Gramáticas Mínimas y Descubrimiento de Patrones

Rafael Carrascosa

18 de marzo de 2010

Agradecimientos

Primero que a nadie me gustaría agradecer a Gabriel Infante-Lopez por dirigir y darme apoyo en este trabajo, por aceptarme e involucrarme en el grupo de lenguaje natural, y por guiarme e instruirme en las áreas de computación que me cautivan: el procesamiento de lenguaje natual, el aprendizaje automático y la inteligencia artificial.

Por permitirme participar de su investigación y por sus invalorables aportes e ideas quiero agradecer a François Coste y a Matthias Gallé, a este último tambien querría agradecer por el gran compañerismo que me mostró durante la escritura de la tesis y por siempre estar dispuesto a dar una mano (atlántico de por medio).

También quiero agradecer a los docentes de computación de FaMAF, quienes nunca se limitaron a sólo cumplir con una función docente sino que se interesaron por las percepciones y las personas que son sus alumnos y muy apasionadamente compartieron con ellos los detalles y pequeñas perlas de nuestra área del conocimiento.

Entre estos docentes, me gustaría agradecer especialmente a Pedro R. D'Argenio y a Laura Alonso i Alemany a quienes aprecio muchísimo puesto que siento que han apadrinado y definido mi percepción de la computación.

Finalmente, y por ello más importante, quiero agradecer a mis padres Lucía y Raul por la vida que me han dado y dedicarle este trabajo a ellos.

Índice general

Capítulo 1

Introducción

Este trabajo se centra en la relación entre dos problemas distintos pero relacionados del área de computación. Por un lado el de la inferencia de gramáticas libres de contexto de tamaño mínimo y por otro, el de cómo se pueden usar estas para el descubrimiento de patrones.

Fue llevado a cabo en el marco de una investigación de la cual forman parte Gabriel Infante-Lopez, François Coste[1], Matthias Gallé[1], y Rafael Carrascosa. Parte de este trabajo fue incluido en un artículo[5] escrito por las cuatro personas mencionadas que fue aceptado para ser publicado en la conferencia LATA 2010[2] a realizarse en Trier, Alemania en mayo de 2010.

La formalización que llevó a las gramáticas libres de contexto fue desarollada por Noam Chomsky [7] a mediados de los años cincuenta como un intento de formalizar matemáticamente la estructura gramatical del lenguaje natural. Estas cobraron rápidamente importancia en ciencias de la computación empujando el desarrollo del estudio en lenguajes formales y en procesamiento de lenguaje natural. A modo de ejemplo, sabemos que la oración en castellano *"El perro mira el auto"* se compone de un sujeto y un predicado, a su vez el sujeto se compone de un artículo y un sustantivo y el predicado se compone de un verbo y un objeto directo. A su vez, el objeto directo es un artículo y un sustantivo. Estas reglas gramáticales pueden ser

[1] Symbiose Project, IRISA/INRIA Rennes-Bretagne Atlantique, Francia
[2] http://grammars.grlmc.com/LATA2010

expresadas formalmente de la siguiente forma:

$$\begin{aligned}
\text{oración} &\rightarrow \text{sujeto predicado} \\
\text{sujeto} &\rightarrow \text{artículo sustantivo} \\
\text{predicado} &\rightarrow \text{verbo objecto_directo} \\
\text{objeto_directo} &\rightarrow \text{artículo sustantivo} \\
\text{artículo} &\rightarrow \text{el} \\
\text{sustantivo} &\rightarrow \text{perro} \\
\text{sustantivo} &\rightarrow \text{auto} \\
\text{verbo} &\rightarrow \text{mira}
\end{aligned}$$

A cada $A \rightarrow B$ se la llama una *regla* y se interpreta como que A se puede componer de B. Por ejemplo, *sujeto* se puede componer de *artículo* seguido de *sustantivo*. Partiendo del símbolo inicial *oración* se pueden aplicar sucesivamente estas reglas reemplazando una ocurrencia de A por B hasta que se obtiene la oración original "el perro mira el auto". A este proceso de aplicar sucesivamente reglas se lo llama *una derivación*. En este punto ajustaremos un poco la terminología, en vez de hablar de "oraciones" nos referiremos a "secuencias" y en vez de decir "palabras" diremos "símbolos". Ahora es importante hacer una distinción cualitativa, hay algunos símbolos que forman la secuencia (*perro,el,auto,mira*) y otros distintos que se usan para formar las reglas (*sujeto,predicado,artículo,...*). A los primero se los llama *símbolos terminales*, mientras que a los segundos se los llama *símbolos no-terminales* puesto que se usan en los pasos intermedios de las derivaciones. Otro punto importante de este ejemplo es que hay dos reglas para *sustantivo* y que cualquiera de las dos puede ser usada, es por esto que usando la misma gramática también se puede derivar la secuencia "el perro mira el perro". Es decir, esta gramática tiene la propiedad de derivar al menos dos secuencias distintas. En este trabajo esta propiedad es importante puesto que las gramáticas usadas en el problema de la gramática mínima deben poder derivar sólo una secuencia.

Esta forma de ver una gramática como un conjunto de reglas $A \rightarrow B$ se formaliza de manera simple en términos matemáticos y esto ha permitido un estudio bastante profundo de sus propiedades. Su representación y tratamiento computacional también ha sido ampliamente estudiado al punto que se utilizan en compiladores de lenguajes de programación, en compresión de datos y en descubrimiento de patrones, sólo para nombrar algunas aplicaciones. Variantes probabilísticas de las gramáticas libres de contextos son muy activamente utilizadas e investigadas en el área de procesamiento de lenguaje natural.

Habiendo introducido esto, el problema de la gramática mínima es el siguiente. Dada una secuencia w, cómo construir una gramática libre de contexto capaz de derivar sólo a w y cuyo tamaño sea mínimo. En este

contexto, "secuencia" quiere decir cualquier secuencia de símbolos, ya sean letras, números, espacios en blanco, datos binarios, etc. Decir que se busca una gramática de tamaño mínimo puede sonar confuso, porque ¿qué es el tamaño de una gramática? Sin profundizar demasiado diremos que una gramática que tiene reglas más largas es más grande que una que tiene reglas con menos símbolos. Entonces, parafraseando intuitivamente el problema, se busca cómo construir una gramática libre de contexto que derive una única secuencia tal que la longitud de sus reglas sea lo menor posible. Entonces la solución a este problema es un algoritmo, es decir un proceso mecánico que se puede ejecutar dentro de una computadora, que reciba como dato una secuencia y que en un tiempo razonable calcule una gramática de tamaño mínimo para esa secuencia.

Las motivaciones que llevan a buscar una solución a este problema son varias. Para empezar, la mayor cantidad de aportes a la solución de este problema fueron efectuados por investigadores con interés en compresión de datos. Varios algoritmos de compresión de datos pueden ser vistos en términos de gramáticas libres de contexto (por ejemplo LZ78[19]) tomando ventaja de que almacenar a la gramática de una secuencia en un archivo usa menos espacio que almacenar la secuencia misma, y puesto que teniendo una gramática es posible recuperar la secuencia que le dió origen, se podría comprimir un archivo de esta forma. La idea es que en vez de almacenar una secuencia cualquiera w sólo sería necesario almacenar una gramática que genere a w, y mientras más pequeña la gramática, mejor. Por usos de esta idea ver Apostolico y Lonardi [1], y Nevill-Manning y Witten [16]. A modo de ejemplo, considere la siguiente secuencia w:

$$w = baab\ baab\ baab \ldots baab$$

Y ahora considere la siguiente gramática para esta secuencia:

$$\begin{aligned} \mathrm{S} &\rightarrow AAA \ldots A \\ \mathrm{A} &\rightarrow baab \end{aligned}$$

La cual se podría almacenar usando menos espacio puesto que cada vez que se usa A en vez de $baab$ se ahorran 3 símbolos.

Por otro lado, ha habido algunos investigadores que han presentado aportes al área motivados por la posibilidad de descubrir patrones en w. Como se explicará un poco más adelante en este capítulo las gramáticas libres de contexto pueden describir una estructura que es subyacente en el lenguaje que generan, y claramente, los no-terminales y constituyentes de una gramática se pueden ver como patrones y regularidades que podrían ser interpretadas por el observador interesado [15]. La factibilidad de esta motivación para la obtención de gramáticas mínimas ocupa un espacio importante en este trabajo y se desarrollará más extensamente en el Capítulo 4.

Finalmente, existe una motivación para la solución a este problema relacionada con la complejidad de Kolmogorov. La complejidad de Kolmogorov es una forma propuesta para medir la complejidad de un objeto y, en este caso, la complejidad de una secuencia. Tal como se explica en Charikar et al. [6] el tamaño de una gramática mínima se puede ver como una forma más tratable de calcular la complejidad de una secuencia en el espíritu de las ideas de Kolmogorov.

Resolver el problema de la gramática mínima no es difícil, es sencillo construir algoritmos que lo resuelven, pero la cantidad de tiempo en el que se ejecutan es exponencial, esto significa que si el tamaño de los datos de entrada es N, el tiempo que le tomará al algoritmo terminar es del orden de 2^N. Esto podría no significar mucho para una persona ajena a la matemática, así que lo clarificaré con un ejemplo. Si andar en auto 1 Km toma 1 minuto, entonces es razonable esperar que andar 10 Km tome 10 minutos y que andar 53 Km tome 53 minutos. Sin embargo, cuando se trata con algoritmos exponenciales sucede lo siguiente: si andar 1 Km toma un 1 minuto, andar 10 Km toma $2^{10} = 1024$ minutos y andar 53 Km toma 2^{53} minutos, lo cual es más tiempo que la edad del universo.

Entonces, idealmente la solución al problema de la gramática mínima es encontrar un algoritmo que calcule la gramática mínima en tiempo polinomial. Esto quiere decir que si el tamaño de los datos de entrada es N la cantidad de tiempo que le tomaría al algoritmo terminal es del orden de N^k donde $k = 0, 1, 2, 3, \ldots$ es un número fijo. O sea que tomando $k = 3$ y volviendo al ejemplo del auto, esto querría decir que si recorrer 1 Km toma 1 minuto entonces recorrer 53 Km toma $53^3 = 148877$ minutos, lo cual es mucho, pero está muy lejos de ser la edad del universo.

Desafortunadamente, un resultado reciente demuestra que no existe un algoritmo que calcule la gramática mínima y se ejecute en tiempo polinomial a menos que sea cierto que $P = NP$[13]. El problema de decidir si $P = NP$ o $P \neq NP$ es un problema abierto de computación de una importancia fundamental, planteado hace 40 años, y que fue estudiado por muchos científicos del área desde entonces. Esto implica que resolver el problema de la gramática mínima con un algoritmo que se ejecuta en un tiempo razonable sería equivalente a responder si $P = NP$ o $P \neq NP$. Por lo cual, dada la magnitud de esta tarea, en el contexto de este trabajo sólo se aspira a descubrir algoritmos para aproximar el tamaño de la gramática lo mas cercano al mínimo posible.

El procedimiento habitual para evaluar el desempeño de los algoritmos en el problema de la gramática mínima es ejecutarlos sobre un conjunto predefinido de secuencias y reportar los tamaños obtenidos sobre estas secuencias. A este conjunto de secuencias se les llama un *corpus*. En general, un corpus es un conjunto de ejemplos representativos extraídos de la realidad con el expreso propósito de ser utilizados en experimentos. Puesto que la mayor cantidad de aportes al problema de la gramática mínima fueron reali-

zados con el propósito de comprimir datos. Para evaluar nuestros algoritmos utilizaremos dos corpus que son estándar en la comunidad de compresión de datos. Por un lado el Cantenbury Corpus [2], que consiste de 11 archivos de contenido variado elegidos para representar los tipos de datos que se suelen comprimir. Y por otro lado el corpus histórico compilado por Manzini y Rastero en [14] usado específicamente para evaluar la compresión de ADN. Para cada archivo de estos corpus se reporta el tamaño de la gramática obtenida al ejecutar un algoritmo sobre el contenido del archivo.

Por otro lado, en lo que respecta al descubrimiento de patrones la relación entre las gramáticas y los patrones es casi evidente. Ya hemos visto en un ejemplo de lenguaje natural cómo la gramática puede contener información fundamental sobre el lenguaje, como las categorías gramaticales y la forma en la que estas se combinan. Como ya se dijo, las gramáticas libres de contexto describen una estructura que es subyacente en el lenguaje que generan, y claramente, las palabras y constituyentes de una gramática se pueden ver como patrones y regularidades que podrían ser aprehendidos por el observador interesado. Pero uno se preguntaría ¿porqué tienen que ser gramáticas mínimas? Dada una secuencia de datos, la cantidad de posibles patrones o regularidades que se pueden encontrar es enorme. Sin un criterio para decidir si un conjunto de patrones es más útil que otro, el problema se vuelve intratable. Por lejos, el criterio más ampliamente utilizado para resolver este dilema es el principio de descripción mínima, como una medida de la informatividad, el cual fue introducido por Jorma Rissanen en 1978 [18]. Vale recalcar que, a su vez, que el principio de descripción mínima es una formalización del criterio conocido como la navaja de Occam, que goza de gran importancia epistemológica. Este criterio dice “las entidades no deben ser multiplicadas innecesariamente”.

De esta forma, se esperaría que una gramática inferida usando el principio de descripción mínima tenga entre sus constituyentes patrones de utilidad. Y así es descripto en el trabajo de Nevill-Manning y Witten[16] hecho en 1997. Sin embargo, puede darse, por diversos motivos, que esta forma de aproximarse al descubrimiento de patrones no sea viable: Al existir una imposibilidad práctica para obtener la gramática mínima en un tiempo razonable puede darse que los patrones obtenidos por aproximaciones subóptimas no converjan suavemente a los patrones que tendría la gramática óptima, esto es, que las gramáticas casi óptimas no compartan casi ningún patrón con la óptima. Otra dificultad que podría existir es que el poder expresivo de las gramáticas libres de contexto no sea el suficiente para capturar la estructura que generó a la secuencia. De darse alguno de estos hechos, implicarían una imposibilidad práctica y hasta tal vez teórica, de utilizar gramáticas libres de contexto para el descubrimiento de patrones.

Con respecto al problema de la gramática mínima, los resultados obtenidos fueron muy satisfactorios. Para empezar, el problema se abordó de la siguiente forma:

Se desarrolló una caracterización del conjunto de posibles soluciones al problema de la gramática mínima, la cual es un subconjunto estricto de las gramáticas libres de contexto. Es decir, no toda gramática libre de contexto puede ser una solución sino que debe tener una forma especial. Esto se debe a que las restricciones propias del problema, por ejemplo que el lenguaje debe contener exactamente una secuencia, implican que algunas gramáticas no pueden ser solución al problema. En el contexto de este trabajo se definen aquellas gramáticas que cumplen con estas condiciones más estrictas como *gramáticas compactas* y se prueba que la gramática que soluciona el problema es una gramática compacta. Fundamentalmente, una gramática compacta no tiene reglas repetidas y puede ser caracterizada únicamente con un árbol sintáctico (o un árbol de derivación) sobre la secuencia w. Por ejemplo, para la siguiente oración *"El perro es feliz"*, se propone un árbol sintáctico:

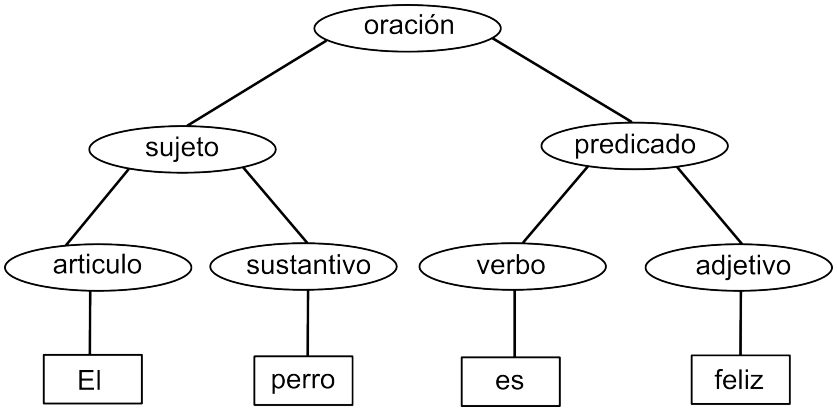

Figura 1.1: Un árbol de análisis sintáctico

Y este árbol sintáctico caracteriza unívocamente a la siguiente gramática compacta:

$$\begin{aligned} \text{oración} &\rightarrow \text{sujeto predicado} \\ \text{sujeto} &\rightarrow \text{artículo sustantivo} \\ \text{predicado} &\rightarrow \text{verbo adjetivo} \\ \text{artículo} &\rightarrow \text{el} \\ \text{sustantivo} &\rightarrow \text{perro} \\ \text{verbo} &\rightarrow \text{es} \\ \text{adjetivo} &\rightarrow \text{feliz} \end{aligned}$$

El hecho que la caracterización de las gramáticas compactas tenga varias restricciones implica que el espacio de búsqueda de los algoritmos es menor en tamaño al de las gramáticas libres de contexto y estas restricciones son aprovechadas por el algoritmo que es eje de este trabajo.

El pilar fundamental de este trabajo es un algoritmo polinomial de optimización que ordena el espacio de búsqueda del problema de la gramática mínima volviendolo mucho más simple de explorar. Este algoritmo, llamado *reordenación*, permite que sea suficiente para caracterizar la solución del problema con encontrar un conjunto de constituyentes adecuados. Un constituyente de una gramática es la secuencia que se obtiene al derivar un símbolo no-terminal hasta que la secuencia este compuesta solo por símbolos terminales. Por ejemplo, en la gramática:

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{ABAab} \\ \mathrm{A} &\rightarrow \mathrm{aba} \\ \mathrm{B} &\rightarrow \mathrm{bab} \end{aligned}$$

Los constituyentes de la gramática son aba, bab y abababababaab.

El algoritmo de reordenación recibe un conjunto de subsecuencias y genera una gramática que tiene tamaño mínimo en el conjunto de todas las gramáticas que tienen exactamente ese conjunto de subsecuencias como constituyentes. Es decir, fijado un conjunto de constituyentes se puede obtener una gramática mínima en tiempo polinomial. Otra forma de decirlo es que el algoritmo de reordenación puede encontrar una gramática mínima en una partición del espacio de búsqueda y que cada partición puede ser caracterizada con cada conjunto de constituyentes.

Por suerte, dada una secuencia w de longitud N la cantidad de posibles constituyentes en una gramática para w es finita y es de orden $O(N^2)$ puesto que por las restricciones de las gramáticas compactas un posible constituyente solo puede ser una subsecuencia de w, y que además tenga longitud mayor o igual a dos y frecuencia mayor o igual a dos. Se prueba en este trabajo que hay un subconjunto de estos posibles constituyentes tal que si se lo da al algoritmo de reordenación produce la gramática mínima globalmente. Por desgracia, la cantidad de subconjuntos de un conjunto es exponencial, es decir, si hay M posibles constituyentes entonces hay 2^M subconjuntos de posibles constituyentes. Esto implica que utilizar la reordenación para encontrar la solución perfecta al problema es impráctico.

Sin embargo el espacio de búsqueda (reordenado con una optimización de tiempo polinomial) es ahora más simple de comprender y explorar, solo se esta trabajando con conjuntos de secuencias elegidas de un conjunto más grande. Entonces, un algoritmo de aproximación podría sencillamente visitar ciertos conjuntos de constituyentes, calcular en tiempo polinomial una gramática mínima para cada uno de ellos y devolver una de ellas. Y lo que es más interesante es que cualquier algoritmo que visite una cantidad polinomial de estos conjuntos de constituyes sería un algoritmo polinomial para aproximar una solución al problema de la gramática mínima. Los algoritmos de aproximación que se desarrollan en este trabajo (**zigzag** entre ellos) se basan en esta idea: visitan un conjunto de constituyentes y luego eligen un

conjunto sucesor, una cantidad polinomial de veces, hasta que se detienen y devuelven la última gramática producida, que es además la menor de las exploradas.

Usando la reordenación se encontró un algorítmo para aproximarse a una gramática mínima bautizado **zigzag**, el cual resultó bastante exitoso puesto que supera al mejor algoritmo previamente conocido en un 4 % o hasta 10 % dependiendo de los casos, convirtiendose en el nuevo estado del arte en aproximación de gramáticas mínimas.

En lo que respecta al descubrimiento de patrones los resultados obtenidos también fueron satisfactorios, por sobretodo un tanto reveladores.

La forma aquí usada para evaluar las posibilidades de este enfoque en el descubrimiento de patrones nace de la siguente situación hipotética: Dados dos investigadores que quieren descubrir patrones sobre la misma secuencia de datos ¿Cuanto acuerdo podrían lograr entre ellos si ambos se guiaron por el principio de que las gramáticas pequeñas tienen patrones útiles?

Para responder a esta pregunta, y basandose en la idea que distintos investigadores usarían distintas estrategias para encontrar gramáticas pequeñas, se buscaron varias gramáticas pequeñas y diversas para una misma secuencia. Esto se logró adaptando ligeramente el algorítmo de reordenación y el de **zigzag** para que explorasen diversas partes del espacio de búsqueda.

Una vez encontradas gramáticas diversas se realizan mediciones sobre dos características distintas. El objetivo de estas mediciones es poder capturar las características que serían esenciales si se estuvieran buscando patrones. La primera característica es las palabras que se obtienen de la gramática. Los constituyentes de cada gramática representan las "palabras" que esta tiene. La segunda característica es la estructura jerárquica de la gramática, es decir, en términos del primer ejemplo, características como que el *artículo* esta dentro de *sujeto* y que este a su vez esta dentro de la *oración*.

Para medir la similitud entre las palabras de dos gramáticas distintas se utilizo el índice de Jaccard, el cual es una medida estándar para comparar similitud entre conjuntos, y la F-measure entre los conjuntos, la cual es una métrica proveniente del área de clasificación estadística. El procedimiento es tomar los conjuntos de constituyentes de cada gramática y reportar el índice de Jaccard y la F-measure entre estos dos conjuntos.

Para poder comparar similitud estructural, dado que las gramáticas mínimas tiene un único árbol de derivación (y estos son árboles de parseo), resultó natural tomar una medida estándar usada por la comunidad de parsing para comparar árboles de parseo como una forma de comparar la similitud estructural de dos gramáticas. Esta medida se llama *unlabeled bracket F-measue*, la cual es un número entre 0 y 1, tipicamente expresado como un porcentaje que indica disimilitud total en 0 % e igualdad en 100 %.

Los experimentos se llevaron a cabo sobre secuencias conteniendo ADN y sobre una secuencia que contiene lenguaje natural. Se encontró que los patrones obtenidos utilizando estos atributos son de escasa confianza puesto

que el acuerdo entre dos estos investigadores hipotéticos puede ser bastante bajo, y que para confirmar los patrones como válidos sería necesarío realizar multiples busquedas de patrones con diversas gramáticas pequeñas o contar con un experto en el dominio de los datos que los confirme.

El resto del trabajo esta estructurado de la siguiente forma:

- En el Capítulo 2 se dan definiciones elementales, se explica el problema de la gramática mínima y algunos algoritmos de aproximación a la gramática mínima previamente existentes en la literatura.
- En el Capítulo 3 se definen los conceptos, los algoritmos desarrollados para este trabajo y se presentan los resultados obtenidos utilizando **zigzag** en dos corpus estándares.
- En el Capítulo 4 se explica y justifica la metodología utilizada para evaluar la factibilidad de las gramáticas mínimas para el descubrimiento de patrones y luego se muestran los resultados experimentales obtenidos sobre algunos archivos obtenidos de corpus estándares.
- En el Capítulo 5 se conluyen este trabajo y se explican posibles vias de trabajo futuro.

Capítulo 2

Marco teórico

En este capítulo se darán algunas definiciones básicas que se utilizan a lo largo de este trabajo. Como convención, a lo largo de este trabajo la variable w se usará para hacer referencia a la secuencia de entrada sobre la cual se infieren gramáticas y N será la longitud de tal secuencia.

2.1. Gramáticas libres de contexto

Una gramática libre de contexto es una 4-upla $G =< V, T, P, S >$ donde V es un conjunto de elementos llamados *no-terminales*, T es un conjunto disjunto del anterior cuyos elementos son llamados *terminales*, S es un elemento especial de V llamado *símbolo inicial*, y por último P es un conjunto de reglas de la forma $A \to \alpha$ donde $A \in V$ y $\alpha \in (V \cup T)^*$ es una secuencia de símbolos llamada la *definición* de T o el *cuerpo de la regla* de T.

Se dice que α *deriva a* β (en símbolos $\alpha \overset{*}{\Rightarrow} \beta$) si reemplazando sucesivas veces un no-terminal de α por su definición se obtiene β.

Se define la *expansión* de una gramática como una función $E : (V \cup T)^* \to \mathcal{P}(T^*)$ tal que cada palabra de $E(\alpha)$ se obtiene por sucesivos reemplazos de cada no-terminal de α por su definición hasta que solo queden terminales.

Se define el *lenguaje* de una gramática $G =< V, T, P, S >$ como el conjunto todas las palabras formadas por terminales que se pueden derivar a partir del símbolo inicial, es decir $E(S)$.

Se definen los *constituyentes* de una gramática cómo la expansión de cada uno de los no-terminales de la grámatica, es decir $\bigcup_{A \in V} E(A)$.

Dada una gramática libre de contexto $G =< V, T, P, S >$ se define un árbol de derivación (también conocido como árbol sintáctico concreto o árbol de parseo) como un árbol que cumple las siguientes propiedades:

1. Cada nodo tiene una etiqueta que es un símbolo de $V \cup T$

2. La etiqueta del nodo raíz es S

3. Si un nodo es interior y tiene etiqueta A entonces $A \in V$
4. Dado un nodo interno, sus hijos tienen un orden.
5. Si el nodo n tiene etiqueta A y los nodos $n_1, \ldots, n_k$ son sus nodos hijos en orden con etiquetas $X_1, \ldots, X_k$ respectivamente entonces

$$A \to X_1 \ldots X_k$$

es una producción de P.

Es importante notar que si la gramática es inambigüa y tiene una única palabra en el lenguaje entonces existe un único árbol de derivación asociado a esta gramática. Este hecho se mencionará y utilizará en el Capítulo 4

A continuación se presenta como ejemplo una gramática inambigüa con lenguaje $\{abababababaabaababababa\}$:

$$S \to AaBA$$
$$A \to aBBBB$$
$$B \to ba$$

Y su correspondiente árbol de derivación (existe solo uno) en la Figura 2.1

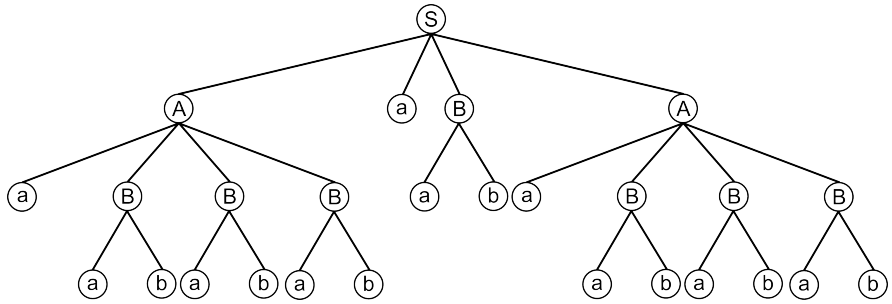

Figura 2.1: Un árbol de derivación

2.2. El problema de la gramática mínima

Dada una secuencia w, el problema de la gramática mínima es computar la menor gramática libre de contexto $G =< V, T, P, S >$ cuyo lenguaje es exactamente $\{w\}$.

Dado que queremos encontrar gramáticas de tamaño mínimo necesitaremos una métrica que nos permitar compararlas. En la literatura de gramáticas mínimas para compresión de datos se han definido diversas formas de evaluar el tamaño de una gramática, por ejemplo, en [13] la definen como:

$$|G| = \sum_{A \to \alpha \in P} |\alpha|$$

Donde $G =< V, T, P, S >$ es una gramática libre de contexto. Esta resulta una definición simple y practica que es bastante utilizada cuando los autores buscan obtener resultados teóricos. Otros autores prefieren utilizar una métrica un poco más aplicada al área de compresión, que incluye el costo en bits basado en la probabilidad de ocurrencia de los símbolos de la gramática [1].

Basandonos en una justificación que se explicara más adelante, para este trabajo se define el tamaño de G como:

$$|G| = |V| - 1 + \sum_{A \to \alpha \in P} |\alpha|$$

Es decir, el tamaño aumenta en 1 por cada no-terminal salvo el inicial y en 1 por cada símbolo en la definición de cada regla, lo cual es intuitivamente aceptable. Una motivación más detallada de esta métrica se dará luego de que se introduzcan las gramáticas compactas.

Como acotación final al problema de la gramática mínima, recientemente se probó que incluso aproximar una solución por debajo de un factor de aproximación pequeño es NP-Hard[13].

2.3. Algoritmos previos

LZ78 Originalmente descripto por Lempel y Ziv[19]. Basicamente busca crear reglas nuevas reutilizando una regla anterior y concatenandole un símbolo nuevo. Es por esto que funciona en tiempo casi lineal sobre el tamaño de w. Originalmente fue diseñado para la compresión de datos y no fue expresado en términos de gramáticas, sin embargo, la estructura generada por LZ78 puede ser mapeada a las gramáticas libres de contexto.

Sequitur Fue descripto por Nevill-Manning y Witten[15], Consiste de un procedimiento simple, que sucesivamente agrega los símbolos observados a la producción inicial. Si el último par de símbolos agregado se repite en otro lugar de la gramática, se genera una regla nueva y se reemplazan sus ocurrencias. Despues de agregar cada símbolo se fuerzan dos propiedades basicas sobre la gramática resultante: ningun par de símbolos adyacentes aparece más de una vez en la gramática y toda regla es usada más de una vez. Aunque generalmente es considerado de importancia historica más que práctica, es interesante notar que en su publicación original se muestran cualitativamente las capacidades de este algoritmo de inferencia de gramáticas para el descubrimiento de patrones.

Sequential Publicado como "Greedy senquential grammar transform" por Yang y Kieffer[10]. El procedimiento es una version mejorada de Sequitur que busca el mayor prefijo en la parte sin procesar de la entrada que pueda

ser generado por una regla ya existente y agrega su no terminal a la regla inicial. Si no existe tal prefijo entonces procede como sequitur. Un concepto interesante que fue introducido junto con la publicación del algoritmo es el de *irreducible grammar*, las cuales son gramáticas para las cuales existen cotas superiores para su ratio de aproximación, y además se incluye una prueba de que Sequential genera este tipo de gramáticas.

Greedy Publicado por Apostolico y Lonardi[1], se trata de una idea simple y eficiente, comenzando con una única producción $S \rightarrow w$ consiste en repetidamente agregar reglas $A \rightarrow \alpha$ y reemplazar todas las ocurrencias de α en la gramática por A de manera que el tamaño de la gramática disminuye lo más posible. El α a reemplazar es elegido como el que maximiza la reducción de tamaño en la gramática. Al haber sido diseñado para la compresión de datos, la métrica utilizada para medir el tamaño de la gramática es bastante diversa a la que se utiliza en este trabajo e incluye consideraciones como la entropía de la secuencia. En la implementación que se considerará aquí la métrica que Greedy utiliza es la misma que utiliza **zigzag**.

2.4. Similitud entre conjuntos

Aquí se detallan algunas métricas clásicas para medir similitud entre conjuntos que serán utilizadas ampliamente en el Capítulo 4 para reportar similitud entre gramáticas.

El índice de Jaccard Dados dos conjuntos A, B el índice de Jaccard se define como

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

y busca medir la similitud entre estos dos conjuntos con un número que va entre 0 y 1, 0 indicando disimilitud total (tienen una intersección vacia) y 1 indicando igualdad. Fue concebido por el botánico Paul Jaccard en 1901 [11]. Guarda una fuerte relación con el coeficiente de Tanimoto, el cual es una extensión de la similitud del coseno.

Precision & Recall Se trata de dos métricas complementarias ampliamente usadas para cuantificar la forma en que dos conjuntos se superponen. Fueron originalmente desarrolladas para la tarea de clasificación estadística, en la cuál se debe decidir qué tan bueno es el resultado del un algoritmo comparandolo con un conjunto de test con respuestas correctas al problema atacado.

Entonces, en el escenario en el que fueron desarrolladas se considera un conjunto con los datos de test correctos A y un conjunto que se datos que se desean evaluar B. Intuitivamente hablando precision es la proporción de

respuestas correctas dentro de B, o sea la proporción de los elementos de B que estan en A. Recall (covertura) es la proporción de respuestas dadas dentro de B, o sea la proporción de los elementos de A que estan en B. Para explicarlo con una Figura (2.2) Precision es el ratio entre la cardinalidad

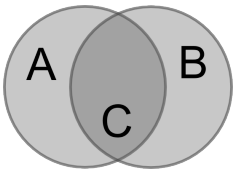

Figura 2.2: Diagrama de venn para explicar Precision & Recall, C es la intersección entre A y B

de C y la cardinalidad de A, y Recall es el ratio entre el tamaño de C y el tamaño de B. En formulas:

$$Prec_A(B) = \frac{|A \cap B|}{|B|}$$

$$Rec_A(B) = \frac{|A \cap B|}{|A|}$$

Notar que tanto $Prec$ como Rec son números entre 0 y 1, aunque suelen ser presentados también como un porcentajes. Puesto que en este trabajo simplemente se pretende comparar dos conjuntos de los cuales ninguno puede ser considerado como correcto se hace notar una simetria interesante:

$$Prec_A(B) = Rec_B(A) \text{ y } Rec_A(B) = Prec_B(A)$$

De esta forma se puede ver que aunque no se distinga entre conjunto de evaluación y de test aun se puede reportar precision y recall como una medida de similitud puesto que existe esta simetría.

F-measure También conocida como F_1 score o coloquialmente como harmonic mean se trata de otra medida clásica de la comunidad de calsificación estadística. Es una combinación de precision y recall en una única métrica que permite ser más síntetico a la hora de presentar y comparar resultados. La idea es calcular una media entre los valores de $Prec$ y Rec, en particular, la media harmónica entre estos dos valores. En formulas, dados dos conjunto A y B:

$$F_1(A, B) = 2\frac{Prec_A(B)Rec_A(B)}{Prec_A(B) + Rec_A(B)}$$

Notar que F_1 es un número entre 0 y 1, aunque suele ser presentado también como un porcentaje. Como se puede notar en la definición se asume que el

conjunto considerado correcto es A. Sin embargo notando las simetrías dadas al explicar $Prec$ y Rec se puede ver facilmente que:

$$F_1(A,B) = 2\frac{Prec_A(B)Rec_A(B)}{Prec_A(B)+Rec_A(B)} = 2\frac{Prec_B(A)Rec_B(A)}{Prec_B(A)+Rec_B(A)} = F_1(B,A)$$

Por lo tanto F_1 es una buena forma de medir la similitud entre dos conjuntos aun cuando no se distinga entre conjunto de evaluación y de test.

Capítulo 3

Gramáticas mínimas

En este Capítulo se describira un punto de vista distinto al dado en los trabajos previos puesto que plantea una forma no antes utilizada de explorar el espacio de búsqueda. Luego se introduciran una secuencia de algorítmos que juntos componen el principal aporte al problema de la gramática mínima de este trabajo, el algorítmo **zigzag**.

3.1. Gramáticas compactas

Ahora definiremos un subconjunto de las gramáticas libres de contexto sobre el cual los algoritmos presentados buscarán la de tamaño mínimo.

Definicion 1. *Una gramática libre de contexto* $C =< V, T, P, S >$ *será llamada una gramática compacta sii:*

- $L(C) = \{w\}$ *y* $|w| > 1$
- C *no tiene símbolos inutiles.*
- $|\alpha| > 1$ *para cada* $A \rightarrow \alpha \in P$
- *Para cada elemento de* V *hay una única regla en* P*.*
- *Cada no-terminal deriva exactamente una secuencia de* T^**.*
- *Cada no-terminal deriva una secuencia distinta de* T^**.*

Si se observa con cuidado la definición de gramática compacta se prodrá notar que algunas propiedades implican a otras, esto fue hecho intencionalmente para facilitar la presentación de los algoritmos que se basan en estas propiedades.

La definición de gramática compacta no era previamente existente en la literatura sino que fue desarrollada para su uso en este trabajo. Sin embargo, conceptos similares fueron acuñados por otros autores en sus trabajos, como

por ejemplo la definición de *gramática admisible* dada por Kieffer y Yang [8].

Es importante hacer notar que de la definición se desprende que una gramática compacta no es ambigüa y que su lenguaje tiene una sola palabra, por lo tanto tiene un único árbol de derivación. Es más, este árbol de derivación es suficiente para caracterizar unívocamente a la gramática compacta a la que esta asociado, es decir que una gramática compacta y un árbol de derivación expresan la misma forma de estructurar una secuencia. Este hecho es importante y se recordará en el Capitulo 4.

Además, aún cuando la definición de gramática compacta parece muy restrictiva se puede probar que la gramática libre de contexto de tamaño mínimo es una gramática compacta(ver prueba en el apéndice A.1). Por lo tanto, es suficiente con explorar el espacio de las posibles gramáticas compactas para encontrar la mínima y podemos contar con las propiedades restrictivas de la definición para probar teoremas y algoritmos.

En este punto estamos en condiciones de dar una motivación alternativa a nuestra definición de tamaño. Se considera que una buena medida del tamaño de una gramática es el costo de descripción de la misma, es decir, si se la expresara como una secuencia de caracteres ¿que tal grande sería? Entonces se eligió una forma simple de codificar una gramática compacta en una secuencia de caracteres y se definirá el tamaño de la gramática como la longitud de esta secuencia. La forma de construir la secuencia es la siguiente:

- Se empieza agregando los símbolos del cuerpo de la regla inicial.
- Cada vez que ocurre por primera vez un no-terminal, se inserta a continuación del no-terminal la definición de su regla y cuando esta termina se inserta un símbolo especial (que no esta en $V \cup T$) marcando el final de la definición. Este paso puede aplicarse recursivamente.

Por ejemplo, para:

$$S \rightarrow abAbBaAa$$
$$A \rightarrow abBa$$
$$B \rightarrow bab$$

La secuencia sería:

$$abAabBbab|a|bBaAa$$

Donde | es el símbolo especial que marca el final de una definición. El tamaño de esta gramática como fue definido previamente es 17, al igual que la longitud de la secuencia, es decir, la longitud de esta secuencia esta dada por la función de tamaño que fue definida en la sección anterior. Notar que una de estas secuencias identifica univocamente a una gramática solo si esta es una gramática compacta.

3.2. Reordenación

Ahora veremos que si se fijan las siguientes condiciones podemos encontrar la gramática compacta de tamaño mínimo en tiempo polinomial. El truco es mantener los constituyentes de la gramática fijos, y luego, minimizar el cuerpo de las reglas que generan esos constituyentes usando el camino más corto en un grafo. De esta forma se obtiene un algoritmo $O(N^2M)$ que será el eje central de los algoritmos que siguen (N y M se definen luego).

Supongamos entonces que restringimos el espacio de búsqueda a aquellas gramáticas compactas que tienen ciertos constituyentes $\{y_1, \ldots, y_m\}$ fijos y queremos saber cuál es la gramática de tamaño mínimo. Por ser una gramática compacta sabemos que debe haber exactamente una regla y un no-terminal para cada y_i. Nombraremos V_i al terminal asociado a y_i. Sin perdida de generalidad podemos asumir que el símbolo inicial S es igual a V_1. Por lo tanto, salvando isomorfismos, todas las gramáticas de nuestro espacio de búsqueda comparten los terminales, los no-terminales y el símbolo inicial. Lo único que las diferencia entre si es el conjunto de las producciones. Entonces buscar la gramática mínima es equivalente a minimizar

$$\sum_{A \to \alpha \in P} |\alpha|$$

Pero puesto que una vez fijados los constituyentes la redefinición de una producción no afecta la definición de otra, minimizar esta sumatoria es equivalente a minimizar por separado la longitud de cada $|\alpha|$. Ahora veremos que minimizar $|\alpha|$ se puede ver como encontrar el camino mas corto en un grafo.

Se presentara la forma de construir el grafo con un ejemplo, los constituyentes serán los de la gramática de ejemplo vista en la sección anterior, es decir $\{ababbababbabaabbabaa, abbaba, bab\}$ y la regla a minimizar será la inicial, entonces el grafo sera:

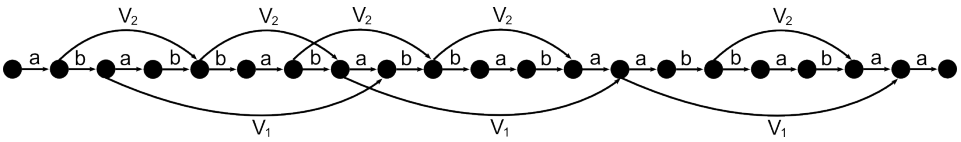

Figura 3.1: Un grafo de reordenación

Como se puede ver, las etiquetas en cada camino que recorre de izquierda a derecha determinan una forma distinta de construir la regla inicial. Por ejemplo, tomar todas las aristas etiquetadas con símbolos terminales nos da $ababbababbabaabbabaa$, si tomamos V_2 cada vez que podemos nos da $aV_2V_2abV_2aabV_2aa$. La idea aquí es que el camino de menor longitud en este grafo determina la forma más corta de escribir la regla inicial, que en este

caso sería $aV_2V_2V_1V_1a$. Notar que el camino mínimo no es necesariamente único, y por lo tanto tampoco lo sera la gramática mínima.

Ahora formalmente, dado que queremos minimizar $|\alpha_i|$ con $V_i \to \alpha_i \in P$, construimos un grafo etiquetado dirigido acíclico con $k+1 = |y_i|+1$ nodos nombrados $n_0, \ldots, n_k$ y aristas:

$$n_a \xrightarrow{y_i[a]} n_{a+1} \quad \text{para } 0 \leq a < k$$
$$n_a \xrightarrow{V_c} n_{b+1} \quad \text{si } y_i[a:b] = y_c \text{ y } c \neq i$$

Donde $s[a]$ es el elemento a-ésimo de una secuencia s, $s[a:b]$ es la subsecuencia de s entre índices a y b, y $n_a \xrightarrow{l} n_b$ representa una arista entre los nodos n_a y n_b con etiqueta l. A este grafo lo llamaremos *grafo_de_reordenacion*$(y_i, \{y_0, \ldots, y_M\})$.

En este punto estamos en condiciones de introducir el algoritmo que es el eje fundamental de este trabajo:

reordenar$(\{y_0, \ldots, y_M\})$:
$q := \{y_0, \ldots, y_M\} \cup \{w\}$
$g :=$ *gramática_vacia*
$i := 0$
for $c \in q$
  $g_c :=$ *grafo_de_reordenacion*(c, q)
  cuerpo := *camino_minimo*(g_c)
  $g := g + \{V_i \to$ *cuerpo*$\}$
  $i := i + 1$
end
return(g)

Figura 3.2: Un algoritmo $O(N^2M)$ para encontrar el óptimo fijados los constituyentes.

Donde *grafo_de_reordenacion* es un grafo construido tal y como fue descrito arriba y *camino_minino* es el algorítmo de Dijkstra. Notar que no es necesario que w este en el conjunto de constituyentes que se le da a reordenar y que por esto *reordenar*$(\{\})$ devuelve una gramática con una sola regla y cuyo tamaño es N, con $N = |w|$.

El costo computacional de contruir el grafo de reordenación es $O(N+N^2)$ puesto que hay que agregar una arista por cada no terminal y una por cada ocurrencia de un constituyente, las cuales pueden ser a lo sumo $\frac{N^2-N}{2}$ porque tienen que ser substrings de w (en detalle mas adelante). Entontrar el camino mínimo en el grafo es $O(N^2)$. Además, el ciclo **for** se repite M veces, donde M es la cantidad de constituyentes en q, por lo tanto la complejidad computacional de **reordenar** es $O(N^2M)$.

Una prueba completa de que **reordenar** produce una gramática mínima se encuentra en el anexo A.2.

3.3. Constituyentes posibles

Así, ya hemos visto que es posible encontrar la gramática óptima en tiempo polinomial si fijamos el conjunto de constituyentes. Además, dada una secuencia w, la gramática mínima que la genera siempre existe y tiene un conjunto de constituyentes. Llamemos a este conjunto de constituyentes q^*. Si ejecutamos el algoritmo de reordenación sobre q^* debería resultar en una gramática de tamaño mínimo globalmente. Entonces si hubiese una cantidad finita de posibles subconjuntos de constituyentes, ejecutando el algoritmo de reordenación para cada uno de estos subconjuntos eventualmente ejecutaríamos la reordenación en q^* y encontraríamos la gramática óptima globalmente.

Pero ¿Son los posibles constituyentes una cantidad finita? Si, lo son, un constituyente solo puede ser útil para la construcción de la gramática mínima si es una subsecuencia de w, porque en caso contrario, la gramática tendría símbolos inútiles. Entonces los posibles constituyentes estan acotados por la cantidad de subsecuencias de w, las cuales nunca pueden superar la cantidad de $\frac{N^2-N}{2}$. De hecho, por lo general serán muchos menos puesto que para que un constituyente sea útil en una gramática mínima debe ocurrir al menos dos veces en los cuerpos de las reglas de la gramática porque de otra manera se ahorarría tamaño no usandolo, y por lo tanto, ninguna subsecuencia que ocurra menos de dos veces en w es un constituyente útil. Además, también se pueden descartar los constituyentes de longitud menor a dos puesto que, por definición, no pueden ser usados en una gramática compacta.

Entonces, definiremos el conjunto de constituyentes posibles para w como el conjunto de subsecuencias de w de tamaño mayor a 2 y que se repitan sin superposición almenos 2 veces. Una profundización sobre los contituyentes y su tratado algoritmico en este trabajo se desarrolla en detalle en el anexo A.3. En lo que resta de este trabajo se supondra que se cuenta con un algoritmo llamado *Constituyentes* que devuelve el conjunto de posibles contituyentes, que se ejecuta en tiempo $O(N^2)$ y que además almacena las posiciones de las ocurrencias de cada constituyente, lo cual es necesario para la construcción eficiente del grafo de reordenación.

Entonces, si C son los posibles constituyentes de w, sabemos que existe un elemento de $\mathcal{P}(C)$ tal que la reordenación de este subconjunto devuelve una gramática de tamaño mínimo. Puesto que $\mathcal{P}(C)$ es un conjunto finito y facilmente calculable a partir de C se podría simplemente revisar todos sus elementos hasta encontrar el que produce la gramática mínima. Con esta idea en mente se definirá en la Figura 3.3 un primer algoritmo que no es una aproximación a la gramática mínima, sino que encuentra la gramática

mínima.

perfecto(w) :
begin
  $g' := reordenar(\{\})$
  for $q \in \mathcal{P}(Constituyentes(w))$ **do**
    $g := reordenar(q)$
    $g' := min(g, g')$
  od
  $return(g')$
end

Figura 3.3: Un algoritmo exponencial para encontrar la solución perfecta.

Puesto que la cantidad de subconjuntos en $\mathcal{P}(C)$ es $2^{|C|}$ el problema práctico de este algoritmo es ser exponencial en la cantidad de constituyentes, lo cual lo vuelve de poco interes práctico. Sin embargo es de utilidad pedagógica puesto que permite encontrar la solución optima para pequeños casos de estudio. Además, nos ofrece un punto de vista distinto acerca de la forma del espacio de búsqueda. Antes, la búsqueda era sobre el conjunto de las posibles gramáticas compactas, que no es un conjunto simple de describir computacionalmete y que, a simple vista, no ofrece una forma natural de ser explorado. Ahora la búsqueda es sobre conjuntos de subsecuencias cuya cantidad es finita y conocida, cuya enumeración es trivial, que pueden ser facilmente descriptos computacionalmente y que son simples de manipular algorítmicamente.

Entonces, para resumir, el algoritmo 3.3 simplemente visita cada elemento de un conjunto y asigna un puntaje a cada uno. El de menor puntaje sera el elemento que corresponde a la gramática mínima.

3.4. El algorítmo zigzag

Nos permitiremos recalcar nuevamente el pilar fundamental de este trabajo antes de seguir avanzando: Ya no es necesario pensar este problema en términos de gramáticas. La gramática mínima queda perfectamente caracterizada una vez fijado el conjunto de constituyentes. Es decir, la pregunta crucial a responder es ¿Cuales contituyentes elegir? Otra forma de expresarlo es que módulo un paso de optimización polinomial el problema cambia a un problema de elegir un subconjunto que minimiza un puntaje.

La reordenación del espacio de búsqueda tiene un orden natural muy interesante, puede ser pensado como un reticulado cuya relación de orden es la inclución de conjuntos, como se muestra en la Figura 3.4.

En este reticulado el elemento *bottom* representa el conjunto vacío, el

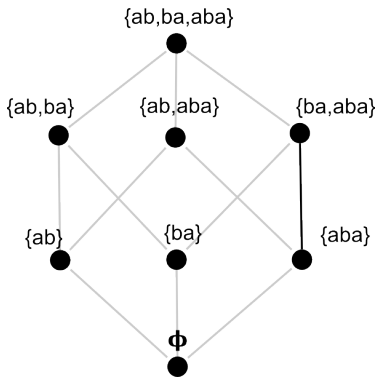

Figura 3.4: Un reticulado para los constituyentes de "abaaba".

elemento *top* representa el conjunto de todos los posibles constituyentes y en su diagrama de Hasse un elemento A tiene una arista con un elemento B si al subconjunto representado por A hay que agregarle exactamente una subsecuencia para obener B. Esta forma de ver el espacio de búsqueda sugiere una forma natural de recorrerlo.

Supongamos ahora que un algoritmo esta explorando este reticulado, y acaba de calcular el puntaje del subconjunto q, entonces ¿Cual sería un buen subconjunto q' en el cual continuar la exploración? Considerando la forma del reticulado dos propuestas surgen naturalmente:

$$\begin{aligned} \text{Agregar un elemento} \quad & q' = q \cup \{c\} \;\; \textit{para} \;\; c \notin q \\ \text{Quitar un elemento} \quad & q' = q - \{c\} \;\; \textit{para} \;\; c \in q \end{aligned}$$

De esta forma un algoritmo podría "caminar" por los arcos del reticulado, agregando o quitando elementos a su conjunto de constituyentes siempre con el fin de ir mejorarando el conjunto actual. Con esta idea en mente se presenta el siguiente algoritmo en la Figura 3.5.

El algoritmo **bottomup** empieza con el subconjunto de constituyentes vacío y lo va aumentando de a un constituyente a la vez con aquel que disminuya en mayor medida el tamaño de la gramática. En cada iteración, debe encontrar el constituyente que agregado a q genera la menor gramática y para poder encontrar este constituyente el algoritmo explora todos los subconjuntos que se obtienen agregando un constituyente. Finalmente se detiene cuando no existe un constituyente que se pueda agregar a q que disminuya el tamaño de la gramática.

Es importante hacer notar que la complejidad de **bottomup** es polinomial con respecto a N. Para ver esto, es necesario notar las siguientes dos propiedades importantes:

El tamaño de la gramática inicial es N: Esto se puede verificar viendo que el algoritmo de reordenación para el caso $q = \{\}$ genera una gramática con una única regla $S \rightarrow w$ y el tamaño de esta gramática es N.

```
bottomup() :
begin
  C := Constituyentes(w)
  q := {}
  g' := reordenar(q)
  while hay mejora en g' do
        for c ∈ (C − q) do
            g = reordenar(q ∪ {c})
            if |g| ≤ |g'| then
                                g' := g
                                c' := c
            fi
        od
        q := q ∪ {c'}
  od
  return(g)
end
```

Figura 3.5: Un algoritmo greedy que agrega constituyentes

Una gramática con más de $N/2$ constituyentes tiene tamaño mayor a N: Es válida puesto que el cuerpo de cada regla tiene longitud almenos 2 y si hay más de $N/2$ reglas entonces el tamaño de la gramática debe ser mayor a N porque solo la suma de la longitud de las definiciones de las reglas sería mayor a N.

Entonces en el ciclo principal a lo sumo se pueden agregar $N/2$ constituyentes, de otra manera el tamaño de la gramática sería superior al tamaño inicial, lo cual no se puede dar porque es un invariante del ciclo, entonces este ciclo se repite $O(N)$ veces. El costo computacional de cada iteración del ciclo principal es el de hacer una reordenación por cada elemento de $(C-q)$, los cuales son $O(N^2)$. La reordenacion tiene complejidad $O(N^3)$ puesto que M es a lo sumo $N/2$. Por lo tanto **bottomup** es $O(N^6)$.

Como se puede ver, **bottomup** es un algoritmo greedy bastante simple pero que no garantiza encontrar la gramática de mínima sino sólo una aproximación a esta. Un recorrido de este algoritmo por el reticulado podría verse como en la Figura 3.6.

En la Figura 3.6 se asume por simplicidad que los constituyentes posibles han sido numerados, entonces un subconjunto de constituyentes se representa con un conjunto de números naturales. Los nodos en negro representan los subconjuntos de constituyentes que son los elegidos, las flechas solidas representan las aristas del reticulado que son tomadas, las a puntos indican que se calculó el tamaño del nodo destino y los conjuntos de números indican

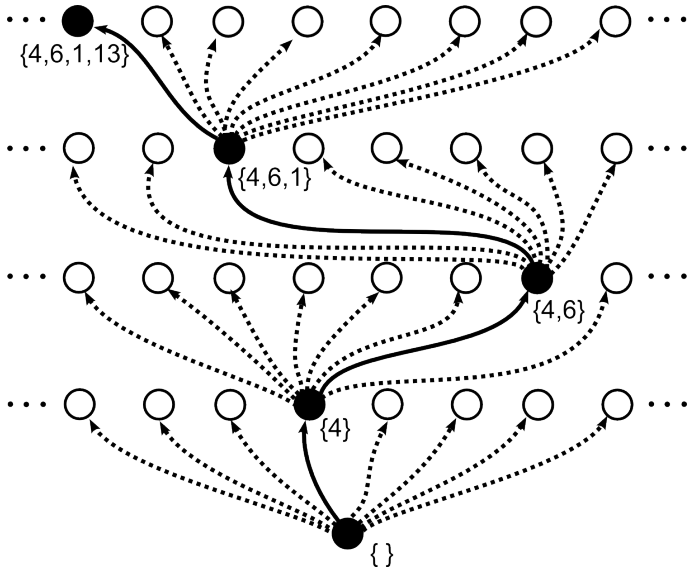

Figura 3.6: Un recorrido de **bottomup** por el reticulado

el subconjunto de constituyentes tomados hasta el momento.

Analogamente a **bottomup** se puede definir un algoritmo que, empezando desde el conjunto de todos los constituyentes posibles, elimina constituyentes de este conjunto uno a uno, siempre quitando el que disminuye el tamaño lo más posible hasta que no puede hacerlo más. Así, este algorítmo decendería por el reticulado hasta un mínimo local. Llamaremos a este algorítmo **topdown**.

Como se puede apreciar en la Figura 3.7, este algóritmo es simétrico a **bottomup**.

Con un argumento similar al de **bottomup** se puede ver que **topdown** tiene complejidad $O(N^8)$. El ciclo externo solo puede quitar $|q|$ elementos q, es decir, hay $O(|q|)$ iteraciones en el ciclo. Cada iteración efectua una reordenación por cada elemento en q. Puesto que M es $O(|q|)$ en este caso el costo computacional de la reordenación es $O(N^2|q|)$ Entonces cada iteración es $O(N^2|q|^2)$ y **topdown** es $O(N^2|q|^3)$. Puesto que inicialmente q es el conjunto de todos los posibles constituyentes entonces $|q| \in O(N^2)$ y por lo tanto se puede decir que la complejidad computacional de **topdown** es $O(N^8)$.

Un recorrido de este algoritmo por el reticulado podría verse como en la Figura 3.8.

Finalmente se introduce un algoritmo que combina pasos de **bottomup** y **topdown** alternadamente. La idea es ascender por el reticulado hasta que no haya más ganancia para luego descender quitando aquellos constituyentes que pueden ser reemplazados por otros, tal vez mejorando el tamaño aun más, y repetir estos dos últimos pasos hasta que no se consiga más ganancia.

```
topdown() :
begin
  C := Constituyentes(w)
  q := C
  g := reordenar(q)
  while hay mejora en g ∧ q ≠ ∅ do
        for c ∈ q do
            g = reordenar(q − {c})
            if |g| ≤ |g'| then
                                g' := g
                                c' := c
            fi
        od
        q := q − {c'}
  od
  return(g)
end
```

Figura 3.7: Una algorítmo greedy que quita constituyentes

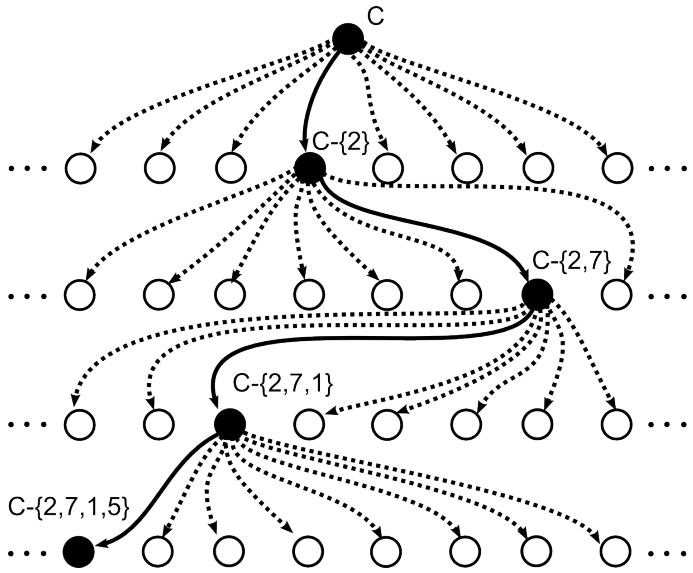

Figura 3.8: Un recorrido de **topdown** por el reticulado

Notar que para "ascender" y "descender" desde algún punto del reticulado utilizando los algoritmos **bottomup** y **topdown** solo es necesario cambiar la forma en la que se inicializa q, por lo tanto, agregaremos un argumento a estos procedimientos que será el valor inicial de q. A este algorítmo lo

presentaremos en la Figura 3.9 y sera llamado **zigzag**.

zigzag() :
$q := \{\}$
$g :=$ *reordenar*(q)
while *hay mejora en* g **do**
    $g :=$ *bottomup*(q)
    $q :=$ *constituyentes de* g
    $g :=$ *topdown*(q)
    $q :=$ *constituyentes de* g
od
return(g)

Figura 3.9: Un algorítmo que recorre el reticulado en zigzag

Un recorrido de este algoritmo por el reticulado podría verse como en la Figura 3.10

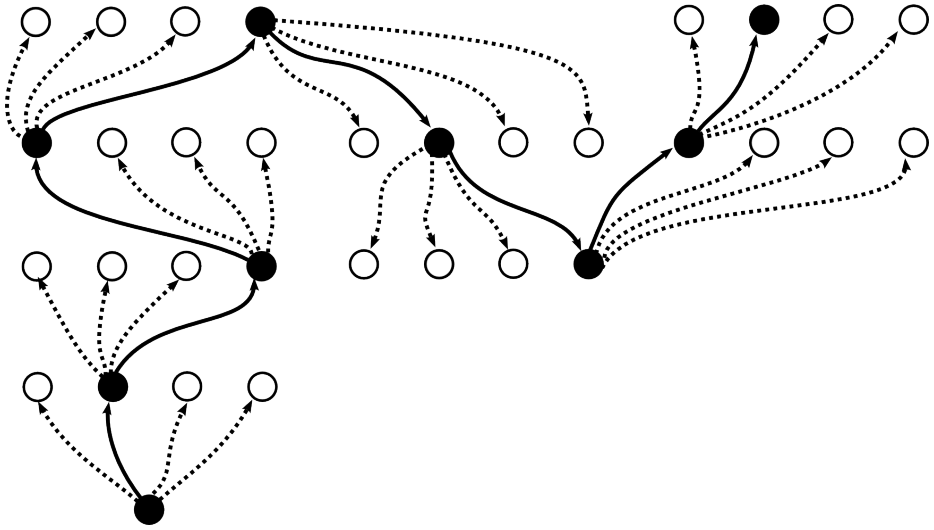

Figura 3.10: Un recorrido de **zigzag** por el reticulado

Este algorítmo también es polinomial, más precisamente $O(N^7)$, por lo siguiente: Ya que el q inicial es el conjunto vacío, el tamaño inicial de g es N, por esto **bottomup** nunca va a agregar más de $N/2$ constituyentes a q porque en ese caso el tamaño de g crecería por encima de N. Como se vió en el análisis de complejidad de **topdown**, éste depende fuertemente de la cantidad máxima de elementos que pueda haber q, y puesto que $|q| \in O(N)$ no es muy difícil ver que en estas condiciones **topdown** es $O(N^5)$. Entonces cada iteración del ciclo principal de **zigzag** es $O(N^6 + N^5)$. Además puesto que cada iteración disminuye estrictamente el tamaño de g (sino termina) y

el tamaño inicial es N (porque $q = \{\}$), entonces puede haber a lo sumo N iteraciones. Por lo tanto **zigzag** es $O(N^7)$.

3.5. Algunas notas

Hay un par de sutilezas que no fueron mencionadas sobre estos algoritmos. Por un lado existe una buena cantidad de subespecificación en las elecciones que realizan los algoritmos.

En el algoritmo de reordenación pueden existir multiples caminos de longitud mínima entro los cuales se debe eligir uno cualquiera. Esto quiere decir que podrían existir varias gramáticas distintas entre sí con el mismo tamaño que compartiesen el mismo conjunto de constituyentes.

Es interesante hacer notar que de este hecho se desprende facilmente que la cantidad de soluciones al problema de la gramática mínima es finita. Puesto que la cantidad de conjuntos de constituyentes es finita, entonces la cantidad de aquellos que reordenan a una gramática mínima también es finita, y como los caminos mínimos dentro de un grafo son finitos, entonces la cantidad de gramáticas que son solución es finita.

En términos prácticos la cantidad de gramáticas mínimas que se pueden construir con el mismo conjunto de constituyentes es enorme, siendo 10^{800} una cantidad de gramáticas común. Este hecho es importante y sera utilizado y discutido más adelante en el Capítulo 4.

En los algoritmos **bottomup**(y **topdown**) el criterio para resolver los empates en tamaño esta subespecificado puesto que la forma en que se resuelven depende del orden en el que se itera sobre los elementos de $C(q$ respectivamente) y este no esta especificado en el algoritmo. También aquí, los empates suelen ser abundantes y además determinantes del camino que sigue la búsqueda.

Además, tanto **bottomup** como **topdown** (y por lo tanto **zigzag**) agregan constituyentes no sólo cuando la mejora en tamaño es estrictamente positiva sino también cuando la mejora en tamaño es nula. Esta es simplemente la diferencia entre un $\leq$ y un $<$, sin embargo, es por esto que antes de que **zigzag** termine tipicamente existe un buen número de constituyentes que pueden ser agregados o quitados sin cambiar el tamaño de la gramática. Es decir que todas esos conjuntos de constituyentes tiene el mismo tamaño y solo uno de ellos debe ser elegido para generar la gramática resultante. En nuestra implementación se opta arbitrariamente por elegir el conjunto que tenga la menor cantidad de constituyentes, sin embargo esta decisión esta subespecificada y cualquier otro criterio también sería correcto.

Dependiendo de la aplicación todas estas decisiones mencionadas podrían resultar lo suficientemente relevantes como para dejarlas ser resueltas con un criterio arbitrario, por ejemplo, en compresión de datos la frecuencia de un símbolo afecta su probabilidad, la cual a su vez afecta la forma óptima

de codificarlo, siendo preferible tener menor cantidad de símbolos de mayor frecuencia. O como se verá más adelante, si la aplicación es el descubrimiento de patrones y cada una de estas decisiones ofrece patrones distintos ¿cual de todas es la decisión correcta?

Además de la subespecificación y sus posibles consecuencias existen otros puntos que vale la pena hacer notar. Como se menciono en un parraffo anterior **bottomup** y **topdown** pueden ser implementados buscando constituyentes con ganacias estrictas en tamaño o aceptando también constituyentes con ganancias nulas. Es punto es bastante suceptible de ser discutido y tiene las siguientes facetas interesantes:

En primer lugar, existe una ventaja teórica si se utiliza $<$ en vez de $\leq$. Si se repasa el análisis de complejidad de **zigzag** se puede ver que un punto fundamental de este es la afirmación de que el tamaño disminuye estrictamente en cada iteración de **bottomup** + **topdown**. Si se utiliza $<$ resulta que el tamaño disminuye estrictamene cada vez que se agrega o se quita un constituyente, lo cual sólo puede ocurrir N veces, por lo tanto, en este caso, el orden de complejidad de **zigzag** se reduce a $O(N^6)$.

Sin embargo, en el sentido práctico, usar $\leq$ resulta útil aun cuando "aumenta" el orden en un grado. En lo que respecta a tiempos de ejecución reales no existe demasiada diferencia entre usar $<$ o $\leq$, puesto que la parte más lenta de la ejecución (el **bottomup** inicial) es practicamente idéntica en ambas aproximaciones. Sin embargo, usar $\leq$ permite que la etapa de **topdown** tenga una mayor variedad de constituyentes con los cuales reemplazar los que va eliminando, esto permite a **zigzag** continuar su exploración aun más de lo que lo hubiese hecho utilizando $<$, y empiricamente ha mostrado obtener gramáticas de menor tamaño siempre y cuando uno este dispuesto a esperar un poco más.

Hay también algunos puntos de vista interesantes acerca de cómo recorre **zigzag** el reticulado. Uno podría ver a **zigzag** como trazando un recorrido por el reticulado, subiendo y bajando en él de forma zigzageante, y naturalmente surgirían algunas preguntas sobre este recorrido:

¿Cual es su máxima longitud? Cada etapa **bottomup** puede agregar a lo sumo $N/2$ constituyentes y lo mismo sucede con **topdown**, es decir que en cada iteración de **zigzag** se recorren a lo sumo N nodos del reticulado. Puesto que el tamaño debe disminuir estrictamente en cada iteración y el tamaño inicial es N entonces una cota a la longitud del recorrido máximo es N^2.

¿Puede recorrer todo el reticulado? Existen ejemplos pequeños para los cuales se puede forzar a zigzag a recorrer todo el reticulado, por ejemplo: $1ab2ab3ab4$ tiene solo un contituyente posible (ab), el reticulado explorado tiene solo dos nodos y **zigzag** los explora a ambos. El argumento de la pregunta anterior nos dice que la longitud del recorrido esta acotada superiormente por N^2, mientras que el tamaño del

reticulado es 2^M donde $0 \leq M \leq \frac{N^2-N}{2}$ es la cantidad de posibles constituyentes. Cuando $N \to \infty$ es posible encontrar casos en donde $2^M > N^2$ y para estos casos **zigzag** no recorre todo el reticulado.

¿Cuántas veces se puede pasar por un mismo nodo? Se puede pensar que cada elemento del reticulado tiene un puntaje dado por el tamaño de la gramática mínima que se puede armar con su conjunto de constituyentes, este puntaje es fijo y no cambia durante la ejecución de **zigzag**. Durante una ejecución, si el tamaño de la gramática actual es N' entonces los nodos del reticulado con tamaño superior a N' no pueden ser visitados puesto que **zigzag** debe siempre disminuir o mantener el tamaño de la gramática actual. Esto quiere decir que a medida que **zigzag** disminuye el tamaño de la gramática más y más zonas del reticulado no pueden ser visitadas. Un nodo con puntaje N' puede ser visitado una vez por **bottomup**, y luego otra vez por **topdown**. Si luego de la etapa **topdown** el puntaje disminuye a N'' entonces $N' > N''$ y el nodo con puntaje N' se ya no puede ser visitado por **zigzag**. Si en cambio luego de **topdown** el puntaje no disminuye entonces **zigzag** termina. En ambos casos el nodo sólo fue visitado dos veces. Por lo tanto cualquier nodo del reticulado puede ser visitado a lo sumo dos veces por **zigzag**.

3.6. Evaluación y resultados

En esta sección se reportan los tamaños de las gramáticas que se obtuvieron corriendo los algoritmos **LZ78**, **Sequitur**, **Greedy** y **Zigzag** sobre el Canterbury Corpus [2] y sobre el corpus usado históricamente para compresion de ADN compilado por Manzini y Rastero en [14]. La elección de los corpus se debe a que son estándares utilizados por la comunidad de compresión de datos, la que más ha aportado a la solución del problema de la gramáticas mínimas. Por lo tanto, esta elección nos permite compararnos con resultados previos y con trabajos futuros.

En las filas los Cuadro 3.1 y 3.2 se reportan los tamaños de las gramáticas obtenidas para cada secuencia utilizando los algoritmos mencionados en las columnas. En negrita se muestran los tamaños que son los más pequeños conocidos para estas secuencias. Por último, en la columna Mejora se indica en que porcentaje **zigzag** mejora el tamaño de la menor gramática conocida previamente y se calcula como $100 \times (\frac{|G|_{previo}}{|G|_{zigzag}} - 1)$.

Lamentablemente no se pudo ejecutar **zigzag** sobre tres de los archivos del Canterbury corpus: lcet10.txt, plrabn12.txt y ptt5. En los tres casos las características de la secuencia llevan a **zigzag** a un caso de peor complejidad algoritmica, ya sea por tener demasiados constituyentes(ptt5) o porque la cantidad de constituyentes en la solucion es demasiado grande(lcet10.txt y plrabn12.txt).

Secuencia	Longitud	lz78	sequitur	greedy	**zigzag**	Mejora
grammar.lsp	3721	4225	1770	1473	**1465**	0.5 %
xargs.1	4227	5309	2329	2006	**1972**	1.7 %
fields.c	11150	11056	4108	3416	**3311**	3.1 %
cp.html	24603	22658	9835	8048	**7767**	3.6 %
sum	38240	35056	15329	12207	**12027**	1.4 %
asyoulik.txt	125179	102296	44123	37474	**35000**	7.05 %
alice29.txt	152089	116296	49147	41000	**37001**	10.8 %
lcet10.txt	426754	288250	112205	**90099**	-	-
plrabn12.txt	481861	338762	142656	**124198**	-	-
ptt5	513216	106456	55692	**45135**	-	-
kennedy.xls	1029744	365466	174585	166924	**166660**	0.01 %

Cuadro 3.1: Resultados sobre el Canterbury Corpus

Secuencia	Longitud	lz78	sequitur	greedy	**zigzag**	Mejora
humdyst	38770	24568	11720	11065	**10078**	9.79 %
humprtb	56737	34484	16042	14885	**13659**	8.68 %
humhdab	58864	35708	16823	15312	**13993**	9.42 %
humghcs	66495	39776	15557	12938	**12031**	7.53 %
humhbb	73308	43572	20044	18704	**17024**	9.86 %
mtpacga	100314	56236	26039	24555	**22188**	10.66 %
chmpxx	121024	65828	30315	28705	**26022**	10.31 %
chntxx	155844	86412	40131	37884	**33940**	11.62 %
mpomtcg	186609	103092	46659	44176	**39911**	10.68 %
vaccg	191737	102868	46045	43706	**39296***	11.22 %
hehcmv	229354	124336	56200	53695	**48291***	11.19 %

Cuadro 3.2: Resultados sobre el corpus histórico de ADN. Los resultados marcados con * son resultados parciales y estan sujetos a una posible disminución en tamaño

Sobre los dos últimos archivos del Cuadro 3.2 vaccg y hehcmv y (marcados con *) se reporta el tamaño encontrado por una ejecución parcial de **zigzag** puesto que en el momento de escritura de este trabajo estas ejecuciones no habían llegado a finalizar tadavía. Por las propiedades de **zigzag** los tamaños finales de las gramáticas que encuentre sólo pueden ser menores o iguales a estos tamaños reportados.

Como se puede apreciar en los Cuadros 3.1 y 3.2, **zigzag** consistentemente obtiene una mejora sobre el previo estado del arte y en casos puntuales obtiene mejoras de entre el 5 % y el 10 %. Por lo tanto **zigzag** se ubica actualmente en el estado del arte en lo que respecta a aproximar gramáticas mínimas.

Es importante hacer notar que el costo computacional requerido por

zigzag es altisimo. Para dar un ejemplo, a **greedy** le toma no más de 20 segundos construir una gramática para cualquiera de los archivos más grandes de este corpus usando solo un procesador, mientras que a **zigzag** le toma entre 5 y 15 dias (a veces más) terminar usando 8 procesadores.

Esta diferencia aplastante en los tiempos de ejecucion entre greedy y **zigzag** responde a que fueron diseñados con dos necesidades distintas, por un lado los algoritmos que fueron diseñados para compresión esperan encontrar una gramática pequeña en un tiempo razonable, y el tiempo es un factor esencial. En cambio, en un algoritmo diseñado para el descubrimiento de patrones el factor esencial es obtener una gramática lo más pequeña posible.

En el caso que la complejidad computacional sea un factor decisivo, existe un algoritmo hermano a **zigzag** llamado **IRCOO** el cual también se basa en el algoritmo de reordenación y ofrece un tiempo de ejecución mucho menor a cambio de encontrar gramátcas ligeramente más grandes. Por referencias de **IRCOO** ver [5] a ser presentado en la conferencia LATA 2010.

Capítulo 4

Descubrimiento de patrones

4.1. Motivación

Intuitivamente hablando, una gramática mínima que codifica a w tiene entre sus constituyentes a palabras que se repiten con alta frecuencia y la regularidad con la que se repiten esta dada por la estructura jerárquica de la gramática. El hecho de qué una gramática sea pequeña fuerza a los no terminales a acaparar la mayor cantidad de ocurrencias posibles, lo cual da una base estadística sobre la que se podría sostener que el no terminal es en realidad una regularidad de la secuencia. Además, la estructura jerárquica de la gramática podría revelar patrones mucho más complejos, que involucran no solo a palabras, si no a la forma en que las palabras se combinan entre si en w, tal vez dando una pista de la posible interpretación o significado de las palabras.

La idea de minimizar el tamaño de un modelo con la esperanza de capturar una estructura subyacente no es nueva, algunos trabajos (como por ejemplo [3] [4] [17] [9]) intentan capturar una estructura que explique un fenómeno observado combinando un modelo de los datos con una inferencia guiada por el principio de descripción mínima (MDL).

El trabajo que más explicitamente publicitó la posibilidad de utilizar gramáticas mínimas para el descubrimiento de patrones fue el de Nevill-Manning y Witten [16] (y menos difundido en [15] pero con explicaciones más extensas). En estos trabajos se describe el uso las gramáticas inferidas para el descubrimiento de patrones en áreas tan diversas como el lenguaje natural, partituras musicales y secuencias de ADN. Las conclusiones de su trabajo sobre el descubrimiento de patrones fueron ampliamente citadas desde su publicación como una importante motivación para el estudio de las gramáticas mínimas por investigadores en esa área .

Por otro lado, el análisis de resultados realizado por Nevill-Manning es casi en su totalidad cualitativo y sobre pequeñas secciones de los resultados las cuales fueron elegidas por el autor (no al azar) luego de realizados los

experimentos. Esto plantea algunas dudas sobre la estabilidad de sus conclusiones. El éxito expuesto por Nevill-Manning podría provenir de bondades del subconjunto de resultados que se decidio mostrar, o de las subjetividades propias del evaluador para medir la "calidad" de los patrones.

Además, como se comentó en los capítulos anteriores cada gramática compacta puede caracterizarse con un único árbol de derivación, entonces la estructura que se puede inferir con una gramática mínima esta limitada a la estructura de un árbol. ¿Como se comportaría la estructura inferida en una gramática mínima si la estructura subyacente que generó la secuencia es más expresiva que un árbol de derivación? ¿Existiría una similitud estructural entre ambas? ¿Y si para inferir correctamente la estructura subyacente hiciera falta utilizar más que una secuencia?. Y aun cuando el lenguaje subyancente fuese el de una gramática compacta ¿Acaso las sucesivas aproximaciones decrecientes en tamaño convergerían suavemente a la estrucura de la gramática mínima? ¿O son todas radicalmente distintas entre sí?. Todo esto deja un espacio sin completar en la pregunta ¿es factible utilizar gramáticas mínimas para el descubrimiento de patrones?

4.2. Hipótesis

Para poder responder la pregunta "¿es factible utilizar gramáticas mínimas para el descubrimiento de patrones?" de forma objetiva se diseñaron experimentos que no requieren una evaluación cualitativa humana y que son independientes del dominio del conocimiento al cual se atribuye la secuencia. La idea esencial detras de los experimentos es poner a prueba la estabilidad de los patrones encontrados simulando diversos métodos para aproximar una gramática mínima.

En esta sección se dará la hipótesis y la definición de patrón que se usarán en este capitulo.

4.2.1. Dos científicos

Se plantea la siguiente situación hipotética: Un científico (A) decide buscar patrones en una secuencia w asumiendo la siguiente hipótesis "Una gramática pequeña revela patrones en w", y para hacer esto diseña un algoritmo de aproximación y lo ejecuta. Simultaneamente, otro científico (B) decide buscar patrones en la misma secuencia w, asumiendo la misma hipótesis, pero diseña otro algoritmo de aproximación y lo ejecuta. Dado que las gramáticas que ambos encuentran son comparables en tamaño (un diferencia $< 1\,\%$) la pregunta que nos hacemos es ¿Qué tan similares son los patrones encontrados por A con los encontrados por B?.

La idea detrás de esta situación hipótetica es determinar que tanto se puede confiar en que un patrón determinado se pueda encontrar con una gramática pequeña, puesto que esto afecta a la estabilidad de los resultados.

Si la estructura de las gramáticas pequeñas convergiese suavemente a la estructura de la mínima a medida que el tamaño disminuye sería de esperar que, suponiendo que se encuentran gramáticas lo suficientemente pequeñas, el científico A y el científico B compartan una buena parte de la estructura inferida.

La situación hipotética de los dos científicos constituirá la hípotesis de trabajo de este capitulo. Dicho explicitamente: Se busca es analizar la factibilidad de que A y B puedan lograr un acuerdo en los patrones que encuentran.

Para verificar esto, como se explicará más adelante, se propone simular las búsquedas hipotéticas de los dos científicos muestreando varios pares de gramáticas de tamaño pequeño para corroborar si estas comparten patrones en común o no.

4.2.2. Definición de patrón

Puesto que la definición de patrón es difusa y puede variar de trabajo en trabajo y para lograr generalidad en los resultados que se presenten en este capítulo nos centraremos en dos atributos fundamentales de la información contenida en las gramáticas encontradas que deberían ser tenidos en cuenta por cualquier búsqueda de patrones:

- **Las palabras** Una gramática compacta tiene en sus constituyentes a secuencias que bien podrían ser equiparadas con palabras, puesto que se trata de secuencias que ocurren de forma indivisible, que se repiten frecuentemente y que ocurren alternadamente en distinto orden a lo largo de la secuencia original. Es por esto que en este trabajo el conjunto de constituyentes de la gramática fue elegido como el conjunto de las palabras que se pueden identificar en la gramática.

- **La estructura** Esto es la jerarquía de constituyentes de la gramática y los lugares en la secuencia original donde quedan ubicadas las palabras. Como ya se dijo anteriormente, las gramáticas compactas tienen un único árbol de derivación. Además, puesto que este árbol es suficiente para caracterizar unívocamente a la gramática compacta a la cual esta asociado, la estructura de este árbol esta fuertemente ligada a la estructura de la grámatica que lo genera. Por esto es que en este trabajo se eligió al árbol de derivación como indicador de la estructura de la gramática.

Para su uso en el resto de este Capítulo se define a los patrones de una gramática como su conjunto de constituyentes y su árbol de derivación.

Para poder comparar cada uno de estos dos aspectos entre dos gramáticas distintas se utilizarán las métricas definidas en la Sección 2.4 y se explicará como hacerlo en la Sección 4.3.1.

4.3. Armado experimental

4.3.1. Métricas

A continuación se explicará como se decidió comparar los conjuntos de constituyentes y los árboles de derivación entre dos gramáticas.

Comparar las palabras Para medir la similitud entre los conjuntos de constituyentes encontrados q_1 y q_2 por dos gramáticas c_1 y c_2 se reportará $Jacc(q_1, q_2)$ y $F_1(q_1, q_2)$, donde $Jacc$ y F_1 son las métricas definidas en la Sección 2.4. Ambas métricas reportan números entre 0 y 1, que indican disimilitud total en 0 y similitud total en 1. F_1 suele ser presentada como un porcentaje.

Comparar la estructura Para comparar los árboles de derivación de dos gramáticas se utilizará la "Unlabeled bracket F-measure" que es una métrica usada por la comunidad de parsing. Esta métrica ampliamente aceptada se utiliza para sintetizar $Prec$ y Rec de un árbol de parseo sobre una oración. Se puede encontrar una buena descripción de esta en la tesis doctoral de D. Klein [12].

Surge de la necesidad de definir una métrica para parseo que considere resultados parcialmente correctos y se puede utilizar para medir similitud en cualquier estructura que divida jeŕarquicamente una secuencia. Puesto que los árboles de parseo son idénticos a los árboles de derivación, utilizaremos esta métrica como una forma de comparar la estructura de dos gramáticas. Para poder definir ésta métrica es necesario definir antes lo qué es el conjunto de brackets. Dado el árbol de parseo de una gramática compacta c, el conjunto de brackets sin etiqueta se define como los pares (i, j) de índices en w tales que el span de cualquier nodo del árbol es exáctamente la subsecuencia de w contenida entre los índices i y j.

En términos de gramáticas compactas el conjunto de brackets sin etiqueta $B(c)$ se define como los pares (i, j) de indices en w donde comienza y termina la expansion de cada no-terminal. Por ejemplo, la siguiente gramática

$$S \rightarrow aBBAAa$$
$$A \rightarrow abBa$$
$$B \rightarrow bab$$

Tendría representación en árbol como en la Figura 4.1 y su conjunto de brackets $B(c)$ es $\{(0, 20), (1, 4), (4, 7), (7, 13), (9, 12), (13, 19), (15, 18)\}$. La información que representan los brackets es la ubicación de las palabras de la gramática en la secuencia original. El acuerdo entre los brackets de dos gramáticas de alguna forma indica el acuerdo en la ubicación de las palabras, y de esta forma, un acuerdo en la estructura.

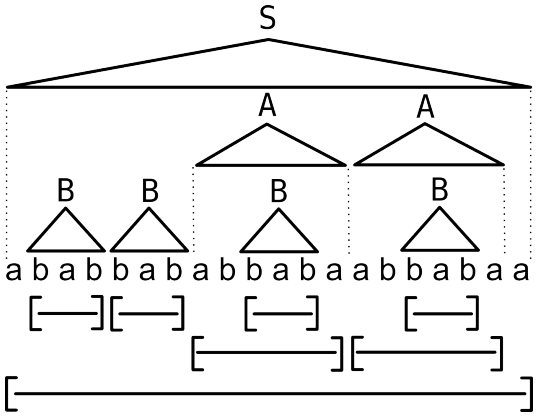

Figura 4.1: Representación en árbol de una gramática y sus brackets

Entonces, para comparar la estructura de dos gramáticas c_1 y c_2 se define unlabeled bracket F-measure como:

$$UF_1(c_1, c_2) = F_1(B(c_1), B(c_2))$$

Donde F_1 es tal como fue definida en la Sección 2.4. Esta será la medida reportada para identificar similitud o disimilitud entre la estructura de las gramáticas c_1 y c_2.

Utilizando estas métricas dos para comparar las palabras y la estructura de dos gramáticas diversas se logra una apreciación objetiva de la posible similitud entre gramáticas pequeñas encontradas.

4.3.2. Método de muestreo

Como se mencionó en las secciones anteriores se calcularán algunas metricas sobre pares de gramáticas pequeñas. Sin embargo nada se dijo aún sobre cómo se obtenendrán estas gramáticas pequeñas. En esta subsección se explicará y justificará el método utilizado para muestrear pares de gramáticas pequeñas con el fin de compararlas utilizando las métricas mencionadas con anterioridad.

Idealmente se debería usar un método que permita muestrear con probabilidad uniforme los elementos del conjunto de las gramáticas de tamaño menor al 1 % de la más pequeña conocida. Encontrar tal método no es trivial. Para empezar, el único algoritmo que puede construir gramáticas tan pequeñas es **zigzag**, el cual es muy intensivo computacionalmente como para producir muestras grandes y además no puede ser adaptado para que muestree el conjunto de las gramáticas pequeñas de manera uniforme.

Es decir, existe una imposibilidad práctica para muestrear gramáticas con estas carácteristicas. Es por esto que para responder a la hipótesis del

acuerdo posible entre los dos científicos se diseñaron dos métodos de muestreo complementarios que, a pesar de no muestrear uniformemente a las gramáticas pequeñas, proveen un buen indicador de dos extremos que se podrían encontrar si la situación de los dos científicos fuese real.

Método 1: Constituyentes iguales Como se explico en la Sección 3.5 una vez que se encuentra un conjunto de constituyentes existen muchas formas de construir una gramática de tamaño mínimo con él. Esto se debe a que por cada regla existen multiples caminos de longitud mínima en su grafo de reordenación. La cantidad de gramáticas distintas que se pueden construir con un mismo conjunto de constituyentes es típicamente superior a 10^{600}. Todas estas gramáticas tienen el mismo tamaño y comparten los constituyentes, es decir la F-measure del conjunto de constituyentes es 100 % para cualquier par de gramáticas. Sin embargo todas son estructuralmente distintas entre sí, es decir que la unlabeled bracket F-measure entre dos gramáticas distintas nunca es 100 %.

Volviendo a la situación de los dos científicos, supongamos que los dos son tan afortunados de encontrar exactamente el mismo conjunto de constituyentes. Luego de esto cada uno de ellos elige una gramática de entre todas las posibles con probabilidad uniforme, entonces surge naturalmente la pregunta: ¿Cuál es el acuerdo en la estructura de las gramática que podrían esperar tener?

Entonces, este muestreo se logró con el siguiente procedimiento:

1. Fijar un conjunto de constituyentes q_{ref} que corresponda al de una gramática que sea el estado del arte en aproximación de la gramática mínima.

2. Tomar K pares de gramáticas del conjunto de todas las gramáticas que se pueden construir con los constituyentes de q_{ref} y al hacerlo tomarlas con probabilidad uniforme dentro de ese conjunto.

Luego, para cada uno de estos K pares se debería calcular el valor de la unlabeled bracket F-measure y tomar la media muestral, la desviación estándar muestral, el valor máximo y mínimo de estos valores. Luego, si K es lo suficientemente grande, la media muestral obtenida es un buen estimador del valor de UF_1 esperado entre las gramáticas que propondrían ambos científicos.

El método usado para muestrear una gramática con probabilidad uniforme fijando el conjunto de constituyentes no es trivial y se explica en detalle en el Anexo A.4. Se decidió no incorporar aquí los detalles de este método para facilitar la lectura en este capítulo.

Es importante notar que, como ya se explicó, usando este método de muestreo no tiene sentido calcular el acuerdo entre constituyentes puesto que este siempre es del 100 %.

Método 2: Constituyentes diversos Opuestamente al método de constituyentes iguales, lo que se plantea aquí es intentar obtener conjuntos de constituyentes lo más disímiles posible antes construir gramáticas con ellos. Es decir que las gramáticas que se buscaría muestrear podrían tener tamaño distinto (pero comparable, ambas deberían ser pequeñas), la F-measure de los conjuntos de constituyentes no sería 100 % y por lo tanto la unlabeled bracket F-measure tampoco.

En la situación hipótetica de los dos científicos esto sería equivalente a que ambos hayan encontrado gramáticas de tamaño similar, tal vez con diferencias de tamaño menores al 1 % pero hayan sido tan desafortunados que los conjuntos de constituyentes que encontraron son bastante disímiles entre sí. Si luego ambos construyen una gramática con su conjunto de constituyentes las preguntas que surgen son: ¿Que tan distintos pueden ser los conjuntos de constituyentes entre sí? ¿Que tan distintas estructuralmente pueden ser las gramáticas que construyen?

Para forzar la busqueda de gramáticas pequeñas y distintas este muestreo se logró con el siguiente procedimiento:

1. Buscar una gramática pequeña llamada c_{ref} usando **zigzag**.
2. Elegir aleatoreamente un subconjunto de los constituyentes de c_{ref}
3. Buscar otra gramática c_i usando **zigzag**, pero inhibiendo la inclusion de los constituyentes elegidos en el paso anterior
4. Volver a 2 K veces.

Como se puede ver, al prohibir el uso de constituyentes de c_{ref} se fuerza encontrar gramáticas que compartan pocos constituyentes con c_{ref}. De esta forma se puede generar una muestra $(c_{ref}, c_1), (c_{ref}, c_2) \ldots, (c_{ref}, c_K)$ con K pares en la que se compara la gramática más pequeña conocida (la de **zigzag**) con otras.

Es claro que no hay garantía de que este método produzca gramáticas pequeñas, sin embargo en la práctica ha resultado bastante exitoso, a veces encontrando gramáticas más pequeñas que la original c_{ref} encontrada con **zigzag**.

4.3.3. Corpus

Debido al gran coste computacional que implica correr el algoritmo de **zigzag** el abanico de archivos de corpus factibles de ser utilizados es bastante pequeño. El límite práctico de tamaño, en el momento de la escritura de este trabajo, es una secuencia de aproximadamente 120.000 símbolos, aunque en realidad, este límite esta mucho más ligado a la cantidad y características de los constituyentes que a la longitud de la secuencia.

Por esto, un criterio importante cuando se elegieron las secuencias fue el tiempo computacional usado por **zigzag** en ellas. Se decidió evaluar en archivos que contengan lenguage natural escrito y secuencias en de ADN. Para ello fueron elegidas tres secuencias de ADN y una secuencia conteniendo lenguaje natual (Inglés).

A continuación se dara una pequeña descripción de cada una de las secuencias elegidas:

ADN: humdyst Es una secuencia con 38770 símbolos de longitud usando un alfabeto de 4 símbolos distintos que contiene ADN de la proteina llamada distrofina presente en los músculos humanos. Pertenece al corpus histórico utilizado para medir compresion de datos en ADN compilado por Manzini y Rastero en [14].

ADN: humhdab Es una secuencia con 58864 símbolos de longitud usando un alfabeto de 4 símbolos distintos que contiene ADN que perteneciente a 3 cósmidos presentes en el humano: HDAB, HDAC, HDAD. También es parte del corpus histórico utilizado para medir compresion de datos en ADN compilado por Manzini y Rastero en [14].

ADN: mtpacga Se trata de una secuencia con 100314 símbolos de longitud usando un alfabeto de 4 símbolos distintos que contiene ADN mitocondrial y también pertenece al corpus histórico utilizado para medir compresion de datos en ADN compilado por Manzini y Rastero en [14].

Lenguaje natural: asyoulik Se trata de una secuencia de 125179 símbolos de longitud usando un alfabeto de 68 símbolos distintos que contiene el texto (en Inglés) de una obra de William Shakespeare titulada "As You Like It". Esta secuencia es parte del Canterbury Corpus [2], el cual ya fue utilizado en el Capítulo 3. Lamentablemente esta es la única secuencia de lenguaje natural que fue posible análizar debido al tiempo que consume hacerlo.

4.4. Resultados experimentales

A continuación se presentará una serie de cuadros reportando toda la información obtenida en los experimentos.

En el Cuadro 4.1 se da una descripción básica del corpus informando para cada secuencia su longitud, la cantidad de posibles constituyentes, el tamaño de la gramática que encuentra **zigzag**, la cantidad de constituyentes de esta gramática, la cantidad de posibles gramáticas que se pueden construir fijando el conjunto de constituyentes al que encuentra **zigzag** (esto es importante para el muestreo con constituyentes iguales) y por último la cantidad de

gramáticas diversas encontradas para la misma secuencia (este número es el K definido en el método de muestreo con constituyentes diversos).

En el Cuadro 4.2 se reportan los resultados del método de muestreo con constituyentes iguales. Para cada secuencia el conjunto de constituyentes que se fija para muestrear las gramáticas es el que encuentra **zigzag**.

En los Cuadros 4.3, 4.4, 4.5 y 4.6 se reportan los resultados del método de muestreo con constituyentes diversos. Cada línea tiene la comparación entre la gramática encontrada por **zigzag** c_{ref} y una gramática c_i muestreada con constituyentes diversos. Sus respectivos conjuntos de constituyentes son q_{ref} y q_i. La columna *tamaño* es el tamaño de c_i. *dif* es la diferencia de tamaño entre c_{ref} y c_i en porcentaje. $Jacc(q_{ref}, q_i)$ es el indice de Jaccard entre los conjuntos de constituyentes. $F_1(q_{ref}, q_i)$ es la F-measure entre los conjuntos de constituyentes. Finalmente $UF_1(c_{ref}, c_i)$ es la unlabeled bracket F-measure entre los árboles de derivación de c_{ref} y c_i.

Secuencia	humdyst	humhdab	mtpacga	asyoulik
Longitud de la secuencia	38770	58864	100314	125179
Cant. de constituyentes	29908	69062	96609	152547
Tamaño de c_{ref}	10078	14041	22188	35000
Tamaño de q_{ref}	554	810	1178	2391
Gramáticas usando q_{ref}	10^{602}	2×10^{813}	10^{1458}	7×10^{968}
Gramáticas muestreadas	10	14	11	11

Cuadro 4.1: Descripciones básicas de las secuencias. c_{ref} es la gramática que encuentra *zigzag* y q_{ref} son sus constituyentes. La cantidad de gramáticas usando q_{ref} son todas las que se podrían muestrear con constituyentes iguales. La cantidad de gramáticas muestreadas es la cantidad que se usa para el muestreo con constituyentes diversos.

Secuencia	humdyst	humhdab	mtpacga	asyoulik
Media muestral de F_1	62.42 %	65.05 %	54.84 %	81.65 %
Tamaño de la muestra	1000	1000	1000	1000
Desviación e. muestral	1.26 %	1.28 %	1.30 %	1.26 %
F_1 mínima muestreda	58.17 %	61.59 %	50.84 %	78.14 %
F_1 máxima muestreda	66.55 %	69.30 %	59.55 %	85.38 %

Cuadro 4.2: Muestreo con constituyentes iguales. F-measure media en gramáticas de tamaño mínimo construidas usando el conjunto de constituyentes que encuentra *zigzag* sobre cada secuencia.

Tamaño	Dif.	$Jacc(q_{ref}, q_i)$	$F_1(q_{ref}, q_i)$	$UF_1(c_{ref}, c_i)$
10057	0.208 %	0.186	31.312 %	**12.928 %**
10071	0.069 %	0.182	30.866 %	**9.521 %**
10086	0.079 %	0.173	29.487 %	**12.632 %**
10108	0.298 %	0.115	20.690 %	**8.223 %**
10115	0.367 %	0.119	21.250 %	**8.146 %**
10119	0.407 %	0.113	20.271 %	**7.941 %**
10128	0.496 %	0.097	17.652 %	**7.712 %**
10164	0.853 %	0.097	17.726 %	**6.029 %**
10191	1.121 %	0.090	16.453 %	**4.281 %**
10430	3.493 %	0.000	0.000 %	**0.000 %**

Cuadro 4.3: Muestreo con constituyentes diversos en humdyst. Cada linea reporta la comparación de una gramática muestreada c_i con la gramática c_{ref} que encuentra *zigzag*. Sus respectivos conjuntos de constituyentes son q_i y q_{ref} Dif. expresa la diferencia de tamaño en porcentaje entre c_i y c_{ref}.

Tamaño	Dif.	$Jacc(q_{ref}, q_i)$	$F_1(q_{ref}, q_i)$	$UF_1(c_{ref}, c_i)$
13985	0.399 %	0.217	35.644 %	**13.798 %**
14025	0.114 %	0.225	36.689 %	**15.404 %**
14026	0.107 %	0.241	38.834 %	**14.388 %**
14029	0.085 %	0.238	38.481 %	**14.774 %**
14039	0.014 %	0.200	33.293 %	**12.313 %**
14052	0.078 %	0.216	35.588 %	**12.843 %**
14052	0.078 %	0.254	40.470 %	**17.877 %**
14065	0.171 %	0.217	35.706 %	**13.943 %**
14068	0.192 %	0.228	37.166 %	**13.918 %**
14070	0.207 %	0.222	36.307 %	**13.910 %**
14077	0.256 %	0.213	35.170 %	**14.238 %**
14106	0.463 %	0.239	38.614 %	**16.572 %**
14233	1.367 %	0.226	36.897 %	**11.180 %**
14243	1.439 %	0.233	37.752 %	**13.444 %**

Cuadro 4.4: Muestreo con constituyentes diversos en humhdab. Cada linea reporta la comparación de una gramática muestreada c_i con la gramática c_{ref} que encuentra *zigzag*. Sus respectivos conjuntos de constituyentes son q_i y q_{ref} Dif. expresa la diferencia de tamaño en porcentaje entre c_i y c_{ref}.

Tamaño	Dif.	$Jacc(q_{ref}, q_i)$	$F_1(q_{ref}, q_i)$	$UF_1(c_{ref}, c_i)$
22192	0.018 %	0.152	26.357 %	**10.537 %**
22215	0.122 %	0.196	32.732 %	**12.141 %**
22224	0.162 %	0.165	28.388 %	**9.107 %**
22228	0.180 %	0.165	28.289 %	**9.615 %**
22230	0.189 %	0.138	24.248 %	**8.621 %**
22234	0.207 %	0.152	26.323 %	**10.694 %**
22251	0.284 %	0.134	23.611 %	**7.787 %**
22272	0.379 %	0.116	20.713 %	**7.400 %**
22299	0.500 %	0.112	20.164 %	**7.996 %**
22350	0.730 %	0.110	19.749 %	**6.622 %**
22418	1.037 %	0.061	11.543 %	**5.047 %**

Cuadro 4.5: Muestreo con constituyentes diversos en mtpacga. Cada linea reporta la comparación de una gramática muestreada c_i con la gramática c_{ref} que encuentra *zigzag*. Sus respectivos conjuntos de constituyentes son q_i y q_{ref} Dif. expresa la diferencia de tamaño en porcentaje entre c_i y c_{ref}.

Tamaño	Dif.	$Jacc(q_{ref}, q_i)$	$F_1(q_{ref}, q_i)$	$UF_1(c_{ref}, c_i)$
35132	0.377 %	0.675	80.604 %	**74.521 %**
35623	1.780 %	0.202	33.572 %	**33.123 %**
35688	1.966 %	0.221	36.223 %	**36.874 %**
35701	2.003 %	0.121	21.597 %	**21.717 %**
35712	2.034 %	0.115	20.668 %	**21.948 %**
35715	2.043 %	0.118	21.068 %	**22.526 %**
35723	2.066 %	0.119	21.294 %	**21.484 %**
35737	2.106 %	0.111	19.967 %	**20.878 %**
35754	2.154 %	0.113	20.311 %	**20.928 %**
35773	2.209 %	0.106	19.206 %	**17.738 %**
35811	2.317 %	0.109	19.723 %	**21.236 %**

Cuadro 4.6: Muestreo con constituyentes diversos en asyoulik. Cada linea reporta la comparación de una gramática muestreada c_i con la gramática c_{ref} que encuentra *zigzag*. Sus respectivos conjuntos de constituyentes son q_i y q_{ref} Dif. expresa la diferencia de tamaño en porcentaje entre c_i y c_{ref}.

4.5. Análisis de resultados

4.5.1. Muestreo con constituyentes iguales

Como se puede ver en el Cuadro 4.2 los valores de la F_1 media son muy distintos entre ADN y lenguaje natural. Para el caso de ADN el acuerdo en la estructura se centra cerca del mismo porcentaje (60 %) para las 3 secuencias. La agrupación en valores resulta llamativa si se considera que son secuencias de tamaño diverso que contienen información de objetos funcionalmente distintos.

En el caso del lenguaje natural el acuerdo estructural es bastante más alto, alcanzando el 81 % de F_1 media.

Para borrar dudas acerca de la precisión de los cálculos de las medias efectuados aquí se adjunta el siguiente razonamiento: Ya que la cantidad de pares muestreados es igual en todos los casos y la desviación estándar muestral es casi idéntica un simple test de hipótesis revela que con una probabilidad de 99.9 % la media de la población esta a menos de 0.14 % de la media muestral. Este análisis deja poca probabilidad a que las medias de la población sean significativamente distintas a las calculadas.

4.5.2. Muestreo con constituyentes diversos

Como se puede apreciar en los Cuadros 4.3, 4.4 y 4.5 las gramáticas encontradas para las secuencias de ADN tienen una diferencia de tamaño ínfima, siendo en la mayoría de los casos menores al 0.5 % y muestran una disimilitud fuerte en lo que respecta a la estructura ya que en casi todos los casos F_1 es menor al 15 %, lo cual es muy bajo.

Por otro lado el acuerdo en los constituyentes para ADN no es tan pequeño, pero sigue siendo sorprendentemente bajo, habitualmente cerca un 30 % de F_1 (un poco más en humhdab).

Podemos pensar que tiene sentido que el acuerdo sea mayor en los constituyentes que en la estructura por dos razones. La primera es que, intuitivamente hablando, parece más simple que los científicos acuerden qué cosa es una palabra de un lenguaje que cómo se organizan estructuralmente esas palabras, puesto que pueden hacerlo de muchas formas distintas en la misma secuencia. La segunda es que, basandose en los resultados, se puede notar en la columna de humhdab en el Cuadro 4.1 que puede haber hasta 10^{800} formas de elegir la estructura de la gramática cuando se fijan los constituyentes. Como consecuencia, se sabe que existen 10^{1600} pares de gramáticas que acuerdan en un 100 % de las palabras pero que no tienen un acuerdo total en la estructura. Entonces, parece natural pensar que si las gramáticas tienen bajo acuerdo en las palabras entonces menor acuerdo tendrán en cuanto a la estructura.

En el Cuadro 4.3 se puede observar en la última fila una muestra particular. Únicamente para esta gramática se decidio llevar al extremo el criterio

de constituyentes diversos y no permitirle a **zigzag** utilizar ningún constituyente de la gramática original c_{ref}. Como consecuencia de esto el acuerdo en palabras y estructura es nulo. El tamaño de esta gramática es superior a las otras encontradas, pero es menor en tamaño a la más pequeña conocida previamente. Siendo que en tamaño esta relativamente próxima a la menor gramática conocida para esta secuencia (un 3.5 %) y que el acuerdo es 0 % nos lleva a pensar que tal vez las sucesivas aproximaciones decrecientes en tamaño no convergen suavemente a la estructura de la gramática mínima (asumiendo que la gramática mínima no es mucho menor a la menor gramática conocida).

Para resumir lo observado en ADN y volviendo al caso hipotético de los dos científicos, esto querra decir que aunque ubiesen encontrado gramáticas tan pequeñas que solo difirieran en un 0.5 % del tamaño con el estado del arte, aun así podrían tener un acuerdo de solo el 14 % en la ubicación de las palabras en la secuencia (la estructura) y del 30 % en la elección de las palabras (los constituyentes).

En cambio en el caso de los experimentos sobre lenguaje natural (asyoulik.txt) los resultados son más difíciles de interpretar. En casi todas las gramáticas el acuerdo estructural también es bastante bajo, sin embargo están a un 2 % del tamaño de la menor conocida, que aunque no es tanto puede volver la comparación entre gramáticas poco válida. Esto es, si el científico A encuentra patrones en una gramática pequeña, dificilmente el científico B vaya a poder contradecirlos con una gramática un 2 % más grande argumentando exclusivamente en base al tamaño.

Curiosamente, existe una sola gramática con un tamaño similar a la encontrada por **zigzag** y el acuerdo estructural en esta es bastante alto (un 74 %), casi tan alto como se podría esperar si los constituyentes fuesen iguales, que es del 81 % según los datos del muestreo con constituyentes iguales.

Esto sugiere la idea de que al menos en el lenguaje natural tal vez existe una convergencia en la estructura de las gramáticas a medida que el tamaño disminuye, pero no se converge hacia una única estructura sino que se converge a algo similar a los resultados del muestreo aleatorio que se obtiene con constituyentes iguales. Tal vez sugiriendo un concepto de "estructura esperada", en el sentido estadístico de la esperanza, más que de una única estructura correcta.

Aun cuando la diferencia de tamaño en las gramáticas muestreadas es en casi todos los casos relativamente alta, tanto el acuerdo estructural como el acuerdo en los constituyentes es superior al que se encontro en ADN.

4.5.3. Algunas conclusiones

Un punto significativo es la diferencia de los resultados experimentales entre las gramáticas inferidas en ADN y en lenguaje natural. Esto da

evidencia empírica de que la efectividad de las gramáticas mínimas para identificar patrones varía significativamente cuando se cambia el dominio del conocimiento al cual pertenece la secuencia.

Otra reflexión luego de analizar estos datos es que existe un riesgo inegable de que las disimilitudes observadas puedan existir en secuencias pertenecientes a otros dominios del conocimiento. Es decir que si los dos cientifícos hipotéticos se basaran exclusivamente en el tamaño de las gramáticas para justificar los patrones encontrados entonces existe un riesgo razonable sus conclusiones no sean estables. Es por esto que, un trabajo que busque la aceptación científica de sus patrones utilizando gramáticas mínimas debera incluir en sus resultados algún tipo evidencia (por ejemplo un muestreo de gramáticas pequeñas) que sustente la estabilidad de sus patrones. Alternativamente, también se podría requerir la validación de sus patrones otorgada por expertos del dominio del conocimiento al que se atribuya la secuencia.

Capítulo 5

Conclusión y trabajo futuro

5.1. Conclusión

En este trabajo se exploró el problema de la gramática mínima y se analizo la factibilidad de utilizarlas para el descubrimiento de patrones.

Se encontró un algoritmo de optimización que ordena el espacio de búsqueda del problema al de los posibles conjuntos de constituyentes de las gramáticas. Sobre esta ordenación del espacio de búsqueda se mostraron varias propiedades y utilizandolas se diseño un algoritmo que mejora al previo estado del arte en la aproximación de gramáticas mínimas.

Con este nuevo algorimo se encontraron multiples gramáticas pequeñas, todas de menor tamaño que cualquier otra conocida para el material de entrenamiento usado, para ser usadas en una serie de experimentos que validen o no la factibilidad de su uso en el descubrimiento de patrones. Se vió que en el caso de secuencias de ADN existe un buen grado de incompatibilidad entre las estructuras de las gramáticas, sugiriendo un problema grave de inestabilidad en los patrones encontrados. En el caso de lenguaje natural se vió que el acuerdo estructural es un poco superior, pero quedan dudas razonables sobre el posible uso de las gramáticas mínimas para el descubrimiento de patrones. Finalmente, se concluye que para poder aceptar patrones como válidos no es suficiente basarse en gramáticas mínimas sino que es necesario contar con alguna evidencia de la estabilidad de los patrones (tal vez muestreando multiples gramáticas pequeñas) o hasta tal vez verificandolos por expertos del dominio del conocimiento al que se atribuye la secuencia.

5.2. Trabajo futuro

5.2.1. Gramáticas mínimas

El trabajo futuro en la aproximación de gramáticas mínimas continuando los lineamientos aquí dados es abundante. Surgen muchas ideas nuevas al

conteplar el problema desde el punto de vista que da la reordenación. Sin embargo, luego de este trabajo se empieza a poner en tela de duda la útilidad práctica de las gramáticas mínimas tal como están definidas.

Como quedo evidenciado, el descubrimiento de patrones podría no ser una buena justificación para estudiar el problema de la gramática mínima. ¿Qué queda entonces como justificación para su estudio? La compresión de datos. Sin embargo, durante el desarrollo de este trabajo se observó que gramáticas demasiado pequeñas podrían resultar impracticas también para la compresión de datos puesto que a medida que disminuye el tamaño podría aumentar la entropía de la secuencia total. Puesto que la entropía de una secuencia es directamente proporcional al tamaño de su codificación usando cualquier tipo de codificador estadístico podría darse que una gramática mínima sea subóptima a la hora de usarla para compresión de datos. De ninguna manera se esta afirmando con esto que la aproximación de gramáticas mínimas no es útil para la compresión de datos, sino que un estudio empirico más extenso podría ser un trabajo futuro interesante.

El eje de esta posible dificultad en la compresión es la definición de la función de tamaño dada en el problema de la gramática mínima. Si esta función considerára la entropía seguramente se obtendrían buenos resultados en la compresión, pero esto cambiaría radicalmente el problema a uno nuevo en el cúal se deberían desarrollar teoría y algorítmos nuevos. Explorar nuevas formas de definir la función de tamaño podría resultar en trabajos futuros también interesantes.

En el caso que la búsqueda de otros algoritmos de aproximación fuese deseable, un buen número de posiles caminos a explorar fueron considerados para superar el mayor problema de **zigzag**, su tiempo de ejecución. La complejidad computacional de **zigzag** para secuencias largas es enorme y dependiendo de la aplicación que se le de a la gramática resultante podría resultar impráctico utilizar este algoritmo. A continuación se explicarán otras aproximaciones menos costosas que fueron consideradas y algunas implementadas. Estás no fueron desarrolladas en profundidad por exceder los alcances de este trabajo sino que fueron anotadas como posible trabajo futuro.

La más simple consiste en asignar un puntaje a cada constituyente y luego agregar a un conjunto de constituyentes aquellos que tengan puntaje más altos. Luego se reordena este conjunto de constituyentes para construir una gramática, la cual es devuelta por el algorítmo. Usar esta estrategia tendría complejidad $O(N^3)$. Un puntaje símple que se podría aplicar es el de la ganancia máxima posible si se reemplazaran todas las ocurrencias de un constituyente en w . Por supuesto que también se podrían considerar puntajes mucho más sofisticados, que incluyan información importante como la superposición con otros constituyentes. Los pocos experimentos hechos con esta estrategia resultaron en gramáticas que se aproximan en tamaño a la de **zigzag** en un 3 % o 5 %.

También otra de las primeras alternativas exploradas fue hacer algorit-

mos híbridos. Esto es, tomar un algoritmo rápido, como greedy por ejemplo, ejecutarlo y luego ejecutar **zigzag** comenzando desde el conjunto de constituyentes que devolvió greedy. Es decir, tomar a **zigzag** como una etapa de refinamiento por sobre algoritmos existentes. Es interesante notar que **zigzag** se puede utilizar de esta manera junto con cualquier otro algortimo que aproxime la gramática mínima.

Otra un poco más compleja consiste en comenzar con un conjunto de constituyentes q vacío e iterar sobre todos los constituyentes posibles. Para cada constituyente c iterado, se lo agrega a q solo si mejora el tamaño (no importa en cuanto), pero si c ya pertenecia a q se considera quitarlo solo si mejora el tamaño(de nuevo, no importa en cuanto). De esta forma se itera repetidas veces sobre el conjunto de todos los constituyentes posibles hasta que luego de una cantidad determinada de constituyentes no haya habido mejora en el tamaño, entonces el algoritmo se detiene y devuelve una gramática que corresponde al q final. Esta estrategia permite muchas variantes dependiendo de los criterios que se utilizen, por ejemplo, cambiando el criterio para establecer el orden en el cual se itera sobre los constituyentes. Los experimentos realizados con esta estrategia mostraron bastante éxito, ejecutandose en tiempo razonable y alcanzdo cerca del 1 % (en dos caso mejorando) del tamaño alcanzado por **zigzag**.

Dada la cantidad propiedades que cumple este problema que todavia no se han aprovechado se observo en que prodrían existir optimizaciones para hacer la búsqueda de gramáticas pequeñas mucho más rapida. Por ejemplo, de encontrarse una forma de actualizar los grafos de reordenación y sus caminos cada vez que se agrega un constituyente en vez de reconstruirlos cada vez se podría reducir la complejidad de **bottomup** (y por ende la de **zigzag**) hasta $O(N^4)$ o $O(N^5)$. Esta posible mejora seguramente se lograría con un estudio previo sobre algorítmos para grafos dinámicos y luego adaptando alguno de estos a este problema en particular.

Además de los posibles trabajos futuros buscando mejorar la eficiencia se encontró que existen un buen número de preguntas teóricas aun sin responder. Entre ellas:

En otro trabajo [13] se probó que el problema de la gramática mínima es NP-Hard, sin embargo esta prueba es válida solo si se consideran alfabetos de tamaño arbitrario. En ese mismo trabajo se menciona que queda como pregunta abierta dilucidar si este problema es polinomial para alfabetos binarios. Puesto que en este trabajo se propone una reestructuración fuerte del espacio de búsqueda, tal vez este punto de viste ofrece ventajas para responder a la pregunta: ¿Es polinomial el problema de la gramática mínima para alfabetos binarios?.

Se mostro que la solución al problema de la gramática mínima esta dentro del reticulado que recorre **zigzag**. Tambien se vio que **zigzag** tiene subespecificado el criterio con el cual resuelve cual constituyente agregar en caso de empate. Si se piensa en una traza en forma de árbol con los posibles caminos

válidos que puede tomar **zigzag** se tendrá un conjunto de nodos posibles de ser alcanzados por **zigzag**. ¿Puede darse que la solución siempre pertezca a las hojas de este árbol?.

Otra pregunta de la índole de la anterior sería: Si partiendo del conjunto vacío se agregase un constituyente a la vez hasta llegar al conjunto de constituyentes que corresponde a la gramática mínima se tendría un camino de conjuntos de constituyentes. ¿Es cierto que todo camino a la gramática mínima tiene un tamaño decreciente?. La respuesta a esta pregunta podría llevar a un nuevo algoritmo mucho más eficiente para resolver el problema de la gramática mínima.

5.2.2. Descubrimiento de patrones

En este trabajo se exploro la factibilidad del descubrimiento de patrones basandose exclusivamente en dos atributos de las gramáticas: sus constituyentes y su árbol de parseo. Un trabajo futuro interesante es explorar en mayor detalle otros posibles patrones contenidos en las gramáticas pequeñas. Podrían existir otros atributos que no fueron considerados(o distintas formas de analizar los mismos) que puedan reportar información relevante para el descubrimiento de patrones.

Por ejemplo, una forma distinta de analizar la similitud entre conjuntos de constituyentes sería considerar medidas más finas de similitud, como comparar las subsecuencias asociadas a cada constituyente con la distancia de Levenshtein (o algún otra distancia de edición) en vez de una comparación iguales/distintas a la hora de decidir si los conjuntos de constituyentes se parecen.

Como se menciono en la Sección 4.5, algunos resultados muestran que a medida que se encuentran sucesivas gramáticas decrecientes en tamaño podría existir una convergencia a una *estructura esperada*, en el sentido estadístico de la esperanza, en vez de una convergencia a una estructura fija. Una profundización del concepto de estructura esperada podría revelar una forma más flexible de definir los patrones estructurales y es un trabajo futuro interesante.

Además, durante los experimentos se encontró que existe un subconjunto muy reducido de constituyentes que comparten todas las gramáticas pequeñas encontradas, este subconjunto no es lo suficientemente grande como para reflejar un impacto en el índice de Jaccard o la F-measure, sin embargo todas las gramáticas pequeñas los incluyen. Esta carácteristica hace a estos constituyentes muy llamativos como posibles patrones, aunque antes de iniciar un trabajo sobre esto valdría la pena asegurarse sobre la calidad de los mismos con expertos de dominio. En el caso de ser relevantes sería muy interesante considerar métodos alternativos para obtener este pequeñisimo núcleo compartido por las gramáticas que no requieran un cálculo tan intensivo. La principal dificultad de identificar este subconjunto es que

es demasiado pequeño en proporcion a la cantidad de constituyentes de las gramáticas, por ejemplo para mtpacga solo el 0.4 % de los constituyentes usados en las gramáticas están en todas las gramáticas muestreadas, y para asyoulik.txt el 5.3 %. Existe una posibilidad de que utilizando datos como la frecuencia de ocurrencia, la longitud del constituyente, la superposición con otros constituyentes y algunas técnicas básicas de machine learning se podría inferir un buen clasificador para determinar este subconjunto sin grandes necesidades computacionales.

Por otro lado, dado que se encuentra que las gramáticas compactas no responden como se esperaba al principio de descripción mínima sería interesante explorar otras formas representación de estructura y regularidad en secuencias, tal vez formas más expresivas que las gramáticas compactas y que tengan mejor comportamiento cuando se las usa junto con el principio de descripción mínima.

Pensamos que en lo que respecta al descubrimiento de patrones, la mejor forma de continuar este trabajo es buscar otras formas de estructurar una secuencia que sean distintas a las gramáticas libres de contexto, no sólo en expresividad, sino también en la forma en la que se las puede describir sinteticamente, ya que esto afecta directamente a la forma en la que el principio de descripción mínima opera sobre ellas.

Apéndice A

A.1. Gramáticas compactas

Aquí se prueba que una gramática libre de contexto que minimiza la función de tamaño dada en la Sección 2.2 es una gramática compacta.

Sea $G =< V, T, P, S >$ una gramática libre de contexto, por la definición del problema podemos asumir:

$$L(G) = \{w\} \tag{A.1}$$

Además asumiremos:

$$\text{si } w \in L(G) \text{ entonces } |w| > 1 \tag{A.2}$$

Puesto que se consideraran a los casos $w = \epsilon$ o $w = a$ luego. Ahora definiremos una forma más general de medir el tamaño de una gramática:

Definicion 2. *Una función $f : CFG \to \mathbb{Z}$ sera llamada una función de tamaño si:*

$$f = k_1|V| + (\sum_{A \to \alpha \in P} k_2|\alpha|) + k_3$$

para algunos $k_1, k_2, k_3 \in \mathbb{Z}$ con $k_1 > 0$ y $k_2 > 0$.

El caso $k_1 = 0$ fue deliveradamente dejado afuera de esta definición.

Finalmente supondremos que G es una gramática mínima que genera a w fijada una función de tamaño f.

$$\forall G' \in CFG \mid L(G') = L(G) \Rightarrow f(G) \leq f(G') \tag{A.3}$$

Ahora probaremos que G cumple con las propiedades dadas en la definicón de gramática compacta una por una.

Teorema A.1. *G No tiene símbolos inutiles.*

Proof: Por absurdo. Supongamos que G tiene símbolos inutiles. Es posible construir una G' sin símbolos inutiles tal que $L(G') = L(G)$ y por construcción $V' \subset V$ y $P' \subseteq P$. Entonces

$$
\begin{aligned}
& |V'| < |V| \ \wedge \ |P'| \leq |P| \\
\Rightarrow & \\
& k_1|V'| < k_1|V| \quad \wedge \sum_{A\to\alpha\in P'} k_2|\alpha| \leq \sum_{A\to\alpha\in P} k_2|\alpha| \\
\Rightarrow & \\
& k_1|V'| + \sum_{A\to\alpha\in P'} k_2|\alpha| \quad < \quad k_1|V| + \sum_{A\to\alpha\in P} k_2|\alpha| \\
\Rightarrow & \\
& f(G') \ < \ f(G)
\end{aligned}
$$

Lo cual contradice a (A.3). Por lo tanto G no tiene símbolos inutiles.

Si $A \in P$, Sea $E(A) = \{\alpha \in T^* | A \stackrel{*}{\Rightarrow} \alpha\}$. Entonces probaremos

Teorema A.2. $\forall A \in V \quad |E(A)| = 1$ *(Cada no-terminal deriva exactamente una secuencia de T^*)*

Proof: Si $|E(A)| = 0$ entonces A es inutil y por teorema A.1 un absurdo.
SI $|E(A)| > 1$ entonces $\exists \alpha, \beta \in E(A)$ con $\alpha \neq \beta$.
Como A es util $\exists \gamma_1, \gamma_2$ tal que

$$S \stackrel{*}{\Rightarrow} \gamma_1 A \gamma_2 \stackrel{*}{\Rightarrow} w = L(G)$$

Sean γ_1', γ_2' tal que $\gamma_1 \stackrel{*}{\Rightarrow} \gamma_1'$ y $\gamma_2 \stackrel{*}{\Rightarrow} \gamma_2'$ y $w = \gamma_1'\alpha\gamma_2'$ (α sin perdida de generalidad). Entonces también se puede derivar $S \stackrel{*}{\Rightarrow} \gamma_1 A \gamma_2 \stackrel{*}{\Rightarrow} \gamma_1'\beta\gamma_2' \neq w$ lo cual es una contradicción por (A.1)

Teorema A.3. $\forall A \to \alpha \in P \quad |\alpha| > 1$ *($|\alpha| > 1$ para cada $A \to \alpha \in P$)*

Proof: Por contradicción. Supongamos $\exists A \to \alpha$ con $|\alpha| \leq 1$. Si $A = S$ y $\alpha = \epsilon$ entonces $\epsilon \in L(G)$ lo cual es absurdo por (A.2). Si $A = S$ y $\alpha \neq \epsilon$ entonces no puede ser que $\alpha = B$ con $B \in V$ puesto que se podría construir una gramática equivalente sin B la cual sería de menor tamaño. Asi que debe ser que $\alpha = a$ con $a \in T$, pero esto también es un absurdo por (A.2). Entonces $A \neq S$. Ahora construiremos una nueva gramática $G' =< V', T, P', S >$ con

$$V' = V - \{A\}$$

$$P' = \{B \to \beta' \mid B \to \beta \in P \wedge \beta' = \text{“}\beta \textit{ reemplazando } A \textit{ por } \alpha\text{”}\}$$

Por que $S \in V'$ y por construcción se puede ver que $L(G') = L(G)$. Notar que la longitud de una β' es menor o igual que la de su β, y también $V' \subset V$, por lo tanto se puede ver que $f(G') < f(G)$, lo cual es absurdo.

Teorema A.4. $\forall A \in V \quad |\{A \to \alpha \in P\}| = 1$ *(Para cada elemento de V hay una única regla en P)*

Proof: Si $|\{A \to \alpha \in P\}| = 0$ entonces es absurdo porque constradice A.2. Si $|\{A \to \alpha \in P\}| > 1$ entonces $\exists \alpha, \beta$ con $\alpha \neq \beta$ tal que $A \to \alpha \in P$ y $A \to \beta \in P$. Ahora definiremos una nueva gramática $G' =< V, T, P', S >$ con

$$P' = P - \{A \to \beta\}$$

Cada derivación en G que usa $A \to \beta$ puede ser imitada en G' por una que usa $A \to \alpha$ porque (usando el teorema A.2) $A \Rightarrow \alpha \overset{*}{\Rightarrow} w$ y $A \Rightarrow \beta \overset{*}{\Rightarrow} w$. Entonces $L(G') = L(G)$ y porque el cuerpo de todas las reglas es mayor a uno (usando teorema A.3)

$$\sum_{C \to \gamma \in P'} k_2|\gamma| < \sum_{C \to \gamma \in P} k_2|\gamma|$$

Y entonces $f(G') < f(G)$ lo cual es un absurdo.

Teorema A.5. *Sea $w \in T^*$,$A, B \in V$. Si $A \overset{*}{\Rightarrow} w$ y $B \overset{*}{\Rightarrow} w$ entonces $A = B$ (Cada no-terminal deriva una secuencia distinta de T^*)*

Proof: Por absurdo. Supongamos $A \neq B$. Sea $G' =< V', T, P', S >$ con

$$V' = V - \{B\}$$

$$P' = \{C \to \gamma' \mid C \to \gamma \in P \wedge C \neq B \wedge \gamma' = \text{“}\gamma \textit{ reemplazando } B \textit{ por } A\text{”}\}$$

Claramente, cada derivación en G puede ser imitada por una derivación similar en G' usando solo A, entonces $L(G') = L(G)$. Además $V' \subset V$ y el largo de el cuerpo de todas las reglas son lo mismo en G' que en G, salvo porque hay almenos una menos, entonces

$$\sum_{A \to \alpha \in P'} k_2|\alpha| < \sum_{A \to \alpha \in P} k_2|\alpha|$$

y por lo tanto $f(G') < f(G)$ lo cual es absurdo.

De esta forma se ha visto que G cumple las condiciones de una gramática compacta. Es facil ver que para los casos $w = \epsilon$ y $w = a$ (que habian sido dejados de lado) las gramáticas que minimizan la función de tamaño tambien son gramáticas compactas. Solo queda enunciar el teorema.

Teorema A.6. *Toda gramática libre de contexto G que tiene una sola palabra en su lenguaje y que minimiza una función de tamaño (como se la definio aqui) es una gramática compacta.*

A.2. Grafos y gramáticas mínimas

Aquí se prueba que es posible entontrar la gramática mínima en tiempo polinomial si se fijan algunas condiciones sobre las posibles gramáticas. La idea es mantener fijos los constituyentes de la gramática y luego minimizar el cuerpo de las reglas que generan esos constituyentes usando el camino más corto en un grafo.

Sea T un conjunto de terminales, $V = \{V_1, V_2, \ldots, V_M\}$ un conjunto de variables y $\{Y_1, Y_2, \ldots, Y_M\}$ subsecuencias de w pertenecientes a T^* tal que $|Y_i| > 1$. Estas secuencias serán los constituyentes. Dada una gramática compacta c, $V(c)$, $T(c)$, $P(c)$, $S(c)$ serán las proyecciones sobre el conjunto de variables, terminales, reglas y el símbolo inicial respectivamente.

Sea $E_c(A) = \{\alpha \in T^* | A \underset{c}{\overset{*}{\Rightarrow}} \alpha\}$ la expansion de A en la gramática c.

$$\text{Sea } C = \{c \in GC \mid V(c) = V \wedge T(c) = T \wedge S(c) = V_1 \bigwedge_{1 \leq i \leq M} E_c(V_i) = \{Y_i\}\}$$

Esto es, el conjunto de todos las gramáticas compactas (no isomorfas) con constituyentes $Y_1, Y_2, \ldots, Y_M$. Notar que al hacer V_1 el símbolo inicial no se pierde generalidad. Además, supondremos que C no es vacio.

Siendo que gran parte de las gramáticas de C esta fijado, algunas propiedades interesantes se cumplen.

Lema A.7. *Dadas $c, c' \in C$ entonces si $\gamma \underset{c}{\overset{*}{\Rightarrow}} w$ con $w \in T^*$ entonces $\gamma \underset{c'}{\overset{*}{\Rightarrow}} w$*

Proof: Esto sigue naturalmente de la definición de gramática compacta que dice que cada no terminal deriva una única secuencia de T^* y el hecho que ambas gramáticas comparten las secuencias que cada no terminal genera.

Este lema nos permite ignorar cuál gramática en particular se esta usando con $\overset{*}{\Rightarrow}$ porque todas ellas (las de C) son equivalentes si la secuencia esta hecha de terminales. Ahora, para $1 \leq i \leq M$ definiremos:

$$P_i = \{\gamma \in (T \cup V)^* \mid \gamma \overset{*}{\Rightarrow} Y_i \wedge \gamma \neq V_i\}$$

Como el conjunto de todos los posibles cuerpos de regla para V_i. Notar que porque $\gamma \neq V_i$ y $Y_i > 1$ entonces $\gamma > 1$. Ahora sea γ_i^* el elemento de menor longitud de P_i y sea c^* una gramática compacta tal que

$$V(c^*) = V \ \wedge \ T(c^*) = T \ \wedge \ S(c^*) = V_1 \wedge$$
$$P(c^*) = \{V_i \rightarrow \gamma_i^* \text{ para todo } 1 \leq i \leq M\}$$

Teorema A.8. *c^* es la gramática de menor tamaño en C.*

Proof: Es evidente que $c^* \in C$ puesto que cumple exactamente con la definición de C. Además como $|\gamma_i^*| \leq |\gamma_i|$ para todo $\gamma_i \in P_i$ entonces

$$\sum_{1 \leq i \leq M} |\gamma_i^*| \leq \sum_{1 \leq i \leq M} |\gamma_i|$$

$$\Longrightarrow$$

$$k_1|V| + k_2 \sum_{1 \leq i \leq M} |\gamma_i^*| + k_3 \leq k_1|V| + k_2 \sum_{1 \leq i \leq M} |\gamma_i| + k_3$$

$$\Longrightarrow$$

$$f(c^*) \leq f(c) \text{ para cualquier } c \in C$$

Por lo tanto c^* es la menor gramática en C.

Sigue de este teorema que si resultara posible cálcular eficientemente los γ_i^* entonces también se podría construir la gramática c^* eficientemente. De esta manera hemos transformado el problema de encontrar la gramática mínima en C a encontrar las secuencias mas pequeñas para cada P_i. Usando la definición de P_i veremos como construir un grafo etiquetado G_i tal que cada camino en G_i (entre el nodo de comienzo y el de fin) es un elemento de P_i y que un camino de longitud mínima es también el elemento de longitud mínima en P_i.

Sea $N = |Y_i|$ y sea G_i un grafo etiquetado dirigido con $N + 1$ nodos $n_0, \ldots, n_N$ tal que

$$n_a \xrightarrow{A} n_{b+1} \iff A \stackrel{*}{\Rightarrow} Y_i[a : b]$$

Donde $Y_i[a : b]$ es la subsecuencia de Y_i entre los indices a y b, y A es un elemento de $(V \cup T) - V_i$. Esto es, el grafo tiene una arista por cada ocurrencia(en Y_i) de un terminal que va hasta el siguiente terminal, y tiene una arista por cada no terminal que va por sobre toda la longitud de su expansion. Llamaremos a n_0 el nodo de comienzo, a n_N el nodo de fin y un camino en G_i sera la secuencia de etiquetas usadas en las aristas del camino.

Teorema A.9. $\gamma \in P_i \iff \gamma$ *es un camino de comienzo a fin en* G_i

Proof: $\Longrightarrow$
Sea $\gamma = \alpha_1 \ldots \alpha_K$ with $\alpha_j \in (V \cup T) - \{V_i\}$.
Como $\gamma \stackrel{*}{\Rightarrow} Y_i$ entonces $\alpha_j \stackrel{*}{\Rightarrow} Y_i[a_j : b_j]$ para cada α_j y para algunos a_j, b_j.

Entonces $n_{a_j} \xrightarrow{\alpha_j} n_{b_j+1}$ es una arista de G_i.
Notar que $a_1 = 0$ y $b_K = N$, y como la expansión de α_j es seguida por la expansión de α_{j+1} entonces $b_j = a_{j+1}$ con $j < K$. Por lo tanto

$$n_0 = n_{a_1} \xrightarrow{\alpha_1} n_{a_2} \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_{K-1}} n_{b_K} = n_M$$

es un camino de comienzo a fin en G_i.

Proof: $\Longleftarrow$
Sea $\gamma = \alpha_1 \dots \alpha_K$ con $\alpha_j \in (V \cup T) - \{V_i\}$.
Como γ es un camino sabemos que $n_{a_j} \overset{\alpha_j}{\rightarrow} n_{b_j+1}$ esta en G_i para algunos a_j, b_j. Entonces $\alpha_j \overset{*}{\Rightarrow} Y_i[a_j : b_j]$.
También como es un camino $b_j = a_{j+1}$ con $j < K$ y $a_1 = 0$ y $b_K = N$.
Notar que este grafo es aciclico, entonces $a_1 < a_2 < \ldots < a_K$ por lo tanto

$$Y_i[a_1 : a_2]Y_i[a_2 : a_3] \dots Y_i[a_K : b_K] = Y_i[a_1 : b_K] = Y_i$$

Y entonces es posible derivar $\gamma \overset{*}{\Rightarrow} Y_i$ y por lo tanto $\gamma \in P_i$

Lema A.10. *Si γ es el menor camino de principio a fin en G_i entonces γ es la regla más pequena en P_i.*

Proof: Por absurdo. Supongamos que existe $\gamma' \in P_i$ tal que $|\gamma'| < |\gamma|$, entonces por teorema A.9, γ' es un camino en G_i. Pero entonces γ no es el menor camino en G_i lo cual es absurdo.

De esta forma hemos visto que solo son necesarias M búsquedas de caminos mínimos (una por cada P_i) para poder encontrar los γ_i^*. Como encontrar el camino mínimo en un grafo se puede resolver en $O(N^2)$, entonces, el problema de encontrar a c^* se puede resolver en $O(N^2M)$.

A.3. Constituyentes

En esta sección se describira el conjunto de constituyentes utiles para una gramática mínima c y luego se mostrara el algorimo utilizado en este trabajo para calcular este conjunto. En principio cualquier $\alpha \in T^*$ podría ser un constituyente de c, pero como c es una gramática compacta no puede tener símbolos inutiles, por lo tanto, si α no es una subsecuencia de w claramente el no termina que genera α es un símbolo inutil. Entonces los posibles constituyentes estan acotados por la cantidad de máxima de subsecuencias de w, las cuales nunca pueden superar $\frac{N^2-N}{2}$.

Además, otra condición de gramática compacta es que la longitud del cuerpo de todas las reglas debe ser mayor a 1, lo cual no podría darse si hubiese constituyentes de longitud 1. Además, si un no terminal se usa una sola vez en una gramática se podría construir una gramática de menor tamaño reemplazando la ocurrencia de este no terminal por su definición, lo cual nos dice que si un constituyente ocurre menos de dos veces sin superposición en w entonces no podría ser usado en c.

Por estos motivos definiremos el conjunto de constituyentes posibles como aqueyas subsecuencias de w que sean de longitud almenos 2 y que se repitan sin superposición almenos 2 veces. Existen un buen número de algoritmos que se pueden utilizar para calcular este conjunto eficientemente, sin embargo, dado que los constituyentes serán usados en la construcción

del grafo de reordenación hay algunos datos extra que sería util calcular. El grafo de reordenación agrega una arista por cada ocurrencia de un constituyente, es decir, que para construirlo es necesario conocer las ocurrencias de ese constituyente. Estas podrían ser calculadas cada vez que fuera necesario recorriendo linealmente a w, sin embargo, esto afectaría seriamente a la eficiencia (y al orden del algoritmo). Entonces, sería conveniente calcular las ocurrencias de cada constituyente solo una vez y luego reutilizarlas.

Entonces es necesario un algoritmo que no solo calcule las subsecuencias y su frecuencia sin superposición si no que también lleve nota de donde ocurre cada subsecuencia. Para cumplir estas metas se desarrollo un algoritmo a medida que, hasta donde el autor conoce, no existe en una publicación previa.

El algorimo se llama **cons** y la idea por detras es que dadas las ocurrencias de todas las subsecuencias de tamaño k es posible "extender" cualquiera de esas ocurrencias sumando el símbolo que sigue en w a esa ocurrencia y, de esa forma, obtendriamos una nueva ocurrencia de una subsecuencia distinta de tamaño $k+1$.

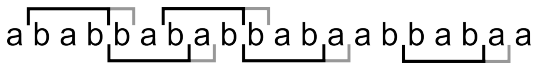

Figura A.1: Ocurrencias de "aba" y sus extensiones

Notar como en la Figura A.3 las 5 ocurrencias de "aba" se expanden en 2 ocurrencias de "abab" y en 3 de "abaa". Es facil ver entonces que cada constituyente de tamaño k solo puede extenderse a otros $|T|$ (el tamaño del alfabeto) constituyentes distintos de tamaño $k+1$. Además, la suma de las ocurrencias de estos nuevos constituyentes de tamaño $k+1$ es a los sumo igual a las del prefijo que los genera de tamaño k. Entonces el algoritmo funciona como una inducción, extendiendo todos los posbiles constituyentes de tamaño k en todos los posibles constituyentes de tamaño $k+1$. Al final de cada paso inductivo se filtran aquellos constituyentes que tienen frecuencia sin superposición menor a dos. Puesto que toda subsecuencia de tamaño $k+1$ tiene un prefijo tanto o mas frecuente de longitud k (k ¿0) el algoritmo es completo. El caso base puede ser facilmente calculado en tiempo lineal como todas las subsecuencias de longitud 2 y sus ocurrencias.

Dado que el alfabeto es fijo, C' se puede implementar como un arreglo y las operaciones "$C' := C' \cup \{p\}$" y "*anotar que p ocurre en* $(i, j+1)$" se pueden implementar en orden constante eligiendo inteligentemente los indices. Cada vez que se efectua "*anotar que p ocurre en* $(i, j+1)$" es posible contar en tiempo constante la frecuencia sin superposición controlando la distancia hasta la ultima ocurrencia sin superposicion. Además, notar que cada par (i,j) es analizado una única vez por el algoritmo, y es sabido que hay a lo sumo $\frac{N^2-N}{2}$ de estos pares. Por todo esto, **cons** puede encontrar todos los constituyentes, sus ocurrencias y su frequencia sin superposición

cons(w) :
$C := \{$*subsecuencias de longitud 2 y frecuencia sin superposición* ≥ 2 *en* $w\}$
for $c \in C$ **do**
    $C' := \{\}$
    for $(i, j) \in$ *ocurrencias*(c) **do**
        $p := w[i : j + 1]$
        anotar que p *ocurre en* $(i, j + 1)$
        $C' := C' \cup \{p\}$
    od
    $C := C \cup \{$ *elementos de* C' *con frecuencia sin superposición* $\geq 2\}$
od
return(C)

Figura A.2: Un algoritmo para encontrar los constituyentes y sus ocurrencias

con una complejidad de $O(N^2)$.

A.4. Conteo y muestreo fijando constituyentes

En este anexo se detallan los algoritmos utilizados para contar y muestrear las gramáticas de tamaño mínimo que se pueden construir a partir de un conjunto de constituyentes fijado.

Como se mostro en el Capítulo 4 la cantidad de gramáticas mínimas que se pueden construir con un mismo conjunto de constituyentes es enorme, por ejemplo en el caso de asyoulik.txt pueden ser tantas como 7×10^{968}. Tan grande es este número que crea dudas acerca de si fue correctamente calculado, es por eso que en este apéndice se daran pruebas que muestran que estos números son correctos.

La multiplicidad de gramáticas se debe a la multiplicidad de caminos mínimos en cada grafo de reordenación. Puesto que cada cada camino mínimo en el grafo de reordenación es una forma distinta pero de igual longitud de escribir cada regla, entonces da igual cual de todos estos caminos se usen puesto que el tamaño de la gramática no cambia.

Antes de empezar, se dara una pequeña justificación intuitiva que muestra lo simple que es alcanzar grandes números contando gramáticas mínimas. Dado un conjunto de constituyentes con 2391 elementos (como para asyoulik.txt) una gramática mínima para este debe tener 2391 reglas. Supongamos que para la mitad (1195) de estas reglas el grafo de reordenación tiene solo 2 caminos de longitud mínima, la cual es una suposición razonable según datos empiricos. Es decir que cada una de estas reglas tiene dos formas distintas pero equivalentes de ser escrita. Ahora, si pensamos que cada forma de elegir como se escriben estas reglas determina una gramática distinta,

entonces un simple análisis combinatorio nos dice que se pueden encontrar 2^{1195} gramáticas distintas solo aprobechando la combinatoriedad de la mitad de las reglas. Pero esto es solo un argumento empirico, a continuación se probaran algunos algoritmos.

Conteo de caminos mínimos Comenzaremos analizando la forma en la que se pueden contar los caminos de longitud mínima en un grafo de reordenación. En este punto se recomienda al lector releer la definición de grafo de reordenación dada en la Sección 3.2 También en esta sección se dijo que el camino mínimo se encontraba utilizando el algoritmo de Dijkstra, sin embargo cualquier otro algoritmo para encontrar un camino minimo serviría.

A continuación explicaremos un algoritmo para calcular la longitud del camino mínimo que puede ser facilmente extendido para poder contar la cantidad de caminos mínimos. Dado que un nodo n_j solo se puede ser alcanzado desde un nodo n_i con $i \leq j$ entonces la idea es conocer para cada $i \leq j$ la longitud del menor camino desde n_0 hasta n_i y utilizar esta información inductivamente para calcular la longitud del menor camino entre n_0 y n_{j+1}. Suponiendo que G es un grafo de reordenación con $k+1$ el procedimiento se detalla a continuación en la Figura A.3:

long_minima(G) :
$dist := array[k+1]$
$dist[0] := 0$
$j := 1$
while $j < k+1$ **do**
    $dist[j] = j+1$
    for $i \in \{i \mid n_i$ *tiene una arista que llega a* $n_j\}$ **do**
        if $dist[i] + 1 < dist[j]$:
            $dist[j] := dist[i] + 1$
        fi
    od
    $j := j + 1$
od
return$(dist[k])$

Figura A.3: Un algoritmo que calcula la longitud del camino mínimo

Es interesante notar que si existen empates en el algoritmo long_minima cuando se calcula la distancia hasta el nodo j estos son ignorados porque no afectan el valor de la distancia. Pero para poder contar todos los caminos de longitud mínima es necesario conocer estos empates, es decir, es necesario conocer desde cuáles nodos se puede llegar a un nodo j con un camino de longitud mínima. Asi que se propone en la Figura siguiente modificación a

long_minima para poder calcular desde cuales nodos se puede alcanzar a n_j con un camino de longitud minima.

empates_minimos(G) :
$empatados := array[k+1]$
$dist := array[k+1]$
$dist[0] := 0$
$j := 1$
while $j < k+1$ **do**
    $dist[j] = j+1$
    for $i \in \{i \mid n_i$ *tiene una arista que llega a* $n_j\}$ **do**
        if $dist[i]+1 < dist[j]$:
            $dist[j] := dist[i]+1$
            $empatados[j] := \{i\}$
        fi
        if $dist[i]+1 = dist[j]$:
            $empatados[j] := empatados[j] \cup \{i\}$
        fi
    od
    $j := j+1$
od
return$(empatados)$

Figura A.4: Un algoritmo que calcula empates en el camino mínimo

Tomando el arreglo *empatados* que devuelve empates_minimos se puede recorrer un camino de longitud mínima como sigue:

1. Asignar $i := k$.
2. Elegir asignar a i algún elemento del conjunto $empatados[i]$.
3. Mientras que $i \neq 0$ volver a 1.

De esta forma, los i visitados son los indices de los nodos en un camino de longitud mínima solo que el orden de camino esta invertido, es decir, el camino va de n_k a n_0. Entonces, *empatados* es un arreglo de conjuntos que identifica un subgrafo dentro del grafo de reordenación, en el cual todo camino entre n_0 y n_k es un camino de longitud mínima.

Usando el arreglo *empatados* es posible contar la cantidad de caminos entre n_0 y n_j de forma inductiva tal como se describe a continuación: Suponiendo que se conoce la cantidad de caminos de longitud mínima entre n_0 y n_i para todo $i \leq j$ entonces se puede calcular la cantidad de caminos de longitud mínima para n_{j+1} sumando la cantidad de caminos mínimos para todos los elementos de $empatados[j+1]$. El procedimiento se presenta a continuación en la figura A.5.

conteo_caminos(G) :
$empatados := empates\_minimos(G)$
$conteo := array[k+1]$
$conteo[0] := 1$
$j := 1$
while $j < k+1$ **do**
    $conteo[j] = 0$
    for $i \in empatados[j]$ **do**
        $conteo[j] := conteo[j] + conteo[i]$
    od
    $j := j + 1$
od
$return(conteo[k])$

Figura A.5: Un algoritmo que calcula la cantidad de caminos mínimos

El lector podría suponer que los empates son infrecuentes y por lo tanto la cantidad de caminos de longitud mínima es pequeña, sin embargo en general esto no es asi y para mostrarlo se presentara a continuación un ejemplo simple y elegante en el cual la cantidad de caminos de longitud mínima crece exponencialmente con la longitud de la secuencia.

Sea $aba(i)$ una secuencia tal que:

$$aba(i) = \overbrace{aba\, aba\, \ldots\, aba}^{i\ veces}$$

Ahora buscaremos ver cuantos caminos posibles existen en el grafo de reordenación para la secuencia $aba(i)$ cuando se utiliza como conjunto de constituyentes a $\{ab, ba\}$. Es claro que aba puede ser escrito tanto como $V_1 a$ o como aV_2, donde $V_1 \rightarrow ab$ y $V_2 \rightarrow ba$. Además ninguno de los constituyentes ocurre entre dos ocurrencias consecutivas de aba, entonces el camino mínimo en el grafo de reordenación es cualquiera que use a $V_1 a$ o aV_2 para cada ocurrencia de aba. Se puede elegir entre usar una u otra opción indistintamente para cada ocurrencia de aba y una vez que se eligio para todas las ocurrencias estas elecciones determinan una forma de construir un camino de longitud mínima. Puesto que cada elección es independiente una de otra, la combinatoriedad de estas elecciones da que hay 2^i formas distintas de realizarlas y entonces hay 2^i caminos mínimos distintos. Por lo tanto, para esta secuencia, la cantidad de caminos de longitud mínima crece exponencialmente con su longitud.

Conteo de gramáticas mínimas Como se explico en la Sección 3.2, una vez fijados los constituyentes de la gramática la forma en la que se construye

una regla es independiente de la forma en la que se construye otra. Es por esto que el algoritmo de reordenacion realiza tantas búsquedas de caminos mínimos como reglas haya para construir.

Si hubiese M reglas por construir y cada regla R_i tuviese K_i formas distintas de construirse con longitud mínima entonces todas las gramáticas mínimas que se podrían construir son aquellas que surgen de las diversas combinaciones de las formas de armar las reglas, es decir:

$$\textit{cantidad de gramaticas mínimas } = \prod_{i=1}^{M} K_i$$

Entonces, si se fija un conjunto de constituyentes q, la cantidad de gramáticas de tamaño mínimo que se pueden construir con q se pueden calcular siguiendo el procedimiento de la figura A.6.

contar_gramaticas(q) :
$q := q \cup \{w\}$
$c := 1$
for $c \in q$
    $g_c := \textit{grafo\_de\_reordenacion}(c, q)$
    $c := c \times \textit{conteo\_caminos}(g_c)$
end
return(c)

Figura A.6: Un algoritmo para contar la cantidad de gramáticas mínimas fijando un conjunto de constituyentes

Muestreo uniforme de caminos mínimos Para explicar cómo se realizo el muestreo uniforme de caminos mínimos en un grafo de reordenación primero se analizara un caso particular y luego se lo generalizara con una inducción.

Supongamos que se cuenta con el arreglo *conteo* que calcula conteo_caminos y con el arreglo *empates* que calcula empates_minimos. Los caminos de longitud mínima que llegan al nodo n_i solo lo pueden hacer a travez de los nodos de $\textit{empates}[i]$. Entonces dado un elemento $j \in \textit{empates}[i]$, ¿cual es la probabilidad de que un camino mínimo llegue a i a travez de j?. Si los caminos se muestrean uniformemente esta probabilidad es exactamente la proporción de los caminos que llegan a i por j sobre el total de caminos que llegan a i. El total de caminos que llegan a i es $\textit{conteo}[i]$ y la cantidad de caminos que pueden llegar a i por j esta dada por $\textit{conteo}[j]$, puesto que por cada camino que llega j, existe una unica forma de extenderlo para llegar a i, es decir, son la misma cantidad. Por lo tanto la probabilidad $P(j|i)$ de

que un camino mínimo que llega a i pase por j se puede calcular como

$$P(j|i) = \frac{conteo[j]}{conteo[i]}$$

Ahora, usando esta probabilidad es posible empezar por el nodo n_k y muestrear desde que nodo $i \in empates[k]$ llegaría un camino de longitud mínima si se los muestreara uniformemente. Este proceso se podría repetir para i inductivamente hasta que $i = 0$. De esta forma se pueden muestrear con probabilidad uniforme los caminos mínimos en un grafo de reordenación.

Muestreo uniforme de gramáticas mínimas Como se explico anteriormente la forma en la que se construye una regla es independiente de la forma en la que se construye otra una vez que se fija el conjunto de constituyentes. Por lo tanto, para muestrear una gramática mínima con probabilidad uniforme es suficiente con muestrar independientemente y también con probilidad uniforme los caminos que determinan la forma de construir cada una de las reglas.

Bibliografía

[1] Alberto Apostolico and Stefano Lonardi. Some theory and practice of greedy off-line textual substitution. In *Proc. Data Compression Conference, IEEE Computer*, pages 119–128. Society Press, 1998.

[2] Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In *DCC '97: Proceedings of the Conference on Data Compression*, page 201, Washington, DC, USA, 1997. IEEE Computer Society.

[3] Alvis Brazma, Inge Jonassen, Jaak Vilo, and Esko Ukkonen. Pattern discovery in biosequences. In *ICGI '98: Proceedings of the 4th International Colloquium on Grammatical Inference*, pages 257–270, London, UK, 1998. Springer-Verlag.

[4] Michael R. Brent, Claire Cardie, and Raymond Mooney. An efficient, probabilistically sound algorithm for segmentation and word discovery. In *Machine Learning*, pages 34–71, 1999.

[5] Rafael Carrascosa, François Coste, Matthias Gallé, and Gabriel Infante-Lopez. Choosing word occurrences for the smallest grammar problem. In *LATA 2010: Proceedings of the 4th International Conference on Language and Automata Theory and Applications (forthcoming)*, 2010.

[6] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and abhi shelat. Approximating the smallest grammar: Kolmogorov complexity in natural models. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 792–801, New York, NY, USA, 2002. ACM.

[7] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[8] John C. Kie Er and En hui Yang. Grammar based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46:2000, 2000.

[9] John A. Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198, 2001.

[10] En hui Yang and John C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform – part one: Without context models. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 46(3):755–777, 2000.

[11] P. Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272, 1901.

[12] Dan Klein. *The unsupervised learning of natural language structure.* PhD thesis, Stanford, CA, USA, 2005. Adviser-Manning, Christopher D.

[13] Eric Lehman and Abhi Shelat. Approximation algorithms for grammar-based compression. In *In Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 205–212. ACM/SIAM, 2002.

[14] Giovanni Manzini and Marcella Rastero. A simple and fast dna compressor. *Software - Practice and Experience*, 34:1397–1411, 2004.

[15] Craig G. Nevill-Manning. *Inferring Sequential Structure.* PhD thesis, University of Waikato, 1996.

[16] Craig G. Nevill-Manning and Ian H. Witten. Compression and explanation using hierarchical grammars, 1997.

[17] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Inf. Comput.*, 80(3):227–248, 1989.

[18] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[19] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding, 1978.