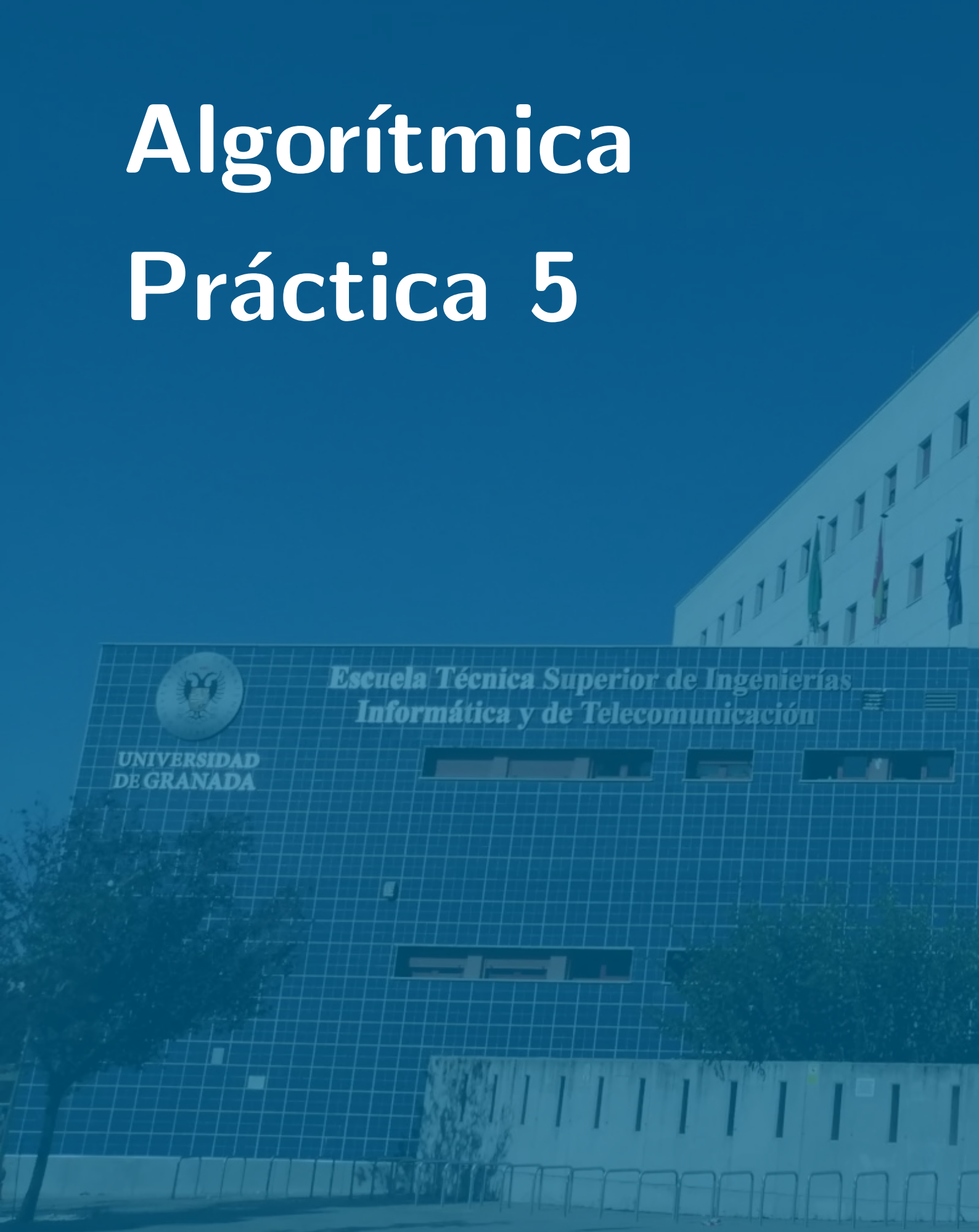
Algorítmica
Práctica 5



Los Del DGIIM, losdeldgiim.github.io

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Algorítmica
Práctica 5

Los Del DGIIM, `losdeldgiim.github.io`

Laura Mandow Fuentes
Chengcheng Liu
Daniel Hidalgo Chica
Roberto González Lugo
Elías Monge Sánchez

Granada, 2023-2024

Índice

1. Participación

- Laura Mandow Fuentes. e.lauramandow@go.ugr.es 100 %
- Roberto González Lugo. e.roberlks222@go.ugr.es 100 %
- Daniel Hidalgo Chica. e.danielhc@go.ugr.es 100 %
- Chengcheng Liu. e.cliu04@go.ugr.es 100 %
- Elías Monge Sánchez. e.eliasmonge234@go.ugr.es 100 %

2. Objetivos

En esta práctica nos disponemos a profundizar en las ideas tras el revolucionario algoritmo de Viterbi, además de comprender cómo y por qué las técnicas de la programación dinámica nos permiten una implementación clara y eficiente del mismo. Asimismo, demostraremos la eficacia de este algoritmo en diferentes problemas: a saber, en la producción de inferencias sobre cierto tipo de procesos estocásticos (Etiquetado gramatical o *'Part of speech tagging'*, entre otros) y para una suerte de problemas de caminos mínimos.

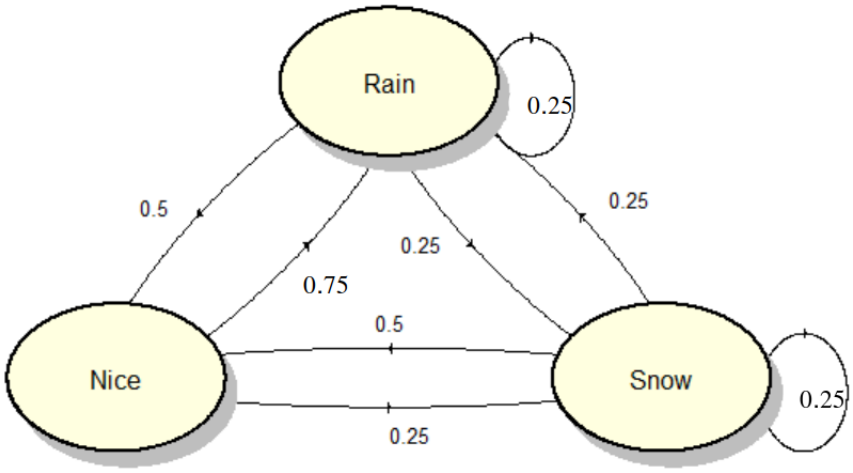
Para ello, presentaremos primero una motivación del algoritmo, en particular en el contexto de los Modelos Ocultos de Márkov, previamente introduciendo los conceptos y la terminología para su comprensión.

3. Introducción

3.1. Cadenas de Márkov y Modelos Ocultos de Márkov

Una cadena o proceso de Márkov es un modelo que describe una serie de estados o eventos, donde la probabilidad de que ocurra un evento o se transicione a un estado en cada instante depende únicamente del estado o evento anterior (nótese que esta propiedad, aunque aún informalmente, ya nos recuerda a las ideas de la programación dinámica sobre reducción de la información relevante en una secuencia de decisiones).

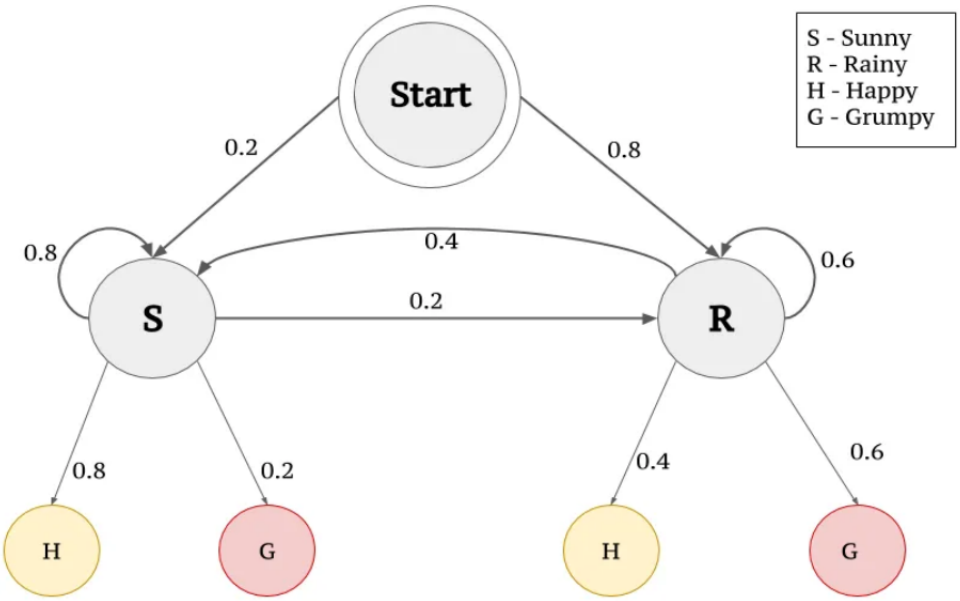
La idea queda clara con una imagen:



En este caso modelamos el tiempo que hace en un día determinado usando una Cadena de Márkov, donde los tres estados posibles son 'Lluvia', 'Buen tiempo' y 'Nieve'. Las ponderaciones entre las flechas que conectan los estados representan la probabilidad de, estando cierto día en el estado desde el que sale la flecha, que al día siguiente se pase al estado al que llega la flecha. A estas probabilidades las llamaremos 'probabilidades de transición'.

Presentamos ahora, con un ejemplo similar, los Modelos Ocultos de Márkov y el concepto de probabilidad de emisión.

Consideremos el siguiente modelo de Márkov que también representa el tiempo en un día determinado, pero donde hemos considerado solos dos estados posibles 'Rainy' y 'Sunny'. Además hemos añadido dos consecuencias posibles sobre el estado de ánimo de algún sujeto particular, María, durante un día en función únicamente del tiempo de ese día, con las probabilidades indicadas en las flechas de que el sujeto esté feliz 'Happy' o molesto 'Grumpy'.



Bien, proponemos ahora la siguente situación:

'María se ha mudado a Chile y no tenemos acceso a información sobre la meteorología del país por ninguna fuente. Sin embargo, por su manera de comunicarse con nosotros, notamos que María ha estado molesta durante tres días consecutivos.'

Recordando que el estado de ánimo de María depende únicamente del tiempo que hace en Chile, y que cuando hace sol tiene un 80 % de probabilidad de estar contenta y un 20 % de estar molesta, mientras que cuando llueve las probabilidades son de un 0.4 % y de un 0.6 %. Planteamos la siguiente pregunta: ¿No es más coherente inferir que, en principio, es más probable que halla estado lloviendo en Chile estos tres días?

Pues bien, nuestro propósito, y la cuestión que nos orientará hacia el uso del algoritmo de Viterbi, es, tan solo teniendo acceso a las consecuencias observables (en este caso, los estados de ánimo de María), tratar de inferir cuál es la secuencia de estados en la cadena de Márkov que los ha producido.

Antes de proponer un abordaje rudimentario a este problema, introduzcamos algo de terminología general sobre él. Los fenómenos observables dentro del modelo son llamados **estados observados**, y los estados de la cadena de Márkov a los que no tenemos acceso y que queremos adivinar son los **estados ocultos**. Un cierto estado oculto tiene una cierta probabilidad de producir un determinado estado observado: la **probabilidad de emisión**. En nuestro ejemplo, la probabilidad de emisión del estado "Happy"de María inducida por el estado oculto "Sunny"de la cadena de Márkov es de 0,8.

Pues bien, notemos que las probabilidades que queremos calcular, y de cual queremos extraer la más alta para saber cuál es la hipótesis más probable son:

$$
\begin{gathered}
P(G_1 \cap G_2 \cap G_3 \cap R_1 \cap R_2 \cap R_3) \\
P(G_1 \cap G_2 \cap G_3 \cap R_1 \cap R_2 \cap S_3) \\
P(G_1 \cap G_2 \cap G_3 \cap R_1 \cap S_2 \cap R_3) \\
P(G_1 \cap G_2 \cap G_3 \cap R_1 \cap S_2 \cap S_3) \\
\dots \\
P(G_1 \cap G_2 \cap G_3 \cap S_1 \cap S_2 \cap S_3)
\end{gathered}
$$

Donde los subíndices representan cada uno de los tres días o etapas del proceso de Márkov

Es decir, la probabilidad de que el causante del estado observado haya sido cada una de las posibles secuencias de estados ocultos posibles. Es claro que la mayor de estras probabilidades vendrá dada por el estado oculto que más probablemente haya causado la secuencia de estados observados, y entonces bastaría hacer estos cálculos para tener resuelto nuestro problema. Sin embargo, nos encontramos con un impedimento: la eficiencia. Veamos cómo calcularíamos una de estas probabilidades, por ejemplo la primera y aprovechemos para ver cómo las propiedades de las cadenas de Márkov simplifican mucho el problema.

$$
\begin{aligned}
P(G_1 \cap G_2 \cap G_3 \cap R_1 \cap R_2 \cap R_3) = P(R_1 \mid \text{ Start }) \cdot P(R_2 \mid R_1) \cdot P(R_3 \mid R_1 \cap R_2) \cdot P(G_1 \mid R_1 \cap R_2 \cap R_3) \cdot \\
P(G_2 \mid G_1 \cap R_1 \cap R_2 \cap R_3) \cdot P(G_3 \mid G_2 \cap G_1 \cap R_1 \cap R_2 \cap R_3)
\end{aligned}
$$

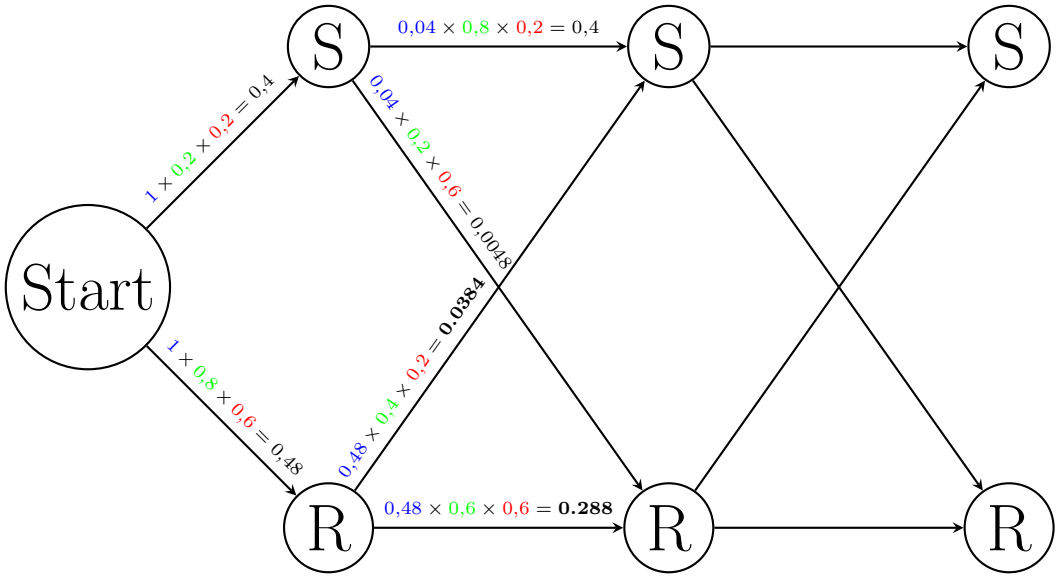
Ahora bien, usando que un estado en la cadena de Márkov depende **únicamente del estado anterior**:

$$
\begin{aligned}
P(G_1 \cap G_2 \cap G_3 \cap R_1 \cap R_2 \cap R_3) = P(R_1 \mid \text{ Start }) \cdot P(R_2 \mid R_1) \cdot P(R_3 \mid R_2) \cdot P(G_1 \mid R_1) \cdot \\
P(G_2 \mid R_2) \cdot P(G_3 \mid R_3)
\end{aligned}
$$

Pero en cualquier caso, teniendo en cuenta que en una situación con k estados posibles del proceso de Márkov y n observaciones tendríamos que efectuar k^n cálculos de probabilidades como esta, notamos que el problema es del orden $O(k^n)$ con respecto al número de observaciones, y muy rápidamente los costes computacionales resultan inasumibles: necesitamos de una herramienta que nos permita llegar a esa combinación de estados ocultos que nos proporciona la probabilidad máxima de manera más eficiente. Aquí entra el algoritmo de Viterbi.

4. Definición del Algoritmo

La idea detrás del algoritmo de Viterbi reside tras la propiedad de subestructuras optimales de la que gozan las soluciones a los problemas que aborda. Y es que notemos que una solución puede interpretarse como nada más que un recorrido por etapas entre los distintos estados ocultos, de esta manera:



En esencia, podemos ver que considerando cada estado como un nodo siempre obtenemos una estrucutura de grafo donde los nodos se organizan por columnas (cada columna hará referencia a una etapa) y cada nodo (estado) solo podrá conectarse con algún nodo de las columnas adyacences (referiendose a la transición de estados). Notemos que partiendo de un estado solo tiene sentido cambiar a otro estado cuando pasemos a otra etapa, así los nodos(estados) nunca pueden estar conectados con nodos que sean de la misma columna, puesto que no se considera la transición de estados en una misma etapa.

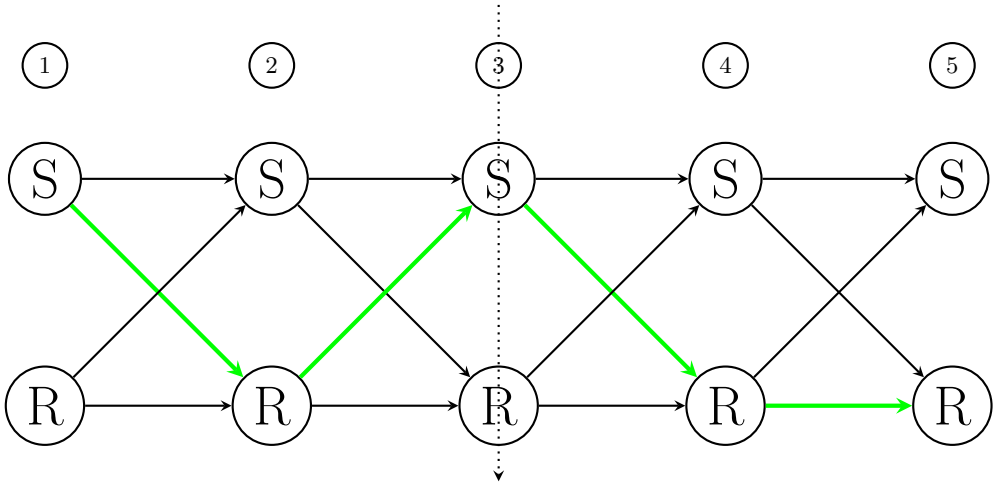
En cada estado oculto hemos calculado, usando los datos del diagrama, la probabilidad de que éste haya sido el causante del estado de ánimo de María ('Grumpy') en cada día. Notemos que al pasar de una etapa a otra lo único que hacemos es multiplicar la probabilidad de estar en el estado anterior por la probabilidad de transición al estado al que llegamos y por la probabilidad de emisión del estado observado 'Grumpy' en cada caso (0,2 si el estado oculto es 'Sunny' y 0,6 si el estado oculto es 'Rainy'). Se han marcado en cada paso en verde la probabilidad de transición y en rojo la probabilidad de emisión del estado que hemos observado. También, para los estados ocultos de la segunda etapa, se han encuadrado las probabilidades acumuladas máximas al llegar a ese estado de entre las dos posibles que hay: considerar la diferencia entre las probabilidades de llegada por diferentes caminos será fundamental más adelante.

De esta forma, vemos ya que nuestro problema, bajo este planteamiento, queda sintetizado en encontrar cuál es la secuencia de estados que tiene mayor probabilidad acumulada al llegar a la última de las 3 etapas del diagrama (una suerte maximización en el cálculo de caminos).

Estamos ya en condiciones de presentar la idea clave del algoritmo de Viterbi, que nos permite en cada paso descartar como imposibles ciertos caminos (secuencias de estados ocultos) en tanto que no podrán llevarnos a la secuencia con mayor probabilidad al final del diagrama, que es la que nos interesa.

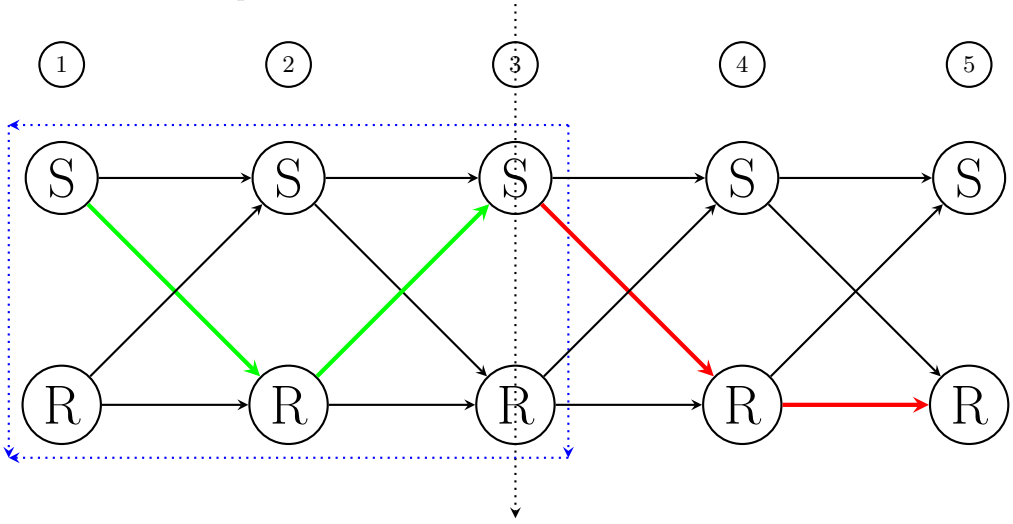
Para ello, observemos la probabilidad de subestructuras optimales de una solución al problema de optimización frente al que nos encontramos. Dado un camino óptimo de la primera a la quinta etapa en un problema análogo al ejemplo que venimos siguiendo, partámosla por la tercera, y consideremos los dos caminos que nos quedan a la derecha y a la izquierda (de la primera a la tercera etapa y de la tercera a la quinta respectivamente).

La idea queda clara con una imagen:



Es claro que, por ejemplo, el sub-camino derecho es la forma óptima (con mayor probabilidad acumulada) de llegar de la tercera a la quinta etapa, porque de ser otro el camino óptimo para ir de la tercera a la quinta etapa, tendríamos que podríamos formar un camino mejor que nuestra solución (total) original uniendo el camino de la parte izquierda (de la etapa 1 a la etapa 3) de nuestra solución original con el camino óptimo de la etapa 3 a la etapa 5 y tendríamos una contradicción con nuestra suposición de que partíamos de un camino óptimo de **1** a **5**. Por tanto, sabemos que cualquier solución que propongamos que pase por **S** en la etapa **3**, para intentar ser óptima, no puede más que hacer exactamente el camino marcado en verde hasta la última etapa.

Consideremos ahora que queremos construir una solución óptima pero tan solo sabemos la existencia del camino óptimo de **3** a **5** anterior.



Entonces, para encontrarla, estamos considerando cuáles de los caminos para llegar desde el comienzo hasta **S** en la etapa **3** es el que logra mayor probabilidad acumulada. Los caminos posibles son los siguientes:

- $S_1 \to S_2 \to S_3$
- $S_1 \to R_2 \to S_3$
- $R_1 \to S_2 \to S_3$
- $R_1 \to R_2 \to S_3$

Entonces, notemos que solo tenemos que continuar considerando uno de los 4 caminos anteriores, el que tenga mayor probabilidad acumulada al llegar hasta S_3, porque desde ese punto lo único que harán los caminos será seguir **todos el mismo** camino óptimo que existe (aunque no lo

conociésemos como en este caso) desde S_3 hasta R_5, y ya que para recorrer ese camino se avanzará multiplicando por unas cantidades fijas (las probabilidades de emsión y transición), será ese el que llegará al final con mayor probabilidad acumulada. Por tanto, podemos dejar de extender cualquier otro de los 4, pues no aspiran a ser la solución de nuestro problema.

De esta forma, en general, y aquí está **la clave** del Algoritmo de Viterbi, al llegar a cualquier nodo dentro del diagrama de estados ocultos que estamos recorriendo[1] tan solo tenemos que continuar (seguir calculando posibilidades) el camino que tenga mayor probabilidad acumulada de entre todos los que intersecan en ese nodo. Y esto es por la razón recién vista: todo camino que parta de ese nodo y aspire a ser la solución óptima, seguirá el mismo camino máximo (siempre existente, aunque fuera desconocido) desde ese nodo hasta el final del diagrama, pudiendo entonces ser solución óptima solo el camino que tenga mayor probabilidad acumulada en el momento de la intersección.

Estamos ya en condiciones, por tanto, de presentar la recurrencia que define el Algoritmo de Viterbi: determinar la máxima probabilidad de encontrarnos en un estado oculto s en la observación t-ésima (de entre todas las posibles secuencias de estados ocultos que pudieran habernos llevado a esa observación). De forma rudimentaria y quizá para algunos más clara: 'de entre todas las posibles secuencias de estados que desencadenan en el estado observado en la etapa t-ésima, consideramos cuál es la probabilidad que tiene aquella que más probabilidad tiene'.

Siendo S el conjunto de estados ocultos, π_s y $a_{r,s}$ las probabilidades iniciales y de transición respectivamente y $b_{s,o}$ la probabilidad de emisión de la observación o en el estado s, y teniendo una secuencia de T observaciones $o_0, o_1, \ldots, o_{T-1}$, nos definimos dos matrices de tamaño $T \times |S|$ de la siguiente manera:

$$P_{t,s} = \begin{cases} \pi_s \cdot b_{s,o_t} & \text{if } t = 0, \\ \text{máx}_{r \in S} \left(P_{t-1,r} \cdot a_{r,s} \cdot b_{s,o_t} \right) & \text{if } t > 0. \end{cases}$$

$$Q_{t,s} = \begin{cases} -1 & \text{if } t = 0, \\ \arg\text{máx}_{r \in S} \left(P_{t-1,r} \cdot a_{r,s} \cdot b_{s,o_t} \right) & \text{if } t > 0. \end{cases}$$

Donde en la matriz P guardamos las probabilidades ya explicadas (notemos que la toma de la probabilidad máxima en cada nodo es exactamente la poda de caminos posibles que se ha explicado anterioremente) y en la matriz Q guardamos simplemente en cada estado oculto al que llegamos, cuál es el estado desde el que llegamos, y que nos servirá para la reconstrucción del camino de probabilidad acumulada máxima al final del algoritmo.

Nótese que es ya totalmente claro que en el núcleo del Algoritmo de Viterbi residen las ideas de la programación dinámica: el principio de Optimalidad de Bellman sobre la suboptimalidad de particiones de una solución, y la reducción de la información para quedarnos tan solo con la relevante en cada caso (no nos interesa por qué camino hemos llegado a cierto estado oculto para tomar una decisión sobre el siguiente, sino tan solo cuál es la probabilidad acumulada al llegar hasta él). También esto se manifiesta en el carácter iterativo del algoritmo para rellenar la tabla de probabilidades que nos da finalmente la solución y la forma de reconstruir el camino, como podemos ver en la siguiente propuesta de pseudocódigo:

[1] multiplicando por probabilidades de transición y por probabilidades de emisión de estados observados en cada paso para encontrar el camino de probabilidad máxima

```
transProb <-- S x S transition probability matrix
emitProb <--- S x K emission probability matrix (K states of the observations)
obs <-- array of T observations

func Viterbi(states, initProb, transProb, emitProb, obs)
    prob <-- T x S matrix of 0's
    prev <-- T x S matrix of -1's

    // Inicializamos la probabilidad
    // de las primeras observaciones en los
    // primeros estados
    for each state s in states do
        prob[0][s] = init[s] * emit[s][obs[0]]

    // Para cada etapa t, guardamos
    // la probabilidad de alcanzar el estado
    // s considerando la máxima probabilidad
    // acumulada con la que se pueda llegar a
    // s desde cualquier nodo r
    for (t = 1; t < T; ++t)
        for each state r in states do
            for each state s in states do
                newProb = prob[t-1][r] * trans[r][s] *
                          emit[s][obs[t]]
                if newProb > prob[t][s]
                    prob[t][s] = newProb
                    prev[t][s] = r


    // Reconstruimos, metiendo como estado final
    // del camino aquel que termine con probabilidad
    // acumulada mayor
    path <-- empty array of size T
    path[T-1] <-- argmax_s (prob[T-1][s])

    // Y cada paso será el previo del siguiente
    for (t = T - 2; t >= 0)
        path[t] = prev[t+1][path[t+1]]
```

Notamos que, siendo clara la eficacia del algoritmo, obtenemos una eficiencia abrumadoramente mejor que en el cálculo por fuerza bruta, el particular, del orden de $O(T \times |S|^2)$

5. Ejemplos de uso

5.1. Cálculo de caminos mínimos por etapas

Veamos ahora cómo se usa el algoritmo de Viterbi para el cálculo de caminos mínimos entre dos nodos en un grafo mediano (Lattice Gragh) [2] donde además se exige que los nodos estén organizados por columnas y que cada nodo solo se pueda conectar con los nodos de las columnas vecinas adyacentes, es decir, no está permitido que se conecten nodos de una misma columna y tampoco nodos que no estén en columnas adyacentes. Este camino buscado también es llamado camino de Viterbi (*Viterbi path*).

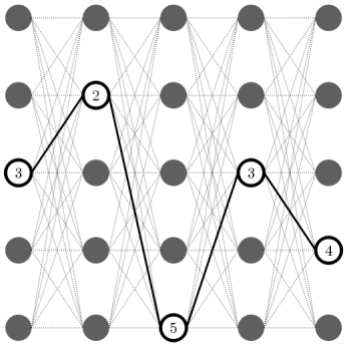


Figura 1: Ejemplo de grafo con nodos organizados por columnas

Podemos ver que esta estructura coincide con la de los grafos que usamos para representar las cadenas de sucesos modelados con un Modelo de Markov.

Aplicándolo a nuestro problema notemos que dado un grafo de este estilo, y dos nodos *start* y *end*, en nuestro problema no es restrictivo considerar única y exclusivamente las columnas comprendidas entre dichos dos nodos, y además, podemos ignorar el resto de nodos de las respectivas columnas a las que pertenecen dichos nodos *start* y *end*. También, está claro que debido a la estructura, para llegar de un nodo a otro necesariamente tenemos que pasar por cada una de las columnas entre medias una vez.

Teniendo en cuenta todo lo anterior, una forma sencilla de encontrar el camino sería listar todas los posibles caminos y quedarnos con el de mínimo coste, pero lo descartamos directamente por la ineficiencia, análogo razonamiento al realizado en la introducción. Aquí ya notamos la similitud con el ejemplo de la introducción.

Vamos ahora a desarrollar el razonamiento a partir de un ejemplo con pocos nodos (en la figura hemos obviado algunos caminos por limpieza):

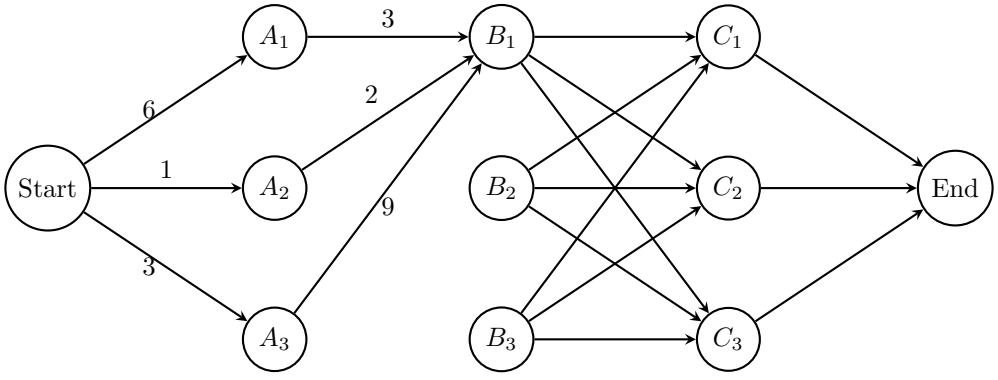


Figura 2: Ejemplo de grafo con nodos organizados por columnas

[2] Un grafo mediano en Matemáticas es un grafo no dirigido en que cualesquiera tres nodos a, b, y c tienen un único mediano. Un mediano (de a, b y c) es un nodo que pertenece a los caminos más cortos entre cualquier par de nodos a,b y c.

Por lo razonado anteriormente, como tenemos que pasar por cada una de las columnas A-B-C necesariamente, veamos con más detalle los primeros pasos.

De los caminos que empiezan en *start* y llegan a los distintos nodos de A, a priori, no podemos concluir con qué cámino sería el óptimo a tomar pues los costes posteriores podrían influir en nuestra decisión final, pero si nos fijamos en los nueve caminos que conducen a los distintos nodos de la columna B sí que podemos obtener información útil.

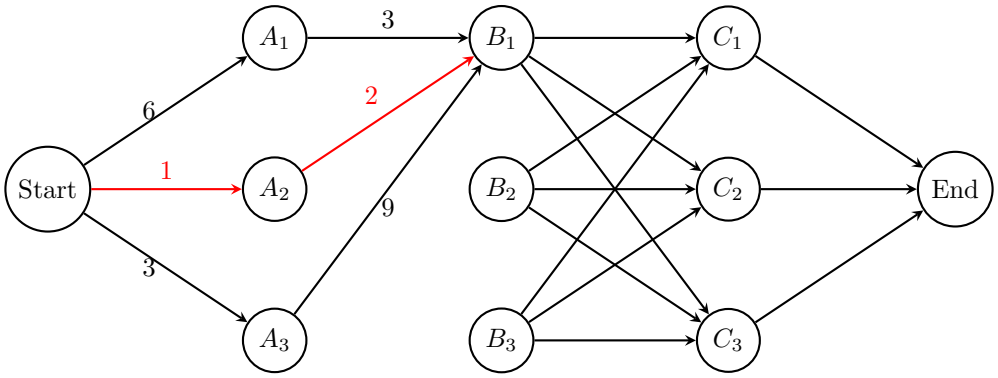
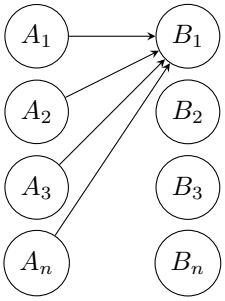


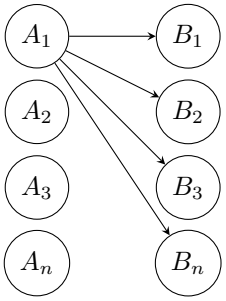
Figura 3: Camino mínimo a los nodos de la columna B

Empecemos fijándonos en particular en el nodo B_1. Para llegar a B_1 nos encontramos con que existen tres caminos que conducen a B_1, de estos tres sí que está claro qué cámino va a ser el óptimo a tomar para llegar a B_1, que en nuestro caso es la que pasa por A_2 (por ser el de coste mínimo). Por tanto, el resto de posibles caminos a B_1 los podemos descartar y a partir de entonces sabemos que todos los caminos parciales que surjan de B_1 hasta *end*, lo óptimo sería pasar por A_2.

Notemos también que la manera correcta de abordar los caminos ha de ser de $A \to B_i$ (forma a), en lugar de $A_k \to B$ (forma b). La forma b no es concluyente, es por esto que cuando consideramos los caminos desde *start* hasta los nodos de la columna A no pudimos concluir con nada.



(a) Forma a, correcta.



(b) Forma b, incorrecta.

En resumen, extrapolando a casos de tamaño mayor concluimos que basta con ir calculando el cámino mínimo para los nodos intermediarios para ir descartando posibles cáminos que ya sabemos que no son el óptimo y así alcanzando el camino mínimo buscado. Así, vemos que sigue al pie de la letra la lógica bajo el Algoritmo de Viterbi, luego es un caso válido donde se puede aplicar dicho algoritmo. La ligera diferencia a lo explicado anteriormente con los estados ocultos modelados con los Modelos de Markov es que, aquí tenemos costes de caminos a diferencia de probabilidades, pero en esencia siguen el mismo funcionamiento. Cuando seleccionamos un camino (buscando el camino de coste mínimo), el coste de nuestro camino aumenta, al igual que cuando seleccionamos un nuevo estado (buscando la secuencia de estados de mayor probabilidad), tenemos que multiplicar nuestra probabilidad total acumulada por dicha probabilidad de cambio de estado, y por tanto, nuestra probabilidad total disminuye.

Resolución del ejemplo:

Ahora vamos a resolver el ejemplo anterior con el algoritmo de Viterbi dadas ahora los costes de todos los caminos en las siguientes tablas:

	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$
S	6	1	3
$\mathbf{B_1}$	3	2	9
$\mathbf{B_2}$	5	4	1
$\mathbf{B_3}$	2	6	3

	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$
E	1	5	3
$\mathbf{B_1}$	2	3	1
$\mathbf{B_2}$	6	3	2
$\mathbf{B_3}$	2	3	3

Mantendremos dos tablas para el algoritmo, uno para reconstruir el camino y otro para los costes. La primera columna representa la columna A, la segunda la columna B y así sucesivamente hasta la última que representa el coste para llegar a *end* desde nodos de la última columna de nodos. Para cada posición (i,j), este hará referencia al nodo i-ésimo de la columna j-ésima, así, (1,1) representa al nodo A_1. En la tabla de costes, cada posición almacenará el coste mínimo para llegar desde *start* hasta al nodo que representa. Y, en la tabla de caminos, cada casilla almacenará qué nodo de la columna anterior ha sido elegido, notemos que la primera columna se rellenará con un valor por defecto, sease -1, para representar el nodo origen.

En la siguiente tabla se puede ver qué representa el valor de cada casilla, donde notamos por $B \to C_1$ al coste (acumulado) o nodo previo (dependiendo de en que tabla estemos) correspondiente al camino óptimo para ir desde la columna B hasta el nodo C_1.

$S \to A_1$	$A \to B_1$	$B \to C_1$	$C_1 \to E$
$S \to A_2$	$A \to B_2$	$B \to C_2$	$C_2 \to E$
$S \to A_3$	$A \to B_3$	$B \to C_3$	$C_3 \to E$

Siguiendo los pasos explicados, cuando hayamos procesado/llegado al nodo B_1 tendremos:

6	1+2	-	-
1	-	-	-
3	-	-	-

(a) Tabla de costes

-1	2	-	-
-1	-	-	-
-1	-	-	-

(b) Tabla de caminos

Obviando los pasos intermedios que son meros cálculos y comprobaciones obtenemos como resultado:

6	3	5	6
1	4	6	11
3	6	4	4

(a) Tabla de costes

-1	2	1	1
-1	3	1	2
-1	3	1	3

(b) Tabla de caminos

De aquí sabemos que los costes mínimos a los nodos B_j, con $j \in 1, 2, 3$, son de coste 3, 4, 6 y pasan por A_2, A_3, A_3, respectivamente. Para los nodos C_k, con $k \in 1, 2, 3$, los costes mínimos son respectivamente 5, 6, 4 y pasando todos ellos por el nodo B_1. Finalmente, el coste mínimo de pasar por los nodos C_k y llegar al nodo *end* son 6, 11, 4 respectivamente. Luego, el camino de coste mínimo tiene coste 4, y pasa por los nodos C_3, B_1 y A_2, esto último se deduce naturalmente por la definición de la tabla de caminos.

5.2. *'Part of Speech Tagging'*

Definición

El Etiquetado Gramatical (en inglés, *Part of Speech Tagging*) consiste en **clasificar** cada una de las palabras que aparecen en un discurso, en función tanto de su semántica como de su contexto en dicho discurso, tanto semántica como morfológicamente.

Esta tarea, que puede parecer un ejercicio de lenguaje para niños del colegio (y es que realmente lo es), tiene cierta complejidad a nivel computacional.

Por ejemplo, en castellano, la palabra *para* en función del contexto podría ser una preposición o un verbo: no significa lo mismo (o al menos es poco probable) en las frases 'para el tren' que en

'para nosotros', igual que sucede con la palabra *ratón*: no significa lo mismo (o al menos es poco probable) en las frases 'se me estropeó el ratón' que en 'el ratón se comió el queso'. En las frases anteriores tenemos **ambigüedad**, en el primer caso morfológica (la cual implica semántica), y en el segundo solo semántica. Nosotros nos centraremos en las morfológicas. Estas son más frecuentes en el inglés que en el castellano, motivo por el cual será el idioma de nuestro ejemplo.

Ejemplo ilustrativo

En inglés, la siguiente frase presenta varias ambigüedades morfológicas:

The fans watch the race.

La palabra *the* no presenta ninguna ambigüedad, siempre será un determinante. La palabra *fans*, en función del contexto, puede ser un verbo (ventilar) o un nombre (ventilador, admirador), no obstante, sabemos que nunca será, por ejemplo, un determinante. Como ya hemos comentado obviaremos las ambigüedades semánticas. La palabra *watch*, también podría ser un nombre (reloj) o un verbo (observar). Igualmente la palabra race podría tratarse de un nombre (carrera) o un verbo (correr, competir).

Definición del problema

El problema entonces consiste en clasificar cada palabra (en nombre, verbo, determinante...) en función de su contexto del discurso y su propia semántica. Esto último hace referencia a que, dado un cierto tipo, habría palabras con mayor probabilidad de serlo que otras.

Como esta tarea la realizará un algoritmo y no una persona, no se podrá deducir el tipo con un 100 % de precisión, entonces lo que se pretende es dar la secuencia de tipos que se corresponda con el discurso con mayor probabilidad.

Análisis del problema

Para ello la primera idea clave es observar que el tipo de cada palabra está íntimamente relacionado con el tipo de la palabra que le precede[3]. Por ejemplo, detrás de un determinante es muy probable que aparezca un nombre, e imposible que apareca un verbo. La otra idea es la que se comentó anteriormente, que para un determinado tipo, hay palabras que tendrían mayor probabilidad que otras de serlo. Por ejemplo, si una palabra fuese un verbo, sería imposible que fuese *the*, y sería quizá más probable que fuese *race* a que fuese *fans*.

Estos datos, necesarios para la resolución del problema, se obtienen de forma empírica y hacen referencia a la **probabilidad de transición** y **probabilidad de emisión** que ya se ha comentado en el apartado de los modelos ocultos de Markov.

Las siguientes tablas muestran las probabilidades de transición y emisión que suponemos obtenidas de forma empírica (realmente se han rellenado de forma pseudoaleatoria, ya que lo que nos interesa es explicar el algoritmo):

Tabla 4: Probabilidades de Transición

	Determinante	**Nombre**	**Verbo**
Inicio	0.8	0.2	0
Determinante	0	0.9	0.1
Nombre	0	0.5	0.5
Verbo	0.5	0.5	0

[3]Aquí nos estamos basando solo en la palabra anterior, pero se podrían considerar las dos anteriores o incluso secuencias más largas.

Tabla 5: Probabilidades de Emisión

	The	**Fans**	**Watch**	**Race**
Determinante	0.2	0	0	0
Nombre	0	0.1	0.3	0.1
Verbo	0	0.2	0.15	0.3

Algunos ejemplos para entender como se lee cada una de las tablas sería que la probabilidad de que haya un determinante al inicio es de 0.8, la de que vaya después de otro determinante o un nombre es 0, y de que vaya después de un verbo es 0.5.

En la segunda tenemos que al encontrar un determinante hay una probabilidad de 0.2 de que sea *the* y sin embargo es imposible que sea cualquiera de las otras 3 palabras. En esta tabla deberían aparecer más columnas, una por cada palabra que haya intervenido en los experimentos, de forma que las filas sumarían 1, aunque aquí solo hemos incluido las que intervienen en el discurso. La no ambigüedad de *the* queda reflejada en que la probabilidad de que sea algo distinto de determinante (al menos para los tipos que hemos cogido) es 0.

Insistimos en que estas tablas se han rellenado de forma pseudoaleatoria simulando que se han obtenido experimentalmente.

El diagrama de estados de este problema es el siguiente:

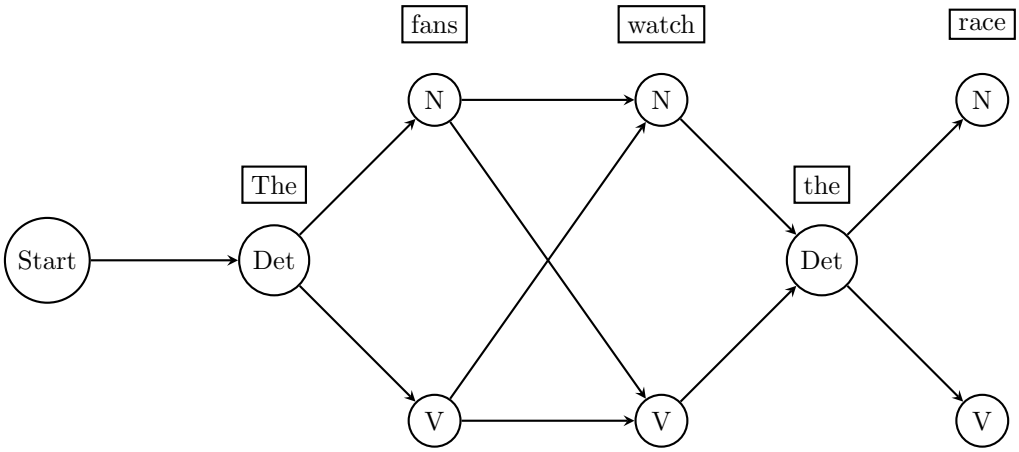


Figura 5: Diagrama de Estados del problema de Etiquetado Gramatical

Como vemos, tenemos un total de $1 \cdot 2 \cdot 2 \cdot 1 \cdot 2 = 8$ secuencias de estados posibles. Nuestro problema se reduce a estudiar cuál de nuestros estados tiene una mayor probabilidad.

Soluciones mejorables

Notemos que una solución siempre es calcular la probabilidad de cada estado y tomar la máxima al final (**fuerza bruta**). Si llamamos P al número de tipos posibles de cada palabra (tomando máximo para la notación O) y L a la longitud del discurso, vemos que la solución de fuerza bruta tendría una eficiencia de $O(P^L)$ por lo que crecería **exponencialmente** con la longitud del discurso y eso no interesa. Esta es la misma situación que ya vimos que se daba en los modelos de Markov (3.1).

Otra solución, de eficiencia $O(L)$, sería simplemente para cada palabra tomar el tipo con mayor probabilidad. En este caso concreto, solo acertaríamos para el determinante *the*, ya que diríamos que *fans* es un verbo cuando es un nombre en este discurso, que *watch* es un nombre cuando en el discurso es un verbo, y que *race* es un verbo cuando en el discurso es un nombre.

Solución con el algoritmo de Viterbi

La solución que combina eficacia y eficiencia es la dada por el **algoritmo de Viterbi**, y es la que se describe a continuación:

Supongamos que, para todo k, con $0 \leqslant k \leqslant L$, p_k denota el tipo morfológico de la palabra en la posición k-ésima, y ω_k representa la palabra que hay en dicha posición. Entonces, podremos escribir la probabilidad máxima del entre todos los estados (S) como:

$$S = max_{p_i} \prod_{k=1}^{L} P(p_k|p_{k-1}) \prod_{k=1}^{L} P(p_k|\omega_k) \tag{1}$$

Donde P es la función de probabilidad. En la expresión de arriba el primer productorio hace referencia a las probabilidades de transmisión y el segundo a las probabilidades de emisión.

En la siguiente figura se muestra la resolución del algoritmo para este ejemplo que se irá comentando paso a paso:

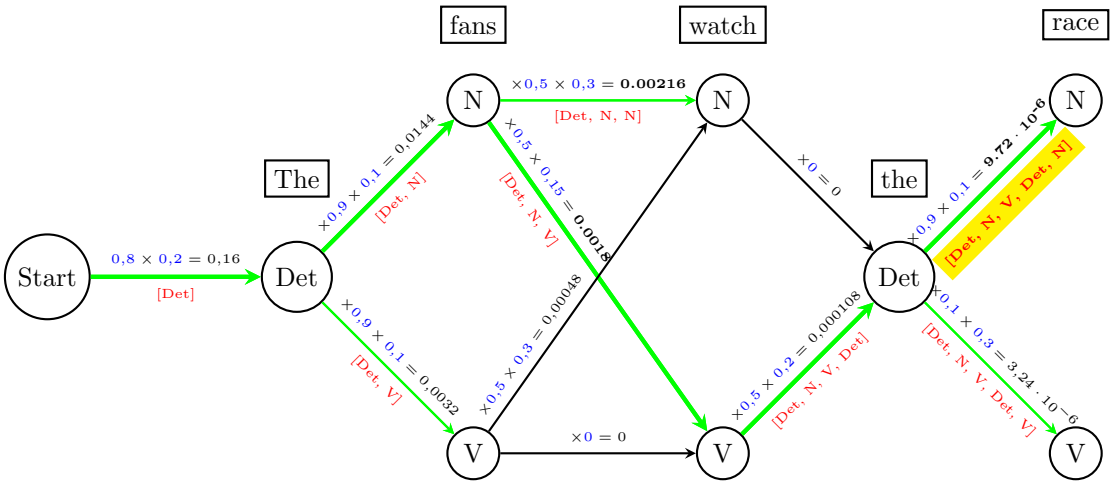


Figura 6: Algoritmo de Viterbi sobre el Diagrama de Estados del problema

Vamos a ir palabra por palabra, de forma que en cada una estaremos resolviendo el subproblema en el que el discurso llega hasta esa palabra.

Desarrollo del algoritmo

1. En la primera palabra, tenemos un único estado posible, el que la palabra *the* sea un determinante. La probabilidad de este estado se calcula con la expresión anterior: la probabilidad de que la primera palabra sea un determinante (probabilidad de transición) por la probabilidad de que, siendo un determinante, que sea *the* (probabilidad de emisión). Esto es $0{,}8 \cdot 0{,}2 = 0{,}16$.

2. Para la segunda palabra, ya tenemos dos posibilidades: la primera que esta sea un nombre y la segunda, que sea un verbo. La primera se calcula como la probabilidad de que sea un nombre condicionada a que la palabra anterior es un determinante por la probabilidad de que la palabra sea fans, sabiendo que es un nombre. La segunda es análoga. Multiplicando las probabilidades por la que habíamos obtenido en el paso anterior, salen $0{,}16 \cdot 0{,}9 \cdot 0{,}1 = 0{,}0144$ para la primera y $0{,}16 \cdot 0{,}1 \cdot 0{,}2 = 0{,}0032$ para la segunda, obteniendo hasta ahora 2 ramas. Notemos que, obedeciendo al productorio de (1) tenemos que las probabilidades que vamos calculando son **acumuladas**, y por tanto, con respecto a la palabra anterior, **decrecientes** (cuanto más largo sea el discurso, más improbable será una secuencia de estados determinada).

3. Para la tercera palabra, como tenemos 2 ramas activas y 2 posibilidades, tendremos 4 probabilidades a calcular. Para el estado N, tendremos la probabilidad de la secuencia [Det,N,N] que es 0.00216 y la de [Det,V,N] que es 0.00048. De estas dos posibilidades, se toma la máxima, descartando la rama [Det,V,N], ya que se sabe por el **principio de optimalidad de Bellman** que no va a dar con la solución óptima. Para el otro estado (V) de la palabra *watch* se hace exactamente lo mismo, quedándonos con 2 ramas activas.

4. En la cuarta palabra convergen ambas ramas activas y se calcula la probabilidad acumulada por cada una de ellas de la misma forma, tomando la máxima que es 0,000108.

5. En la última palabra se separa la rama que teníamos en 2 posibles estados. Tomando de entre ellos la máxima probabilidad ya tendríamos la solución, que es [Det,N,V,Det,N] con una probabilidad de $9{,}72 \cdot 10^{-6}$, una probabilidad muy baja, pero la máxima de todas las combinaciones.

El **paso clave** lo hemos desarrollado en el punto 3, que es donde se ve que el algoritmo de Viterbi mejora la fuerza bruta al descartar combinaciones sin llegar a procesarlas por completo.

Además podemos observar que la secuencia más probable del algoritmo de Viterbi coincide justo con la solución al ejemplo.

Comparación de eficiencia

Por último veamos la eficiencia del algoritmo, comparándola con la fuerza bruta: En el desarrollo del algoritmo, para cada palabra, y para cada estado posible de la palabra evaluamos la probabilidad de dicho estado desde cada uno de los estados anteriores. Por ejemplo, para la tercera palabra, hemos evaluado la probabilidad de [Det,N,N] y [Det,V,N] para tomar máximo entre ambas, y la probabilidad de [Det,N,V] y [Det,V,V] para también tomar máximo entre ambas. En total en cada palabra evaluaremos P^2 probabilidades (P es el número de estados posibles de la palabra actual, y también el de los estados posibles de la palabra anterior, recordando que tomamos máximo para la notación O). Por tanto concluimos que este algoritmo tiene eficiencia $O(LP^2)$, por lo que el tiempo de cálculo crece **linealmente** respecto al número de palabras del discurso y se ve claramente la mejora en eficiencia respecto a la solución de fuerza bruta que era exponencial.

6. Conclusiones

Con la realización de esta práctica, hemos visto ampliamente reforzados nuestros conocimientos sobre **Programación Dinámica**, aunque sea con un enfoque más centrado en el diseño de algoritmos y no tanto en la implementación de estos, con un ejemplo de algoritmo sencillo y famoso como es el **Algoritmo de Viterbi**. Hemos aprendido también un tema de grafos interesante como es el de las Cadenas de Markov y los Modelos Ocultos de Markov, así como su estrecha relación con el mencionado algoritmo y la utilidad de este último en áreas muy importantes de la **Inteligencia Artificial** como pueden ser el cálculo de caminos mínimos por etapas y incluso el etiquetado gramatical, próximo a la predicción de palabras y estudio computacional del lenguaje natural.

Además, hemos tenido que realizar un **trabajo de investigación** para saber de qué se trataba el algoritmo de Viterbi así como informarnos acerca de dónde se usaba en tanto de ser capaces de proporcionar ejemplos de uso. Para ello, también ha sido indispensable alcanzar una **comprensión profunda** del algoritmo así como de sus ejemplos de uso.

En definitiva, consideramos que esta práctica ha sido útil no tan solo en lo que se refiere a la materia de la asignatura, sino también para desarrollar ciertas habilidades transversales, y para tener un acercamiento a campos de estudio interesantes y muy presentes en nuestro día a día como titulados en Ingeniería Informática.