}} = 100$). Se observa que dicho test no es capaz de identificar correlación entre ningún par de variables. Las líneas punteadas señalan aristas de los modelos generadores que no fueron inferidas en ninguna de las repeticiones.

4.3.3. Simulaciones con tests sintácticos

Consideremos ahora la aplicación de los tests sintácticos sobre los casos de prueba, y veamos si este tipo de análisis reproduce los resultados mediante el análisis estadístico más convencional.

Como se mencionó en la Sección 4.2.3, fueron utilizados dos tipos de test sintáctico: uno en el que la métrica de información Z corresponde a medir la longitud de compresión de strings mediante gzip ($Z = \text{gzip}$), y otro en el que Z es la I-complejidad de Becher y Heiber ($Z = \mathcal{I}$). Ahora bien, en verdad es necesario especificar un valor de umbral $t \in \mathbb{N}$ para que la relación de independencia condicional dada por el test sintáctico quede determinada de forma unívoca. En ambos casos se utilizó la la concatenación como función de formación de pares.

Modelos XOR

En un primer lugar, consideremos la aplicación del algoritmo con tests sintácticos sobre modelos de tipo XOR. Como puede verse en la Figura 4.7, empleando un test gzip con umbral $t = 0$, el algoritmo no es capaz de inferir las estructuras causales de los modelos generadores. Más aún, estos resultados nos permiten concluir que ningún valor de umbral permitiría inferirlas. En efecto, se puede observar que en todos los casos se infiere la arista (A, B), la cual no está en los grafos generadores puesto que a, b son siempre strings pseudoaleatorios independientes, y por lo tanto sería necesario un valor de umbral *más alto*, para que la información mutua calculada entre ellos (ya sea marginal o condicional) quede por debajo de dicho umbral y los strings sean juzgados independientes. Al mismo tiempo, se observa que múltiples aristas presentes en los grafos generadores no son inferidas con este valor de umbral, con lo cual sería necesario un valor de umbral *más bajo* para que estas aristas sean inferidas. De esta manera concluimos que no existe un valor de umbral t tal que los grafos inferidos coincidan con los grafos generadores. Si repetimos el análisis con la I-complejidad se obtienen resultados análogos. En las secciones subsiguientes justificaremos mejor la elección de un umbral para el test; por ahora nos basta con el argumento precedente para justificar nuestra conclusión.

La dependencia *marginal* entre dos strings u y v en un modelo de tipo XOR (o de tipo suma módulo n, para otros valores de n) es detectable por un test estadístico, lo cual garantizamos eligiendo una distribución no equipbrobable para los símbolos en los nodos raíz, pero esto no implica que sea detectable por un test sintáctico, dado que esta correlación no se expresa, al menos de manera directa, en la estructura a lo largo de múltiples posiciones de los strings. Consideremos una v-estructura en estos modelos,

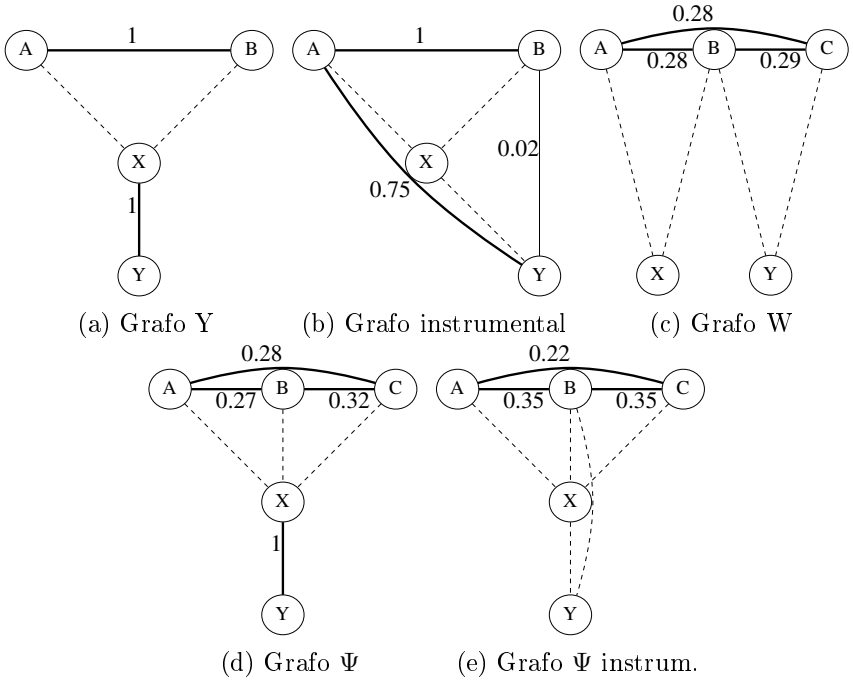


Figura 4.7: Resultados de simulaciones sobre modelos de tipo XOR con test gzip y umbral $t = 0$ ($n_{\text{repeticiones}} = 100$). Se observa que ninguno de las estructuras subyacentes a los modelos generadores es inferida correctamente. Las líneas punteadas señalan aristas de los grafos subyacentes a los modelos generadores que no fueron inferidas en ninguna de las repeticiones.

o equivalentemente la puerta XOR que opera bit a bit sobre los strings de entrada: $w = u \oplus v$. Para encontrar correlación sintáctica entre, por ejemplo, u y w, nuestros tests sintácticos estudian la estructura del string uw; en particular gzip busca comprimir dicho string identificando substrings repetidos. Ahora bien, los substrings repetidos en u no guardan ninguna relación con los substrings repetidos en w en el caso en que v es aleatorio (es decir, bits i.i.d. con $p_1 = 0{,}5$). Si v tiene algo de estructura (por ejemplo, $p_1 = 0{,}4$, como es el caso de nuestros strings a y b) entonces algunas de las cadenas repetidas en u "sobrevivan" en w (aquellas posiciones consecutivas en las que v contiene ceros, y por lo tanto al ser sumado con el substring proveniente de u, da como resultado ese mismo substring). Sin embargo estos strings repetidos serán muy cortos y poco frecuentes. En la medida en que la correlación entre strings u y w resulta inobservable, podemos pensar que nos encontramos nuevamente ante una situación de sinergia informacional en el sentido introducido al comenzar la Sección 4.3, puesto que no importa si el hecho de condicionar sobre el string v hace que los strings se vuelvan sintácticamente dependientes:

al algoritmo le alcanza con juzgar a los dos strings como marginalmente independientes para descartar la posibilidad de una arista entre sus nodos correspondientes[19].

Este razonamiento nos permite entender por qué los tests sintácticos tienen dificultades para detectar correlaciones en este tipo de modelos. Es por esto que a continuación nos concentraremos en analizar simulaciones utilizando las otras dos clases de funciones estructurales (concatenación truncada y errática), dejando como trabajo a futuro la exploración más profunda de la aplicabilidad de estos tests sobre modelos tipo XOR (por ejemplo, variando la probabilidad p_1 de obtener 1 en los nodos raíz).

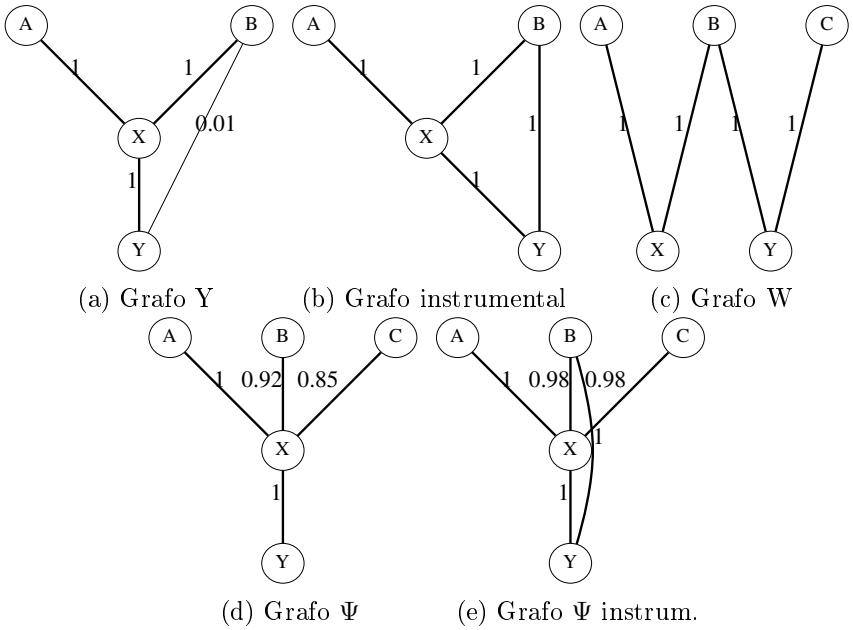
Modelos de concatenación con gzip y análisis de la elección de umbral

En las Figuras 4.8 y 4.9 podemos observar el resultado de las simulaciones empleando el test gzip sobre Íos modelos de concatenación truncada y errática, respectivamente, empleando como umbral $t = 0$. Vemos que los grafos inferidos coinciden con los grafos generadores en ambos casos, salvando el detalle de las aristas (B, X) y (C, X) en el modelo de concatenación errática sobre el grafo Ψ, las cuales son inferidas con probabilidad apenas mayor al 50 %. La existencia de una asimetría entre los nodos B y C, por un lado, y el nodo A por el otro, puede atribuirse a un detalle de implementación de la función de concatenación truncada: el primer fragmento del string resultante siempre proviene del string que fue pasado como *primer* argumento a la función (el cual es a en todos estos casos). Esta pequeña asimetría en la función respecto de permutaciones de sus argumentos explica a su vez la asimetría en las frecuencias de las aristas en el modelo de concatenación errática sobre el grafo Ψ instrumental.

Vemos así que la elección de umbral $t = 0$ resulta satisfactoria; sin embargo, hasta ahora no hemos discutido qué criterio puede emplearse para orientar dicha elección.

En principio, un criterio natural para determinar qué valores de información mutua deben considerarse despreciables y son significativos consiste en asegurarse de que el umbral esté por arriba del valor de información mutua $I_Z(x : y)$ para dos strings x, y de longitud $\ell = 20000$ para los cuales *se sepa que son independientes*. En una situación de inferencia real, en la cual no se conoce el modelo funcional que genera los datos ni está garantizado que exista tal cosa (los datos pueden provenir de procesos "fuera de clase" que no admitan un modelado de este tipo), sería necesario entonces contar con información

[19] En verdad, un compresor convencional como gzip no puede utilizar el string adicional v para mejorar sustancialmente la compresibilidad del string uw, con lo cual no estaríamos realmente en el caso de sinergia informacional si usamos esta métrica de información. Sin embargo, un compresor "modificado" capaz de identificar ciertos substrings de su entrada como el resultado de aplicar XOR sobre otros dos substrings, sí aprovecharía la información contenida en v para reducir la . Este tipo de ideas se explora en la Sección 5.1.1.



(a) Grafo Y (b) Grafo instrumental (c) Grafo W

(d) Grafo Ψ (e) Grafo Ψ instrum.

Figura 4.8: Resultados de simulaciones sobre modelos de concatenación truncada con test gzip y umbral $t = 0$ ($n_{\text{repeticiones}} = 100$). Los grafos subyacentes a los modelos generadores son inferidos correctamente en la mayoría de las repeticiones.

adicional, proveniente de análisis auxiliares o quizá del dominio de conocimiento específico en el cual se aplican estos modelos, que identifique a partir del conjunto de variables (strings) bajo estudio dos de ellas que se pueden asumir independientes[20].

En el caso presente, dado que conocemos los modelos generadores de los datos, sabemos que los strings que deben ser considerados independientes para el tipo de procesos involucrados son los strings de longitud $\ell = 20000$ en los que cada bit es 1 con probabilidad $p_1 = 0{,}4$ de forma independiente (modificar p_1 modifica la distribución de valores de información mutua para este tipo de strings). En particular, para estos valores de los parámetros la información mutua de dos strings independientes (como lo son a y b en todos

[20]Es interesante notar que la dificultad de esta situación reside en que para *definir* nuestra noción operacional de independencia necesitamos fijar un umbral, pero para fijar el umbral requerimos una noción de independencia *a priori*. Incluso si pudiéramos acceder, a través de un oráculo, a los valores exactos de información algorítmica para nuestros strings de interés, en la medida en que sigamos sin poder interpretar en estos strings afirmaciones teóricas de tipo asintótico, este problema persistiría. Una respuesta a este planteo podría ser que el caso de strings finitos es análogo al de una muestra finita de una distribución de probabilidad, y que la noción de independencia condicional probabilística no se define con respecto a muestras finitas sino con respecto a la distribución de probabilidad subyacente. El análogo de dicha distribución en el caso algorítmico correspondería a secuencias simbólicas infinitas, que sería por tanto el tipo de objeto con respecto al cual la independencia algorítmica puede definirse de forma absoluta.

los modelos empleados) es de $t' = 453 \pm 25$. A la luz de esta información, puede resultar sorprendente que hayamos obtenido resultados razonables usando un umbral *menor* a este valor, puesto que, según se podría pensar, en tal caso los nodos A y B deberían ser haber sido juzgados dependientes y deberíamos, por lo tanto, ver en los reultados una arista entre ellos con alta probabilidad.

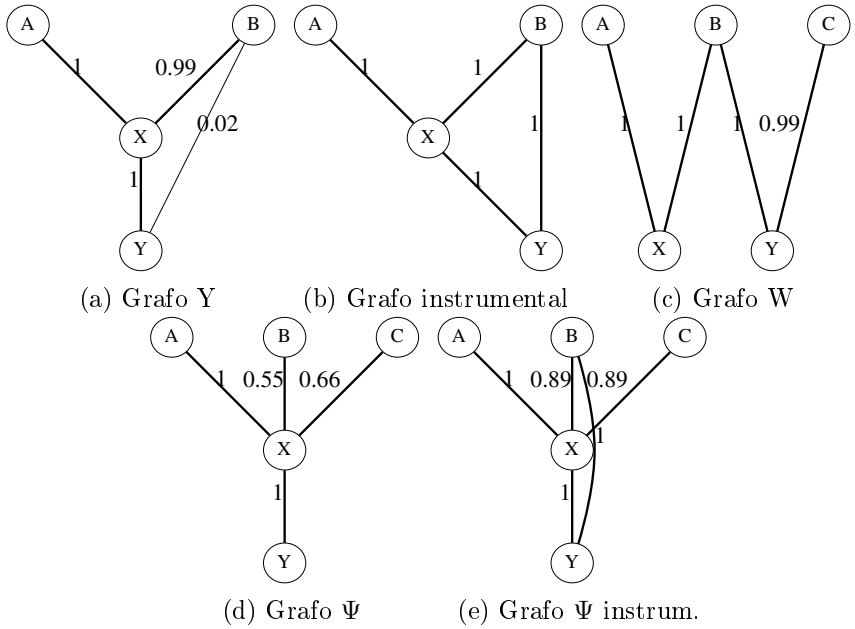


Figura 4.9: Resultados de simulaciones sobre modelos de concatenación errática con test gzip y umbral $t = 0$ ($n_{\text{repeticiones}} = 100$). Los grafos subyacentes a los modelos generadores son inferidos correctamente en la mayoría de las repeticiones, observándose frecuencias menores al 90 % únicamente en las aristas correctas (B, X) y (C, Y).

Más sorprendente aún es el hecho de que si se emplean valores de umbral del orden de 450, los grafos resultantes difieren considerablemente de los grafos generadores: la mayoría de las aristas que sí están presentes en los segundos son inferidas con una frecuencia muy baja, y algunas simplemente no son inferidas nunca para el número de repeticiones empleado. Esto implica que el criterio mencionado arriba resulta absolutamente inadecuado, pues *ningún* valor por encima del valor de información mutua entre strings independientes permitirá inferir la estructura de los grafos generadores.

La clave para entender estas dos observaciones se encuentra en el hecho de que el funcionamiento del algoritmo no utiliza únicamente juicios de independencia marginal. En efecto, lo que ocurre en las simulaciones presentadas en las Figuras 4.8 y 4.9 es que al elegir t por debajo de la información mutua entre strings independientes, el algoritmo

aún así logra bloquearlos dado que juzga que son *condicionalmente* independientes. Esto señala un hecho importante sobre nuestra función computable $I_Z(- : -|-)$: condicionar sobre strings tiende a disminuir la información mutua, en concordancia con lo reportado para funciones de información basadas en compresores en [8].

En nuestro caso, en el que empleamos la medida de información mutua I_{gzip}, no solo se observa que el condicionamiento no puede aumentar la información mutua entre strings (comportamiento monótono) sino que, más aún, en general este disminuye considerablemente la información mutua entre strings. Así, fijar $t \simeq t'$ resulta inadecuado porque los valores de información mutua condicional caen sistemáticamente por debajo de t'. Por otro lado, dado que esto ocurre en particular para strings que ya eran marginalmente independientes (strings pseudoaleatorios), esto permite al algoritmo descartar las aristas entre ellos incluso cuando $t < t'$.

En síntesis, en nuestro caso de estudio conocemos el grafo mediante el cual se generaron los datos y usamos esa información para determinar si existe un umbral t tal que el algoritmo devuelve como salida el esqueleto no dirigido de dicho grafo. En este caso, $t = 0$ es una elección de umbral posible. Algo distinto ocurriría en una situación de inferencia real, en la cual solo se poseen los datos observados, en cuyo caso puede ser deseable realizar un barrido en valores de umbral y determinar el valor más adecuado según criterios adicionales (ver Sección 4.4.3).

Finalmente, mencionamos ahora que el hecho de que el algoritmo asigne conjuntos bloqueadores no vacíos a pares de strings que según nuestro modelo generador debieran ser marginalmente independientes es indicativo de que las fases de orientación de aristas del algoritmo IC, las cuales quedan por fuera del alcance de este estudio, fallarán. Volveremos sobre esto en la Sección 4.4.

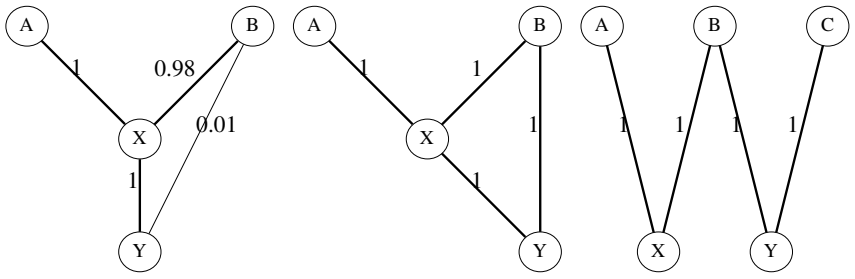
Introducción de shifts

Volvamos ahora sobre los modelos funcionales generadores con shifts, introducidos en la Sección 4.2.4. La motivación detrás de emplear esta clase de modelos fue la posibilidad de que los tests sintácticos sean capaces de inferir su estructura causal subyacente. Esto es efectivamente el caso, tal como puede observarse en la Figuras 4.10 y 4.11. En ellas se muestran los resultados de simulaciones en las que se utilizó el test gzip con umbral $t = 0$ sobre los modelos de concatenación truncada y errática, aplicando shifts sobre los strings en el nodo X y el nodo Y.

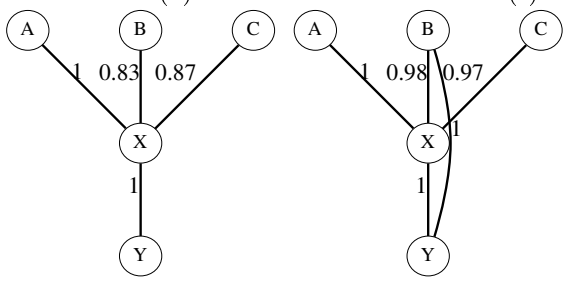
En este caso, debido a que la implementación del test gzip sobre los strings trabaja opera una representación de los strings en términos de *bytes* (es decir, toma como símbolos

básicos a los bloques consecutivos de 8 bits en el string), los shifts de magnitud 1 en X y magnitud 2 en Y fueron reemplazados por shifts de magnitud 8 y 16 respectivamente. Si se emplean en cambio los valores previos, tal como es de esperar el test no es capaz de encontrar correlaciones entre los strings, debido a que todos los bytes de cada string se ven modificados.

Cabe destacar que no fue necesario modificar el valor de umbral respecto de las simulaciones previas para que el test de independencia sintáctica permita reconstruir las estructuras de los grafos generadores.



(a) Grafo Y (b) Grafo instrumental (c) Grafo W



(d) Grafo Ψ (e) Grafo Ψ instrum.

Figura 4.10: Resultados de simulaciones sobre modelos de concatenación truncada con shifts (de magnitud 8 en X y 16 en Y), con test gzip y umbral $t = 0$ ($n_{\text{repeticiones}} = 100$). Los grafos subyacentes a los modelos generadores son inferidos correctamente en la mayoría de las repeticiones. Los resultados son análogos a los de los modelos sin shifts correspondientes.

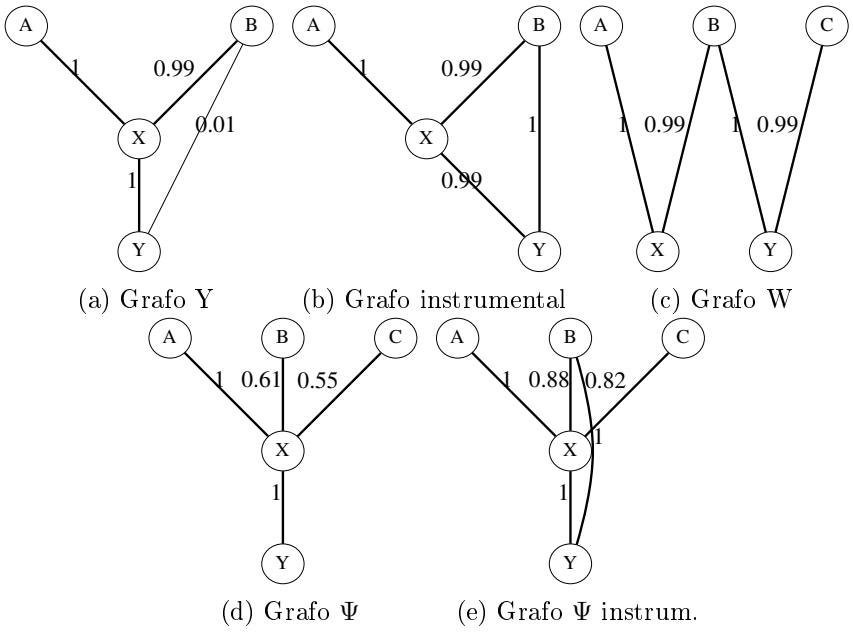


Figura 4.11: Resultados de simulaciones sobre modelos de concatenación errática con shifts (de magnitud 8 en X y 16 en Y), con test gzip y umbral $t = 0$ ($n_{\text{repeticiones}} = 100$). Los grafos subyacentes a los modelos generadores son inferidos correctamente en la mayoría de las repeticiones, observándose frecuencias relativamente bajas de alrededor de 60 % únicamente en las aristas correctas (B, X) y (C, Y). Los resultados son análogos a los de los modelos sin shifts correspondientes.

Modelos de concatenación con I-complejidad

Al emplear la I-complejidad para determinar independencias condicionales, se observa nuevamente el fenómeno de que la información mutua condicional cae sistemáticamente por debajo del valor de información mutua marginal entre strings correspondientes a nodos raíz, el cual en este caso es $t' = 271 \pm 2$. También se observa una menor variabilidad en los valores de I-complejidad entre una repetición y otra de la simulación, dando lugar a una mayor cantidad de frecuencias observadas iguales a 0 o 1.

En este caso, nuevamente es posible inferir los esqueletos no dirigidos de todos los grafos generadores, para las dos clases de función de concatenación, tanto con shifts como sin ellos. Sin embargo, a diferencia del caso anterior, no fue posible encontrar un único valor de umbral con el cual las cinco estructuras fueran inferidas simultáneamente. Para mostrar este fenómeno, se eligió presentar únicamente los resultados con funciones de concatenación errática, dado que los resultados con concatenación truncada son similares. Además, se omiten los resultados de las simulaciones con modelos sobre el grafo W, dado que este es inferido con frecuencia cercana a 1 para todos los umbrales presentados.

En la Figura 4.12 se presentan los resultados de estas simulaciones sobre modelos sin shifts (los resultados sobre los modelos con shifts son análogos y se presentan en la Figura 4.13). La primera fila corresponde a los grafos inferidos con umbral $t = 35$. Se observa que para los modelos con grafos Y e instrumental se obtienen grafos completos, mientras que para los modelos Ψ y Ψ instrumental se obtienen los esqueletos de los grafos generadores. En ambos casos, las frecuencias observadas son básicamente 0 o 1: hay muy poca variablidad entre una repetición y otra de las simulaciones.

Si por el contrario observamos la tercera fila, correspondiente al umbral $t = 50$, vemos que son los grafos Y e instrumental aquellos cuyos esqueletos son reconstruidos, mientras que en los otros dos casos se observa que varias aristas presentes en los grafos generadores son inferidas con probabilidades bajas o nulas.

Para determinar si es posible un valor intermedio de t que permita reconstruir los esqueletos de todos los grafos generadores simultáneamente, puede realizarse un barrido o bien estudiar cuáles son los valores de información mutua condicional que están siendo calculados. Siguiendo este camino, podemos comenzar observando los valores de información mutua correspondientes a los pares (A, X), (B, X) y (C, X) en el grafo Ψ. Para que el algoritmo infiera una arista entre estos pares de vértices, es necesario que no encuentre un conjunto bloqueador para los mismos. Dado que la información mutua decrece a medida que agregamos strings al conjunto sobre el cual se condiciona, nos basta

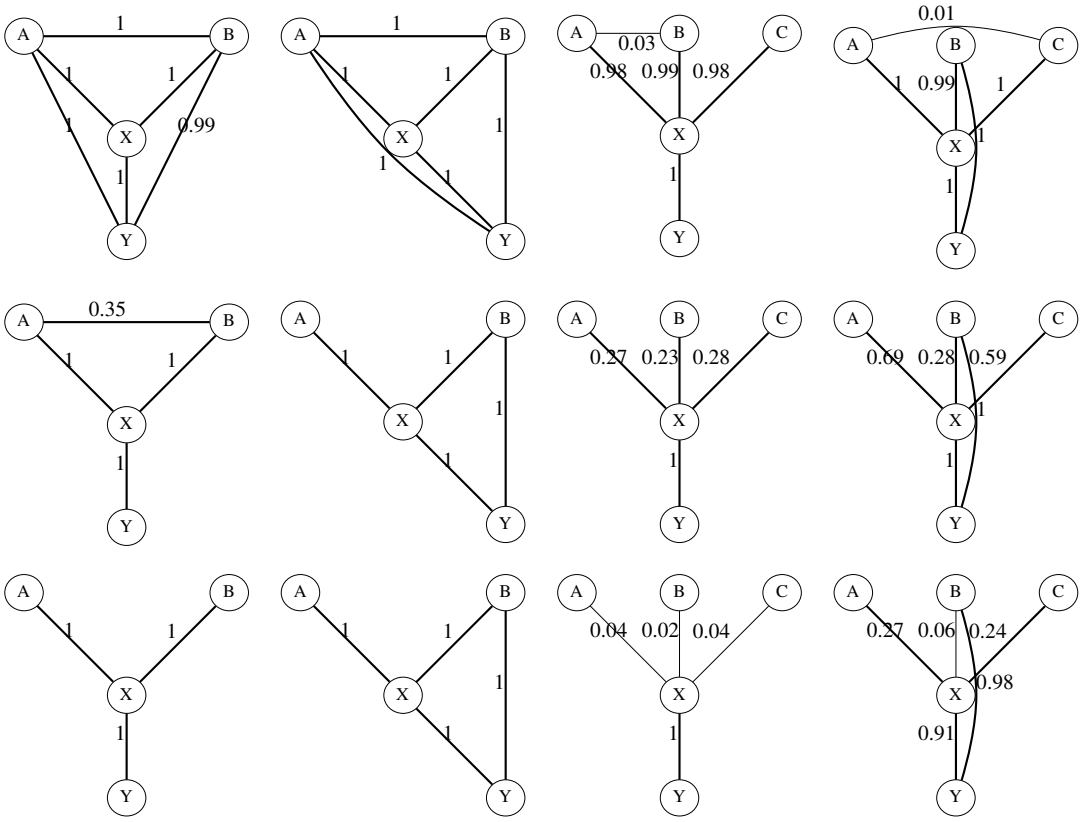


Figura 4.12: Resultados de simulaciones sobre modelos de concatenación errática con test de I-complejidad y umbrales $t = 35$ (arriba), 45 (centro), y 50 (abajo) ($n_{\text{repeticiones}} = 100$). De izquierda a derecha, el orden de presentación de los resultados es Y, instrumental, Ψ, y Ψ instrumental. Se observa que los modelos Y e Instrumental son inferidos correctamente para $t = 50$, mientras que los modelos Ψ y Ψ instrumental son inferidos correctamente empleando $t = 35$. Para el valor de umbral intermedio $t = 45$, se observan frecuencias de detección de aristas intermedias, indicando que los valores de información mutua relevantes para el algoritmo oscilan por arriba y por debajo de dicho valor. Se omiten los resultados para el modelo sobre el grafo W, el cual es inferido correctamente en los tres casos.

con que las cantidades

$$I_Z(a : x|\langle b, c, y\rangle)$$
$$I_Z(b : x|\langle a, c, y\rangle)$$
$$I_Z(c : x|\langle (a, b, y\rangle)$$

sean mayores a t. Observando estos valores en algunas repeticiones de la simulación vemos que estos valores oscilan entre 40 y 50. Es decir que podríamos subir el umbral desde 35

hasta un valor de 40 sin que estas aristas dejen de ser inferidas.

Por otro lado, para que en el grafo Y la arista (A, B) *no* sea inferida, siguiendo el mismo argumento requerimos que $I_Z(a, b|\langle x, y\rangle) \leq t$. Se observó que este valor oscilaba en este caso entre 44 y 46.

Estas observaciones muestran que no existe un valor de t tal que todos los esqueletos de los grafos dirigidos sean inferidos simultáneamente con probabilidad cercana a 1, puesto que los valores de información mutua condicional involucrados en unos grafos y otros no son los mismos. Sin embargo, aún así podemos elegir un umbral intermedio y observar los resultados. Así, en la fila central de la Figura 4.12 vemos los resultados con $t = 45$. En ellos se observan frecuencias de aparición de aristas intermedias, ni muy cercanas a 0 ni muy cercanas a 1.

Este análisis ejemplifica la manera en que las aristas inferidas dependen del umbral: una arista (V_i, V_j) es inferida a partir de una colección de strings de entrada $v_1 \dots v_n$ si y solo si $t < t_{ij} := I_Z(v_i : v_j|\langle\{v_k : k \neq i, j\}\rangle)$. Dado que en nuestras simulaciones los strings de entrada son generados de forma pseudoaleatoria, el valor de t_{ij} oscila entre una repetición y otra de la simulación. La frecuencia con la que se infiere la arista (V_i, V_j) es aproximadamente 1 para valores chicos de t, aproximadamente 0 para valores altos de t, y transiciona entre estos dos valores en un entorno del valor medio de t_{ij} entendido como variable aleatoria, cuyo tamaño está dado por la dispersión estándar de dicha variable.

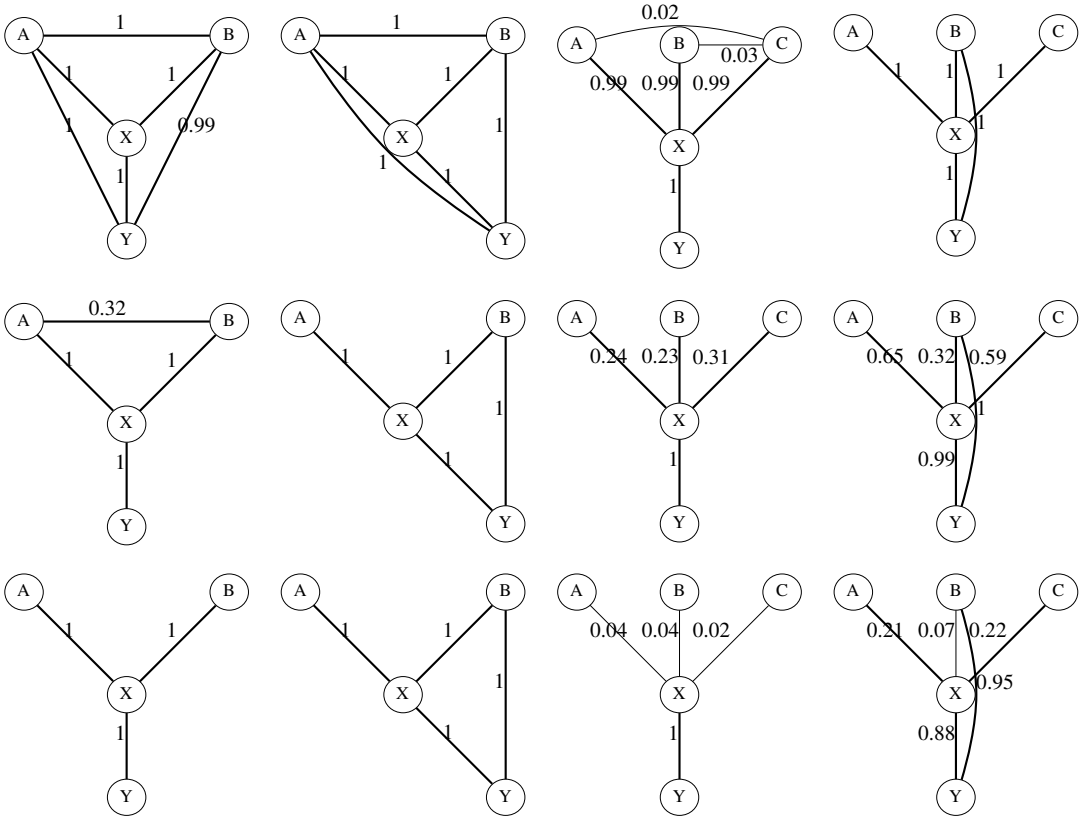


Figura 4.13: Resultados de simulaciones sobre modelos de concatenación errática con shifts (de magnitud 1 en X y 2 en Y), con test de I-complejidad y umbrales $t = 35$ (arriba), 45 (centro), y 50 (abajo) ($n_{\text{repeticiones}} = 100$). De izquierda a derecha, el orden de presentación de los resultados es Y, instrumental, Ψ, y Ψ instrumental. Los resultados son análogos a los observados para los modelos sin shifts correspondientes.

4.4. Discusión

Hemos presentado una serie de simulaciones en las que se aplica la primera fase del algoritmo de descubrimiento causal empleando tests tanto estadísticos como sintácticos sobre strings generados a partir de modelos funcionales sintácticos de tres tipos: XOR, concatenación truncada y concatenación errática. Además, consideramos el caso en que estos modelos son modificados introduciendo shifts en algunos nodos.

Los tests estadísticos permitieron, en la mayoría de los casos, reconstruir el grafo estructural de los modelos para los tres tipos de función estructural, pero no así al introducir los shifts, obteniéndose en ese caso grafos discretos. Esto era de esperar pues los shifts destruyen la correlación posición a posición en los strings, de forma tal que un test que solo puede ver ese tipo de correlación concluye que todos los strings son independientes.

Por el contrario, los tests sintácticos empleados, tanto el basado en gzip como el basado en la I-complejidad, no fueron capaces de reconstruir los grafos estructurales en el caso de las funciones XOR. Esto puede entenderse notando que la correlación sintáctica que estas funciones introducen es extremadamente débil, al menos para el valor de p_1 empleado. Estos tests sí pudieron reconstruir los grafos estructurales correspondientes a los modelos de concatenación, y esto se mantuvo así al introducir shifts. Esto es, nuevamente, razonable, dado que una transformación sencilla como son los shifts no altera significativamente la estructura de los strings.

Sin embargo, si bien los tests sintácticos permitieron recuperar los grafos estructurales, los conjuntos bloqueadores identificados en el proceso no parecen ser los esperados, al menos ingenuamente, en todos los casos, y esto indica que las fases de orientación de aristas del algoritmo (las cuales no fueron estudiadas) probablemente *no* reproduzcan la orientación de las flechas en las estructuras causales generadoras. El caso que evidencia esto de manera más clara es el de los strings generados de manera independiente (strings A y B), los cuales siempre son asignados un conjunto bloqueador no vacío debido a que los valores de umbral que permiten la reconstrucción de los esqueletos no dirigidos son siempre menores al valor de información mutua marginal entre dichos strings. Esto parece estar fuertemente relacionado con el hecho de que nuestras medidas de información, la longitud de compresión por gzip y la I-complejidad, tienden a presentar una disminución significativa en los valores de información mutua a medida que se condiciona sobre una mayor cantidad de strings. Resulta deseable, pues, profundizar en el estudio de este fenómeno. La observación previa sobre el comportamiento de la información mutua al realizar un condicionamiento implica en particular que las medidas de información sin-

táctica contempladas son monótonas en el sentido de [8]. Esto nos condujo a establecer en la Sección 4.3.3 que la arista (V_i, V_i) es inferida a partir de una colección de strings de entrada $v_1 \dots v_n$ si y solo si $t < t_{ij} = I_Z(v_i : v_j | \langle \{v_k : k \neq i, j\} \rangle)$. Esta observación nos permite a futuro analizar para cada modelo causal sintáctico generador de manera directa, sin necesidad de ejecutar el algoritmo completo, si existe o no un umbral tal que la salida del algoritmo coincida con el esqueleto del grafo subyacente. Sencillamente, los valores de umbral tales deben ser menores a t_{ij} para toda arista (V_i, V_j) presente en el grafo generador y mayores a dicha magnitud para todas las aristas que *no* existen en el grafo generador.

4.4.1. Compatibilidad y estabilidad con respecto a nociones de dependencia sintácticas

Para poder entender en mayor profundidad los resultados presentados, una pregunta fundamental es la de si es razonable o no, en cada caso, esperar que el algoritmo devuelva como resultado el esqueleto no dirigido de la estructura causal del modelo sintáctico generador.

En la Sección 4.3.2 observamos que el modelo funcional de concatenación truncada construido sobre el grafo Ψ instrumental presentaba relaciones de dependencia distintas de las implicadas por dicho grafo (según el criterio de d-separación). Por ejemplo, habíamos observado que y no dependía de a puesto que el fragmento de a proveniente de x quedaba totalmente sobrescrito por un fragmento proveniente de b. Esto representaba un ejemplo claro de que el grafo empleado para construir el modelo causal sintáctico generador no necesariamente coincide con la estructura de independencias causales que puede leerse a partir de esos datos.

Por un lado, cabe enfatizar que el hecho de que el algoritmo no infiriera en ese caso, por ejemplo, el grafo Ψ instrumental no es una falla del algoritmo sino una propiedad del modelo causal empleado. En efecto, debemos recordar que la entrada del algoritmo es *un conjunto de datos*, y no los modelos funcionales construidos como generadores de esos datos, ni menos aún la manera concreta de construirlos que se adoptó en este trabajo, combinando libremente grafos con familias de funciones estructurales. En este sentido, el algoritmo es correcto en la medida en que infiere las relaciones causales que pueden ser leídas, ya sea estadística o sintácticamente, a partir de los datos.

Por otro lado, si analizamos por qué estas dos cosas difieren, tiene sentido preguntarnos si las observaciones (strings) que generamos con nuestros modelos son, en un primer lugar, Markov-compatibles con la estructura causal, y en un segundo lugar si son estables

(o monótonamente estables) con respecto a ella. La pregunta sobre la compatibilidad de Markov tiene sentido dado que, a diferencia de lo que ocurría para los modelos causales convencionales y para los modelos causales algorítmicos de [7], no hemos presentado un resultado que garantice que los strings generados por un modelo funcional sintáctico sean Markov-compatibles con su propia estructura causal *cuando observamos las dependencias entre strings a través de tests sintácticos.*

Pero ¿qué quiere decir que unos strings sean Markov-compatibles con un DAG en un sentido sintáctico? A raíz del Teorema 2.2.1, es posible redefinir la compatibilidad de Markov probabilística como una relación entre la lista de independencias condicionales implicada por un DAG (*via* d-separación) y la lista de independencias condicionales satisfechas por una distribución de probabilidad. En base a esta observación, damos una definición abstracta de compatibilidad como una relación entre un DAG y una lista de independencias L obtenidas mediante algún medio a especificar a partir de cierto conjunto de datos. Esta definición complementa la Definición 4.1.3 de estabilidad monótona.

Definición 4.4.1 (Compatibilidad de Markov abstracta)**.** Sea G un DAG con vértices $v_1, \dots v_n$, y sea $I(G)$ el conjunto de relaciones de independencia condicional inducidas por G mediante el criterio de d-separación. Sea L una lista de independencias condicionales entre variables $V_1, \dots V_n$ asociadas a los vértices de G. Entonces G y L son *Markov-compatibles* si y solo si $I(G) \subseteq L$.

Es importante notar que tanto la compatibilidad de Markov como la estabilidad de las observaciones con respecto al grafo generador dependen fuertemente de la noción de independencia condicional contra la cual estamos comparando el criterio de d-separación. En nuestro caso, la lista L proviene de usar uno de los tests sintácticos presentados, con un cierto valor de umbral t, i.e. las afirmaciones en L son de la forma $(U \perp\!\!\!\perp V|W)_{Z,t}$. Es decir que el hecho de si la entrada del algoritmo generada con el modelo M, con estructura causal G, es compatible o estable con respecto al grafo G (y por lo tanto podemos esperar obtener el esqueleto de G a la salida del algoritmo) depende de t.

Es posible que este tipo de modelos funcionales, que hemos llamado modelos funcionales sintácticos, tienda a generar distribuciones probabilísticamente inestables de manera más frecuente que lo que ocurre en otras aplicaciones más tradicionales de la teoría de redes causales. De ser este el caso, podría revisarse la concepción habitual de las distribuciones inestables como "patológicas" o inexistentes en la naturaleza. Tal como se refiere en [7], existe una línea de razonamiento de espíritu Bayesiano que dice lo siguiente: como las distribuciones inestables forman un conjunto de medida cero en el espacio de parámetros, estas no ocurren casi seguramente (i.e. con probabilidad 0) si asumimos

que "la naturaleza elige" los parámetros de la distribución de acuerdo a alguna densidad en el espacio de parámetros. Ahora bien, si los parámetros sobre los que ocurre dicha "elección" no son probabilísticos (kernels de Markov, parametrizados por números reales) sino algorítmicos (e.g. funciones computables sobre strings), entonces cabría imaginar que las distribuciones inestables no sean conjuntos de medida cero en este nuevo espacio de parámetros.

4.4.2. Hipótesis de mecanismos independientes

¿Constituyen los modelos funcionales empleados ejemplos de modelos causales algorítmicos en el sentido de [7]? Más precisamente, dada una realización concreta de los strings, obtenida a partir de la distribución de probabilidad sobre strings generada por un modelo funcional sintáctico, nos preguntamos si ese conjunto de strings es generado por un modelo causal algorítmico cuyas funciones estructurales son esencialmente las mismas que las de nuestro modelo causal sintáctico. El ruido añadido a las funciones estructurales, así como la información referida a cómo se construyen las salidas en las funciones de concatenación errática, pueden ser representados mediante strings de ruido u_j, independientes entre sí y del resto de los strings (en el sentido algorítmico, siempre y cuando ignoremos la correlación algorítmica dada por el generador de números pseudoaleatorios). Ahora bien, se nos presenta una dificultad al considerar la hipótesis de *mecanismos causales independientes*, que en el caso algorítmico consiste en que los programas que computan las funciones f_j sean también algorítmicamente independientes.

Consideremos un ejemplo trivial como en [7]: un grafo discreto con dos nodos y ninguna arista. En este caso, cada string es computado por algún programa, usando opcionalmente un string de ruido como entrada. Si los programas que computan los dos strings son en verdad un único programa, los dos strings pueden ser extremadamente similares (o idénticos, si el programa ignora los strings de ruido) y la información mutua algorítmica entre ambos es máxima. Es claro que se debe evitar este tipo de situaciones para que la estructura causal pueda ser capturada mediante correlaciones algorítmicas.

En principio, en nuestro caso *no* se cumple la hipótesis dado que las funciones fueron elegidas a partir de familias paramétricas. Consideremos dos nodos, j_1 y j_2, con m_1 y m_2 padres respectivamente. Asumamos que estos nodos se encuentran en componentes conexas distintas del DAG, o más en general que están d-separados por el conjunto vacío. Supongamos que las funciones asociadas f_{j_1} y f_{j_2} pertenecen a una familia de funciones computables $\mathcal{C}$ paramétrica en la aridad m. Entonces, ambas funciones pueden especificarse por un programa que computa la función correspondiente dada su aridad,

junto con la aridad m_1 o m_2 según corresponda. Esto muestra que las descripciones algorítmicas de ambas funciones son esencialmente idénticas. ¿Implica esto que los strings x_{j_1} y x_{j_2} están correlacionados algorítmicamente? Razonando informalmente, podemos ver que para generar a cada string por separado, debemos especificar a cada uno de sus strings padre (incluyendo el ruido), la familia paramétrica $\mathcal{C}$ y la aridad de la función a aplicar, mientras que si deseamos generar a ambos strings simultáneamente podemos ahorrarnos una "copia" de la especificación de la familia paramétrica. Esto sugiere que la información mutua entre los dos strings es no nula, y es del orden de la complejidad de descripción algorítmica de la familia paramétrica $\mathcal{C}$ [21].

Sin embargo, es importante considerar no solamente la independencia algorítmica entre las funciones en distintos nodos sino también su complejidad. Parafraseando a Janzing y Schölkopf, si observamos que dos strings son generados a partir de sus padres usando una misma regla *compleja*, el principio de mecanismos independientes nos llevaría a postular un nuevo camino entre los strings en el DAG que explique la similaridad de los mecanismos. Ahora bien, si los mecanismos son *simples*, por más que sean similares, podemos argumentar que puede no ser necesaria dicha modificación.

Más aún, incluso en la medida en que esta estructura compartida entre x_{j_1} y x_{j_2} sea significativa, no necesariamente es estructura que pueda ser detectada por las medidas de información utilizadas en nuestro trabajo. Este parece ser efectivamente el caso. La definición de modelo funcional dada en [8], la cual es relativa a una medida de información general, presenta una posible vía para caracterizar esta relación de adecuación (o no adecuación) entre medidas de información y funciones estructurales empleadas.

4.4.3. Aplicación a situaciones de inferencia

Finalmente, abordemos la pregunta sobre cómo utilizar el método desarrollado en una situación de inferencia genuina, en la que lo único que se posee son las observaciones constituidas por n strings. Dado que se observó una menor variabilidad en los resultados al emplear la I-complejidad con respecto a gzip, y dado que en una situación de inferencia real solo se posee una única realización, nuestros resultados sugieren que sería más conveniente utilizar la I-complejidad. En segundo lugar, para determinar el umbral a utilizar puede computarse para cada par de strings el valor de información mutua entre ellos, dados los demás strings. Esto determinaría la región de valores de umbral que tiene

[21] Esto es tan solo un argumento intuitivo dado que nada garantiza que esa sea la descripción *más corta* posible de los strings (afirmar eso sería afirmar la versión recursiva de la condición de Markov algorítmica). Por ejemplo, las funciones podrían ignorar algunos de sus argumentos y en ese caso la descripción de los mismos sería supérflua.

sentido utilizar, i.e. que no darán lugar a grafos completos ni discretos. Finalmente, el uso de tests sintácticos no implica que no puedan ser utilizados simultáneamente otros tests estadísticos tales como el utilizado en nuestro caso o incluso otros. Las observaciones con los modelos de tipo XOR sugieren que ciertas clases de procesos subyacentes pueden ser fácilmente detectables por los tests estadísticos más sencillos pero ser muy difíciles de identificar mediante tests sintácticos, debido a que la correlación que introducen entre entradas y salidas a lo largo de múltiples posiciones de cada string puede ser débil.

Capítulo 5

Perspectivas y Conclusiones

En este capítulo comentamos en primer lugar una serie de posibilidades de exploración que surgen en torno al núcleo del trabajo realizado. A continuación mencionamos ciertas ideas referidas al estudio en profundidad de las medidas de complejidad. En tercer lugar, mencionamos ciertas áreas en las que sería deseable aplicar los resultados de esta línea de investigación. Finalmente, enunciamos algunas conclusiones a partir del trabajo realizado.

5.1. Perspectivas y trabajo a futuro

Las funciones estructurales empleadas y la aplicación de los shifts constituyen tan solo un primer paso o una prueba de concepto de que la implementación reproduce un comportamiento similar al esperado. Desde este punto de vista, es necesario contemplar otros tipos de funciones estructurales y de transformaciones sintácticas de strings que permitan explorar el alcance del algoritmo y, fundamentalmente, el alcance de las medidas de complejidad sintácticas usadas en dicho algoritmo.[1]

En este sentido, resulta interesante considerar a la tarea de descubrimiento causal como un estándar mediante el cual evaluar el poder de distintas aproximaciones a la información algorítmica, juzgando cuándo una cierta medida de complejidad es adecuada para analizar strings provenientes de una cierta clase de procesos causales, y cuándo no lo es.

Por otro lado, sin necesidad de explorar otros modelos generadores, quedan abiertas ciertas líneas de trabajo cuyo objetivo es entender con mayor profundidad el comportamiento del algoritmo sobre el tipo de entradas que ya han sido consideradas en este

[1] También sería deseable, por supuesto, realizar simulaciones con una mayor variedad de estructuras causales.

trabajo.

Desde el lado de la implementación, los resultados obtenidos sugieren que sería deseable tener una forma más sistemática de comprender la dependencia de los mismos con el umbral. Para esto, resultaría conveniente realizar simulaciones en las que se computen todos los valores de información mutua condicional posibles, para poder luego estudiar cómo varía la lista de independencias condicionales en función del umbral del test. Simultáneamente, esto permitiría estudiar nuevos criterios para concluir la presencia de un vínculo causal que utilicen los resultados del algoritmo para múltiples valores de umbral. Más aún, pueden concebirse modificaciones del algoritmo en sí que contemplen esta información. Una idea más o menos inmediata, si bien poco precisa, es la de reemplazar por completo los juicios de independencia condicional por una alternativa *difusa*, en la que a cada potencial conjunto bloqueador S_{ab} para los nodos a, b se le asigne un *score* dado precisamente por la información mutua condicional $I_Z(a : b|S_{ab})$.

Volviendo a la idea de entender en mayor profundidad los resultados ya obtenidos, un asunto pendiente es estudiar de forma teórica las causas de que el algoritmo reconstruya las estructuras causales únicamente usando valores de umbral "demasiado bajos", i.e. por debajo del nivel de independencia mutua marginal, dando lugar a que strings que desearíamos juzgar marginalmente independientes sean asignados conjuntos bloqueadores no vacíos. Esto resulta esencial si se desea avanzar hacia la implementación con tests sintácticos de las fases de orientación de aristas del algoritmo original, las cuales fallarán si los conjuntos bloqueadores no son los correctos.

Por otro lado, el hecho de que nuestro método utiliza medidas de complejidad sintáctica abre la posibilidad a incorporar nuevos criterios para la orientación de aristas, los cuales podrían reemplazar a las fases de orientación de aristas en el algoritmo original o bien aplicarse con posterioridad a las mismas, de manera tal de distinguir entre estructuras causales Markov-equivalentes[2], cosa a la cual se apunta en [7]. Por ejemplo, podría considerarse como hipótesis causal que la complejidad sintáctica aumenta en la dirección de la causalidad, o bien en la dirección opuesta. Estas hipótesis estarían sustentadas en observaciones previas, específicas de cada dominio de aplicación, sobre si los procesos físicos involucrados aumentan o reducen la información contenida en los sistemas.

Tal como se mencionó en la Sección 4.4.2, una posible línea de trabajo teórico consiste en buscar caracterizar la relación necesaria entre funciones estructurales y medidas de información que permitiría garantizar la validez de la hipótesis de independencia de los mecanismos y la propiedad de que las observaciones generadas sean Markov-compatibles

[2]Recordemos que la equivalencia entre estructuras causales puede expresarse en términos del criterio de d-separación puramente.

y monótonamente estables con respecto al DAG generador. En [8] se presentan ciertas herramientas que podrían ser útiles en esta dirección pero no se presta atención, por el contrario, a la cuestión de la dependencia de la compatibilidad y estabilidad con respecto al umbral.

Finalmente, una línea que se eligió no desarrollar durante el presente trabajo consiste en implementar una variante del algoritmo que permita inferir modelos causales en los que las funciones estructurales son completamente determinísticas. Para esto, debe reemplazarse el criterio de d-separación empleado en este trabajo por un criterio modificado, presentado en [24] bajo el nombre de *D-separación*. Así, se evidenciaría aún más la manera en que el concepto de correlación sintáctica extiende la inferencia causal más allá de la teoría probabilística.

5.1.1. Exploración de complejidades computables

Definir la complejidad condicional según $Z(x|y) = Z(x, y) - Z(y)$ para una medida de información computable Z genérica dista de capturar la esencia de la información algorítmica condicional. Como se mencionó previamente, un programa minimal p para x dado y (cuya longitud determina el valor de $K(x|y)$) puede ser tal que interpreta a y mismo como programa, con la total generalidad que eso implica (nada impide que dicho programa minimal incluya a un intérprete universal, para ejecutar y con alguna entrada z).

Por ejemplo, si y es un programa que realiza operaciones sobre grafos, interpretando el string binario de input como grafo a través de una codificación particular (por ejemplo, y computa la función que dado un grafo G produce el grafo producto $G \times G$) y x codifica un grafo que puede ser obtenido a partir de otro más simple, z, mediante aplicación de y, entonces un programa para x a partir de y podría ser "ejecutar la entrada pasándole como argumento z". De esta consideración, se sigue lo siguiente.

Proposición 5.1.1. *Sea y un programa que computa una función de grafos $f : \Sigma^* \to \Sigma^*$ que produce grafos más grandes a partir de grafos más chicos. Asumimos una codificación tal que todo $x \in \Sigma^*$ codifica un grafo válido y grafos más grandes tienen codificaciones más largas. Entonces*

$$K(x|y) \leq |f^{-1}(x)| + c \tag{5.1}$$

donde f^{-1} es una inversa por derecha de f (i.e. $f \circ f^{-1} = 1_{\Sigma^*}$) y c_y es una constante independiente de x y de y.

Más en general, vemos que condicionar sobre un string y puede acotar las compleji-

dades condicionales de todos los strings por las longitudes de sus *preimágenes* a través de la función computada por y. Esto no implica que los programas minimales vayan a ser efectivamente de esta forma, pero es algo que muy distinto de lo que ocurre en el caso de compresores.

Observaciones de este tipo nos invitan a pensar en compresores modificados que pueden tomar funciones auxiliares como entrada. En la medida en que ciertas funciones permitan comprimir más eficientemente cierto tipo de entradas, esto ofrecería también un marco general para definir compresores modificados ajustados al tipo de mecanismos causales presentes en un cierto dominio de aplicación. Este tipo de modificaciones han sido empleadas, por ejemplo, para el estudio de secuencias genéticas [31].

Por otro lado, una idea cercana es la de explorar nociones de complejidad condicional *intrínseca*, no definida según $Z(x|y) = Z(x,y) - Z(y)$ sino que para computar $Z(x|y)$ empleen un mecanismo intrínsecamente diferente que para computar $Z(x)$ y $Z(y)$ (a diferencia del caso anterior, seguimos asumiendo que y es un string del mismo tipo que x y no un objeto de otra índole). En esta línea resulta interesante estudiar la solución adoptada en [11]. En este trabajo, para definir la distancia de compresión normalizada *condicional* se definen compresores condicionales. Estos compresores, para comprimir x dado y, operan en dos fases: primero construyen un modelo sobre el string y y luego emplean ese modelo para comprimir x.

Este tipo de consideraciones motivó la idea de estudiar una complejidad de descripción "minimalista" basada en codificación de Huffman que pueda ser "aumentada" mediante el uso de funciones auxiliares. A continuación presentamos comienzo de exploración de esta idea.

Complejidad de diccionario

Si nos interesa caracterizar qué es lo que hacen, en esencia, los compresores convencionales pero sin preocuparnos por la cantidad de recursos computacionales necesarios para efectuar una compresión, un camino posible y sencillo es el de definir una complejidad de descripción elemental usando codificación de Huffman.

Primero fijemos un poco de notación. Sea $Part(s) \subseteq \Sigma^{+*}$ el *conjunto de particiones* de s: una partición Π de s es una cadena de strings no vacíos $(s_1, \ldots s_K)$ tales que su concatenación $s_1 \cdot \ldots \cdot s_K$ es igual a s. Vamos a identificar cada partición con una codificación de Huffman de la misma, de forma tal que a cada string s_j que aparece en la partición se le asigna un código autodelimitante c_j cuya longitud es menor mientras más veces aparece s_j en Π; aproximadamente $|c_j| \simeq -\log(\frac{n_j}{K})$ donde n_j es el número de

veces que s_j aparece en Π y $K = |\Pi|$ es el número de elementos de la partición (contando repeticiones).

Esta asignación de strings que aparecen en la partición a códigos constituye un *diccionario* D_Π. Podemos preguntarnos por el string resultante de reemplazar cada aparición de un s_k en el string original s por su código c_k, obteniendo así un string más corto. Llamemos $D_\Pi(s)$ al string que resulta de este proceso.

Si definimos alguna noción adecuada de *tamaño* de un diccionario, que notamos $t(D_\Pi)$, podemos definir una complejidad de descripción de diccionario según

$$\mathcal{D}(s) := \min_{\Pi \in Part(s)} (t(D_\Pi) + |D_\Pi(s)|) \tag{5.2}$$

Es decir, la complejidad de descripción de s es la suma de la longitud de su versión comprimida sumada a la longitud del diccionario que usamos para comprimir, optimizada sobre todas las particiones posibles.

Esta función es simple matemáticamente y es computable. Tiene algunas propiedades interesantes como por ejemplo el hecho de que es claramente invariante ante permutaciones de los símbolos del alfabeto o ante reversión del input. Esto último no es cierto para la mayoría de los compresores comerciales dado que estos procesan el string de izquierda a derecha.

Intentar describir matemáticamente un compresor convencional idealizado cumple dos funciones: por un lado, (1) explorar qué tan poderoso puede ser un procesamiento de los datos que siga siendo en algún punto "puramente estadístico" (ver más abajo) y, por el otro, (2) explorar el mundo de las complejidades computables con nociones de complejidad condicional intrínseca al aumentar esta definición matemática con la posibilidad de usar *funciones auxiliares*, que permitan comprimir más aún los datos de entrada.

Expandimos ahora el primero de estos puntos. Utilizaremos la noción matemática de normalidad. La idea es que una secuencia es normal si, mirando prefijos cada vez más largos, las frecuencias de aparición de bloques de una misma longitud tiende a una distribución equiprobable.

Definición 5.1.1 (Normalidad)**.** Sea $x \in \Sigma^\infty$, $x = x_1 x_2 x_3 \ldots$ una secuencia infinita de símbolos y $x^k = x_1 \ldots x_k$ su prefijo de longitud k. Decimos que x es *normal* si, para cada $n, k \in \mathbb{N}$, la frecuencia de aparición en x^k de cada string de longitud n tiende a $|\Sigma|^{-n}$ cuando k tiende a ∞. En otras palabras, si asignamos una distribución de probabilidad al conjunto Σ^n según la frecuencia de aparición de cada elemento en x en el límite de prefijos cada vez más grandes, esa distribución resulta ser uniforme.

Informalmente, podemos decir que un string (es decir una cadena *finita*) $x \in \Sigma^*$ es "aproximadamente normal" si todos los bloques de símbolos de longitud fija aparecen "aproximadamente" con la misma frecuencia.

Si un tipo de procesamiento de la entrada no puede comprimir ninguna secuencia normal, entonces podemos decir provisionalmente que es *de tipo estadístico*, en contraposición con, como caso extremo, la información algorítmica, que puede detectar estructura algorítmica incluso en (algunas) secuencias normales. Es decir que la compresión estadística sería aquella que permite aprovechar desviaciones respecto de la normalidad, pero no otros tipos de regularidades.

Es importante no confundir este uso de la palabra "estadístico" con el que le dimos al hablar de tests estadísticos; en este caso siempre nos referimos a aspectos sintácticos de strings, pero estableciendo dos clases de métodos de compresión según su potencia. En la clase de compresores estadísticos se encontrarían por ejemplo los compresores convencionales, pero también la complejidad de diccionario $\mathcal{D}$, mientras que en la clase de compresores algorítmicos propiamente dichos se encontraría el compresor idealizado (no efectivo) que asigna a cada string x un programa minimal x^*.

Consideramos ahora el segundo aspecto mencionado. ¿Qué querría decir *aumentar* a $\mathcal{D}$, de forma tal que pueda utilizar algún tipo de ayuda al comprimir entradas?

La idea explorada fue la de permitir como entrada al proceso de compresión un conjunto de *funciones auxiliares* $\mathcal{F}$, y considerar no solo particiones "puras" del string de entrada, como las que ya fueron descritas, sino también particiones que contienen no solo substrings del string original sino también referencias a las funciones auxiliares y cómo deben ser aplicadas. Obtenemos así una complejidad de descripción $\mathcal{D}_\mathcal{F}$ de la cual recuperamos el caso no aumentado tomando $\mathcal{F} = \emptyset$. Esto extiende la métrica de complejidad tanto hacia un terreno más allá del puramente estadístico (dado que dependiendo del conjunto de funciones $\mathcal{F}$, sería posible comprimir ciertas secuencias normales) como hacia el terreno de las nociones de complejidad condicional intrínseca.

Definición 5.1.2 (Complejidad de diccionario aumentada)**.** Sea $s \in \Sigma^*$ y $\mathcal{F}$ un conjunto finito y ordenado de funciones auxiliares. Una cadena finita $\Pi \in (\Sigma^+ \cup \mathbb{N}^2)^*$, cuyos elementos son o bien strings o bien pares de números naturales de la forma (i, j), es una *partición aumentada* de s sii al reemplazar cada tupla (i, j) por el string $f_j(x_i^{lit})$, donde f_j es la j-ésima función auxiliar y x_i^{lit} es el i-ésimo elemento de x, de izquierda a derecha, que sea un *literal* (es decir, un string y no una tupla de índices), y concatenar todos los strings resultantes, se obtiene como resultado s.

Llamamos $PartAug(s)$ al conjunto de particiones aumentadas de s.

Dada una partición aumentada $\Pi \in PartAug(s)$, el diccionario ordenado D_Π asociado a Π se define como una cadena de pares clave-valor donde las claves son los elementos de Π y los valores guardados son sus códigos de Huffman correspondientes. El tamaño del diccionario, $t(D_\Pi)$, se define como la suma de las longitudes de todas las claves y valores. La longitud de las claves de la forma (i, j) (referencia a una clave literal i y a la función auxiliar j) es constantemente igual a $\log|\mathcal{F}| + \log|\Pi|$.

Con estas definiciones, la *complejidad de diccionario aumentada según $\mathcal{F}$ de s* se define como

$$\mathcal{D}_\mathcal{F}(s) := \min_{\Pi \in PartAug(s)} \left(t(D_\Pi) + |D_\Pi(s)|\right). \tag{5.3}$$

Podemos interpretar las funciones auxiliares $\mathcal{F}$ como una formalización del conocimiento de *background* que se tiene sobre la clase de objetos cuya complejidad de descripción se desea estudiar. Por ejemplo, funciones sencillas como $\text{rev} : x \mapsto x^r$ que espejan un string podrían ser utilizadas cuando se sabe que los procesos que relacionan un string con otro implican la reversión de substrings de alguno de los strings. Por otro lado, al no restringir la clase de funciones que pueden ser incluidas (salvo tal vez a que sean funciones computables), es posible elegirlas de manera *ad hoc* para lograr que ciertos strings sean arbitrariamente simples. En este sentido la elección de $\mathcal{F}$ siempre deberá estar fuertemente informada por el objetivo que se persigue en cada aplicación.

* * *

Una pregunta teórica interesante para el presente trabajo es si la complejidad de diccionario $\mathcal{D}$ satisface las propiedades de una "medida de información" tal como son enunciadas en [8], pues en tal caso la relación de independencia condicional basada en la información mutua asociada satisfaría los axiomas de semi-grafoide (ver Sección 2.1) y por lo tanto valdría la equivalencia entre la expresión local y la expresión global de la condición de Markov (Teorema 1 en [8]). Esto sugeriría que el algoritmo de descubrimiento modificado para emplear la complejidad de diccionario es *correcto* para ciertas clases de modelos funcionales (los que satisfagan la Definición 4 de [8]).

Cabe mencionar que la compresión mediante diccionarios es un método de compresión relativamente débil. Un ejemplo de un método de compresión que es por lo menos tan bueno como este (o mejor) es utilizando *gramáticas*. Sin entrar en detalle, las gramáticas formales consisten en conjuntos de reglas sustitutivas que permiten transformar un string en otros strings posiblemente más complejos. Por ejemplo, una gramática que posee la regla $AA \to BCBBC$ permite transformar el string inicial (o "semilla") $AAAAA$

en $BCBBCABCBBC$. Así, es posible comprimir un string x indicando una semilla $\tilde{x}$ así como un conjunto de reglas que permita producir *unívocamente* (es decir, sin arbitrariedades) a x a partir de $\tilde{x}$. La complejidad de descripción basada en gramáticas resulta ser una suma de las longitud de $\tilde{x}$ y la longitud asociada a las reglas.

Si podemos representar el string x mediante un diccionario con K entradas de la forma (s_k, c_k) (donde la clave s_k es una subpalabra que aparece n_k veces en x y c_k es el código de Huffman que se le asigna), entonces también puedo describir s como la única palabra descrita por la gramática que tiene una regla de producción $c_k \to s_k$ por cada entrada del diccionario y una regla adicional $A_0 \to c_{k_1} c_{k_2} \dots c_{k_N}$. Si dejamos fijo a A_0 como símbolo "semillaz no lo contamos en el tamaño de la gramática, y contamos el tamaño de la gramática como la suma de las longitudes de las reglas de producción (longitud del lado izquierdo más longitud del lado derecho), entonces el tamaño obtenido para esa gramática es exactamente el mismo que la longitud de compresión de x usando ese diccionario. Esto establece que una descripción de x mediante un diccionario de Huffman es un caso particular de una descripción de x mediante una gramática.

Por otro lado, intuitivamente la compresión gramática es mejor en general que la compresión de diccionario puesto que es posible agregar reglas de producción intermedias, que podrían pensarse como aplicar iterativamente la compresión de diccionario.

5.1.2. Aplicaciones

En 4.1 se mencionaron trabajos en los que se aplican métricas de distancia [10] y distancia condicional [11] al análisis de secuencias genéticas. Esto resulta una aplicación natural del tipo de ideas presentadas en este trabajo dado que, hablando de manera metafórica, las secuencias genéticas son el caso en que "la Naturaleza nos muestra" información literalmente codificada en términos de strings, específicamente sobre un alfabeto $\Sigma = \{\mathrm{C}, \mathrm{G}, \mathrm{A}, \mathrm{T}\}$ para el caso del ADN.

Por otro lado, resulta de interés la aplicación de estas herramientas a la detección de relaciones causales entre resultados de mediciones en experimentos tipo Bell que buscan estudiar fenómenos de no-localidad cuántica [32, 33]. En su versión más simple, estos experimentos involucran dos qubits en un estado entrelazado y mediciones realizadas sobre ellos por dos agentes, Alice y Bob. Al repetirse el experimento múltiples veces, los resultados de las mediciones de Alice, así como de las de Bob pueden expresarse como una secuencia binaria. El uso de técnicas de detección de correlaciones algorítmicas podría permitir establecer o descartar vínculos causales que pasarían desapercibidos o serían más difíciles de detectar mediante análisis puramente estadísticos.

En lo referido a las aplicaciones en información cuántica, las redes causales han resultado ser un lenguaje natural para describir escenarios de Bell, especialmente escenarios con estructuras causales más complejas en los cuales un sistema entrelazado se distribuye entre múltiples partes, y para expresar las consecuencias de la presencia o ausencia de distintas variables ocultas [32, 33]. Por otro lado, el estudio de la no-localidad cuántica está íntimamente ligado al concepto de *aleatoriedad* [34]. Como se mencionó al pasar en el Capítulo 3, la aleatoriedad de una secuencia de símbolos x está asociada a su impredecibilidad y a la ausencia de patrones en la misma; esto último puede expresarse como el hecho de que $K(x)$ es aproximadamente igual a la longitud de x (al no haber patrones en la secuencia, no hay manera de comprimirla). Esta noción de aleatoriedad permite distinguir las secuencias de inputs a un experimento que son verdaderamente aleatorias de las secuencias meramente pseudoaleatorias, con consecuencias medibles significativas e incluyendo posibles *loopholes* en experimentos de Bell [35, 36]. Por lo tanto, no es sorprendente que la información algorítmica juegue un rol importante en el estudio de la no-localidad. Finalmente, puede mencionarse el trabajo [37] de Bendersky et al. en el cual emplean nociones débiles de aleatoriedad para estudiar el fenómeno de no-localidad, según las cuales una secuencia es débilmente aleatoria si se ve como si fuera aleatoria para cualquier adversario con un poder de cómputo suficientemente limitado. Estas nociones guardan cierta relación con el trabajo realizado, dado que aproximan la noción de aleatoriedad absoluta introduciendo limitaciones en el poder de cómputo de los agentes, de la misma forma que en nuestro trabajo utilizamos complejidades computables, las cuales pueden entenderse como variantes débiles de (o aproximaciones a) la información algorítmica.

5.2. Conclusiones

De este modo culmina la presentación del trabajo. A modo de recapitulación, recordemos que el objetivo general de esta tesis fue el de combinar dos teorías, la teoría de modelos causales y la Teoría Algorítmica de la Información, incorporando la correlación sintáctica entre objetos codificados simbólicamente como una medida de dependencia condicional en base a la cual inferir relaciones causales. Esto implicó cambiar el dominio de discurso, reemplazando las variables aleatorias por strings en cuanto a objetos de estudio, así como considerar nuevas clases de *procesos* que conectan esos objetos (funciones de strings), los cuales surgen de modo natural en este nuevo contexto.

Este objetivo fue abordado concretamente a través de una tarea de inferencia causal específica: el descubrimiento causal. Así, se desarrollaron tests de independencia condi-

cional, uno estadístico y una familia de tests sintácticos, y se implementó un algoritmo basado en vínculos para el descubrimiento de esqueletos causales no dirigidos, permitiendo su operación con dichos tests. Se eligió una familia de modelos causales sintácticos, y con ellos se realizaron simulaciones, generando datos y aplicando sobre los mismos las distintas variantes del método de inferencia. De esta forma se identificó una clase de modelos cuya estructura causal no dirigida pudo ser inferida empleando tests estadísticos, mas no con tests sintácticos (modelos tipo XOR); una clase de modelos cuya estructura pudo ser reconstruida mediante ambos tipos de test (modelos de concatenación) y otra cuya estructura solo pudo ser reconstruida mediante tests sintácticos (modelos de concatenación con shifts).

Los resultados obtenidos sugieren que efectivamente existe un dominio de aplicabilidad para la inferencia causal basada en correlación sintáctica, el cual no necesariamente coincide con el de otros métodos. Sin embargo esto constituye tan solo un paso en la dirección del objetivo general planteado. Muchas líneas de exploración, tanto computacionales como teóricas, quedan abiertas a partir de estas primeras observaciones. Por ejemplo, respecto de los modelos reconstruidos con tests sintácticos, se observó que los conjuntos bloqueadores identificados por el algoritmo no siempre coinciden con los que se deducen de la estructura causal del modelo, lo cual implica que las fases de orientación de aristas del algoritmo fallarán y amerita un estudio más profundo. Desde un enfoque teórico, resulta prometedora la dirección de estudiar la compatibilidad entre clases de procesos causales y aproximaciones computables a la información algorítmica. En este sentido la teoría de la causalidad podría encontrar un lugar natural dentro del estudio de la complejidad de descripción, como un contexto que define una noción de correctitud para las aproximaciones a la información algorítmica.

Bibliografía

[1] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[2] Judea Pearl. *Causality*. Cambridge university press, 2009.

[3] Judea Pearl. On the testability of causal models with latent and instrumental variables. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, páginas 435–443, 1995.

[4] Peter Spirtes. Graphical models, causal inference, and econometric models. *Journal of Economic Methodology*, 12(1):3–34, 2005.

[5] A Philip Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(1):1–15, 1979.

[6] Judea Pearl y Azaria Paz. *Graphoids: A graph-based logic for reasoning about relevance relations*. University of California (Los Angeles). Computer Science Department, 1985.

[7] Dominik Janzing y Bernhard Schölkopf. Causal inference using the algorithmic markov condition. *IEEE Transactions on Information Theory*, 56(10):5168–5194, 2010.

[8] Bastian Steudel, Dominik Janzing, y Bernhard Schölkopf. Causal markov condition for submodular information measures. *arXiv preprint arXiv:1002.4020*, 2010.

[9] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

[10] Rudi Cilibrasi y Paul MB Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[11] Diogo Pratas y Armando J Pinho. A conditional compression distance that unveils insights of the genomic evolution. *arXiv preprint arXiv:1401.4134*, 2014.

[12] Peter D. Grünwald. *The minimum description length principle*. MIT press, 2007.

[13] Hector Zenil, Santiago Hernández-Orozco, Narsis A Kiani, Fernando Soler-Toscano, Antonio Rueda-Toicen, y Jesper Tegnér. A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. *Entropy*, 20(8):605, 2018.

[14] Peter Bloem, Francisco Mota, Steven de Rooij, Luis Antunes, y Pieter Adriaans. A safe approximation for kolmogorov complexity. In *International conference on algorithmic learning theory*, páginas 336–350. Springer, 2014.

[15] Kamaludin Dingle, Chico Q Camargo, y Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature communications*, 9(1):1–7, 2018.

[16] Marek J Druzdzel y Herbert A Simon. Causality in bayesian belief networks. In *Uncertainty in Artificial Intelligence*, páginas 3–11. Elsevier, 1993.

[17] Thomas Verma y Judea Pearl. Equivalence and synthesis of causal models. In *Sixth Conference on Uncertainty in Artificial Intelligence*, páginas 220–227, 1990.

[18] Peter Spirtes y Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

[19] Martin Davis, Ron Sigal, y Elaine J Weyuker. *Computability, complexity, and languages: fundamentals of theoretical computer science*. Elsevier, 1994.

[20] Damien Woods y Turlough Neary. The complexity of small universal turing machines: A survey. *Theoretical Computer Science*, 410(4-5):443–450, 2009.

[21] Jeremy Kun. Kolmogorov complexity - a primer. https://jeremykun.com/2012/04/21/kolmogorov-complexity-a-primer/. Fecha de acceso: 2020-04-02.

[22] Peter Grunwald y Paul Vitányi. Shannon information and kolmogorov complexity. *arXiv preprint cs/0410002*, 2004.

[23] Abraham Lempel y Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.

[24] Dan Geiger, Thomas Verma, y Judea Pearl. Identifying independence in bayesian networks. *Networks*, 20(5):507–534, 1990.

[25] Verónica Becher y Pablo Ariel Heiber. A linearly computable measure of string complexity. *Theoretical Computer Science*, 438:62–73, 2012.

[26] Nithin Nagaraj, Karthi Balasubramanian, y Sutirth Dey. A new complexity measure for time series analysis and classification. *The European Physical Journal Special Topics*, 222(3):847–860, 2013.

[27] Cristian S Calude, Kai Salomaa, y Tania K Roblot. Finite state complexity. *Theoretical Computer Science*, 412(41):5668–5677, 2011.

[28] Jacob Ziv y Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[29] Peter Deutsch. Rfc1951: Deflate compressed data format specification version 1.3, 1996.

[30] Greg Ver Steeg, Rob Brekelmans, Hrayr Harutyunyan, y Aram Galstyan. Disentangled representations via synergy minimization. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, páginas 180–187. IEEE, 2017.

[31] Xin Chen, Sam Kwong, y Ming Li. A compression algorithm for dna sequences and its applications in genome comparison. *Genome informatics*, 10:51–61, 1999.

[32] Rafael Chaves. Polynomial bell inequalities. *Physical review letters*, 116(1):010402, 2016.

[33] Rafael Chaves, Daniel Cavalcanti, y Leandro Aolita. Causal hierarchy of multipartite bell nonlocality. *Quantum*, 1:23, 2017.

[34] Gabriel Senno, Ariel Bendersky, y Santiago Figueira. Randomness and non-locality. *Fluctuation and Noise Letters*, 15(03):1640005, 2016.

[35] Ariel Bendersky, Gonzalo De La Torre, Gabriel Senno, Santiago Figueira, y Antonio Acín. Algorithmic pseudorandomness in quantum setups. *Physical review letters*, 116(23):230402, 2016.

[36] Ignacio H López Grande, Gabriel Senno, Gonzalo De La Torre, Miguel A Larotonda, Ariel Bendersky, Santiago Figueira, y Antonio Acín. Distinguishing computable mixtures of quantum states. *Physical Review A*, 97(5):052306, 2018.

[37] Ariel Bendersky, Gabriel Senno, Gonzalo De La Torre, Santiago Figueira, y Antonio Acin. Nonsignaling deterministic models for nonlocal correlations have to be uncomputable. *Physical review letters*, 118(13):130401, 2017.