Maquinas Autómatas Probabilísticas —

Arimany, Adrian.

Universidad del Valle de Guatemala, Facultad de Ingeneria.
Departamento de Ingeniera de la Computación, Teoría de la Computación, Sección 11

Catedrático: Alan Reyes -

Palabras Clave: *Maquinas Autómatas Probabilísticos, Monte Carlo, Puntos de Corte, Expresiones No Regulares.*

Abstract. *Este informe es una breve introducción a la Teoría de las Maquinas Autómatas Probabilísticas con un enfoque a la Teoría de Maquinas Autómatas estudiadas en Teoría de la Computación. El informe esta estructurado en el siguiente orden: Introducción a a la teoría mediante definiciones y ejemplos sencillos; una exposiciones de teoremas seleccionados que exponen la dificultad de estudiar estas maquinas; seguido por una explicación de un programa para estudiar estas maquinas; finalizando con una breve propuesta a que se va a estudiar de la teoría en el siguiente proyecto.*

1. Introducción

Este proyecto tiene como propósito introducir la teoría sobre Autómatas Probabilísticos (AP), con el añadido de incluir un programa complementario que ilustra cómo estos Autómatas funcionan de manera computacional. Un aspecto clave en este proyecto es comparar los AP con la Teoría de Máquinas Autómatas (MA) aprendida en Teoría de la Computación. Esta comparación se lleva a cabo demostrando teoremas que relacionan ambas teorías, y en última instancia enfatizando la complejidad desafiante que manejan los AP sobre los MA, con el énfasis en que los AP no son equivalentes a los Autómatas Finitos Deterministas (AFD).

Algunos aspectos a considerar sobre los AP, son que históricamente los AP fueron introducidos en un artículo escrito por Rabin (1963), y a lo largo de los años la teoría ha ganado cierta atracción especialmente en el ámbito de la inteligencia artificial. Otro aspecto es que los AP son diferentes a las cadenas de Markov, aunque existe una conexión entre estas dos teorías; sin embargo, en esta introducción debido a la limitación de tiempo no se prestará atención significativa a esta relación. Con respecto al programa codificado, este se realiza en Python y utiliza Simulaciones de Monte Carlo para proporcionar estimaciones numéricas sobre las probabilidades que ciertas palabras pueden tener en un AP dado. La introducción a la teoría se utiliza para entender el mecanismo detrás de lo que se observara en el programa.

Por último, este proyecto asume que el lector posee una comprensión sólida de álgebra lineal y teoría de la probabilidad. Si fuera el caso, algunos teoremas relacionados con estas disciplinas se demostrarán en el anexo, aunque cualquier teorema o definición que no esté directamente vinculado o que no aporte una contribución significativa a la Teoría de los AP solo será referenciado pero no presentado[1]. Además, la mayor parte del material utilizado en esta

[1]Por ejemplo, no escribiré los axiomas de la teoría de la probabilidad de Kolmogórov, ni operaré explícitamente

introducción fue obtenido y estudiado de Pouly (2022), y en algunos casos asistido por OpenAI (2025).

2. Teoría de las Autómatas Probabilísticos

Definición 1 Autómata Probabilístico.

Un autómata que tiene valores ponderados para cada transición y si su transición forma una matriz estocástica (sentido de filas) se le considera una Autómata Probabilístico (AP). Se le denota a las AP con una letra mayúscula ondulada (i.e. $\mathcal{A}$), y son una tupla

$$\mathcal{A} = \{\Sigma, Q, S, T, \mu(a)\}$$

Donde,

- Σ: un alfabeto finito.
- Q: es el conjunto de estados finitos.
- $S \in \{0,1\}^{1\times Q}$: un vector fila estocástico de con el estado inicial.
- $T \in \{0,1\}^{Q\times 1}$: un vector columna estocástico de estados terminales.
- $\mu(a) \in [0,1]^{Q\times Q}$: una matriz estocástica que representa a las transiciones de $\mathcal{A}$ para toda letra $a \in \Sigma$.

Notación 1 Transiciones por estado.

Una forma eficiente de denotar a una transición por un estado, especialmente cuando se usan en grafos, es la siguiente notación:

$$a \mid p \equiv \mu(a)_{q_i,q_j}$$

Notemos que a es la letra en transición, y p es la probabilidad asignada a esa transición. En el caso que $p = 1$ se deja la transición como usualmente se hace en MA. Y si $p = 0$ entonces no se pone la transición.

Notas 1 sobre Definicion 1.

1. A diferencia de las MA vistas en el curso de Teoría de la Computación, las AP tienen como lenguaje determinar las probabilidad de que ciertas palabras se acepten por la maquina. Revise el Ejemplo 1.

2. Las entradas en una matriz estocástica se puede evaluar donde sea a una letra de Σ, y sean los estados $\{q_1, \cdots q_n\}$ entonces la matriz asociada de $\mu(a)$ es,

$$\mu(a) = \begin{pmatrix} \mu(a)_{q_1,q_1} & \cdots & \mu(a)_{q_1,q_n} \\ \vdots & \ddots & \vdots \\ \mu(a)_{q_1,q_n} & \cdots & \mu(a)_{q_n,q_n} \end{pmatrix}$$

   Como es una matriz estocástica, entonces para cada reglón i donde $1 \leq i \leq n$ tendría que ser

$$\sum_{j=1}^{n} \mu(a)_{q_i,q_j} = 1$$

   .

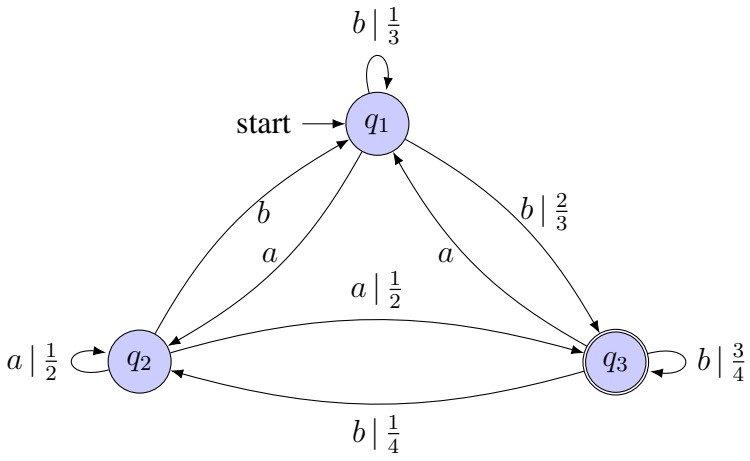
---

la multiplicación de matrices

3. Hay varias formas de evaluar la probabilidad si una palabra es aceptada, en si el mecanismo es algo parecido a las MA, pero la diferencia es que existen diferentes "caminos" o transiciones para llegar a una palabra de aceptación usando la mismas letras. En este informe solo se presentara uno de estos mecanismos que es el mas eficiente a evaluar, este mecanismo se define en *Definicion* 3.

Definición 2 Probabilidad de Aceptación Completa.
Decimos que una palabra w es aceptada completamente si y solo si la probabilidad de que todas las transiciones que forman w llegan con probabilidad 1.
Notación 2 Probabilidad de Aceptación.
Usamos $\mathcal{A}(w)$ como la probabilidad que la palabra w sea aceptada en $\mathcal{A}$. Decimos que una palabra no es aceptada como $w \notin \mathcal{A}$ lo que corresponde que $\mathcal{A}(w) = 0$, y si la palabra es aceptada pero no completa entonces, $w \in \mathcal{A}$, lo que es decir $\mathcal{A}(w) \in (0, 1)$. Pero si la palabra es completa, entonces $w = \mathcal{A}$ o bien $\mathcal{A}(w) = 1$. Esta notación también se puede usar cuando se evalúen ciertas transiciones o caminos que toma una palabra, como $\mathcal{A}(q_i \xrightarrow{w} q_j)$ lo cual es lo mismo a decir $\mu(w)_{q_i,q_j}$.

Figura 1: Grafo del Ejemplo 1



Example 1 Sea $\mathcal{A} = \{\Sigma, Q, S, \mu, T\}$.
Lo que definimos como,

$$\Sigma = \{a, b\},\ Q = \{q_1, q_2, q_3\},\ S = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix},\ T = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Determine la palabra aceptada completamente por $\mathcal{A}$.

Para poder determinar la palabra aceptada por $\mathcal{A}$ primero tenemos que determinar la probabilidad de que cada letra de Σ tienen en llegar al estado de terminación, cual en este caso es q_3. El

mecanismo mas adecuado es usando un producto de matrices.

Definición 3 Método de Matriz.
Sea $\mathcal{A}$ un AP con una matriz estocástica $\mu(a)$ para cada letra a en su alfabeto, tiene Q estados donde el estado inicial esta definido en el vector fila $S^{1\times Q}$ y los estados terminales están definidos en el vector columna $T^{Q\times 1}$. Por lo tanto decimos que la palabra w compuesta de letras $a_1, a_2, \cdots a_n$ tiene probabilidad

$$\mathcal{A}(w) = S \times \mu(w) \times T = S \times \mu(a_1) \cdot \mu(a_2) \cdots \mu(a_n) \times T$$

Notas 2 sobre Definicion 3.

1. Notemos las dimensiones en $S \times \mu(w) \times T$:

$$dim(S \times \mu(w) \times T) = (1 \times Q)(Q \times Q)(Q \times 1) = 1 \times 1 = 1$$

   Osea que claramente este producto va retornar un escalar, y como las matrices son estocásticas, entonces el escalar siempre va estar en el cerrado $[0, 1]$.

2. En el caso de evaluar la transición de una letra vacía ϵ, notemos que la matriz de transición no seria con solo entradas cero porque si esto fuera el caso, cualquier palabra que tenga ϵ retornaría probabilidad 0. Envés, la matriz de transición para la ϵ es el matriz identidad. Esto se debe a que al multiplicar la matriz identidad con otra matriz de misma dimensión retorna la matriz.

3. Si consideramos el producto $S \times \mu(w)$ esto nos da en dimensiones un vector fila, pero este vector fila ya nos dice si la palabra va tener probabilidad 0, ademas nos dice que tan distribuido están las probabilidades de la palabra entre los estados.

4. Algo que no se va a justificar en esta entrega es que si vemos la maquina del Ejemplo 1, podemos notar que hay bucles, y en teoría uno puede hacer el producto de un bucle infinitas veces, pero no necesariamente la probabilidad va a converger.

Ahora que el mecanismo para evaluar probabilidades es conocido, seguimos a investigar la palabra completa en el *Ejemplo* 1. Para esto tenemos que hacer las matrices de transiciones para cada letra del alfabeto, cuales serian:

$$\mu(a) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix} \quad \mu(b) = \begin{pmatrix} \frac{1}{3} & 0 & \frac{2}{3} \\ 1 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} \end{pmatrix}$$

Ahora notemos que si queremos evaluar si la palabra a es parte de $\mathcal{A}$, podemos operar:

$$\mathcal{A}(a) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 0$$

Cual era evidente, dado que al ver el diagrama uno puede notar que solo con la letra a no es posible llegar al estado de aceptación. Por otro lado si queremos evaluar la palabra ab:

$$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{3} & 0 & \frac{2}{3} \\ 1 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 0$$

Cual claramente también podemos observar que no es factible llegar al estado de aceptación empezando de a y después a b. Ahora que ya hemos evaluado unas palabra regresemos a lo que queríamos, cual es la palabra completa de $\mathcal{A}$. Pues la solución es bastante simple, no tiene palabra completa. No importa que palabra usted inserta en Definicion 3 usted nunca va a poder llegar a tener una probabilidad 1. Y esto es importante a entender, las autómatas probabilísticos no están hechas principalmente para evaluar la palabra completa,[2] envés se usan para evaluar si una palabra es menor o mayor a cierta probabilidad, por lo que la siguiente definición nos da ese mecanismo para estudiar.

Definición 4 Automata Ponderado.
Sea $\mathcal{A}$ una PA y $\lambda \in [0,1]$ donde el lenguaje de $\mathcal{A}$ es

$$\mathcal{L}_A(\lambda) = \{w \in \Sigma* \mid \mathcal{A} > \lambda\}$$

Osea es el conjunto de palabras aceptada es $\mathcal{A}$ donde la probabilidad es al menos λ.

Notas 3 sobre Definicion 4.

1. Notemos que $\mathcal{L}_A(\lambda)$ es un lenguaje estocástico, donde decimos que λ es su punto corte (”cut-point”). Pero importante dejar dicho que $\mathcal{L}_A(\lambda)$ NO es por si un lenguaje ponderado, pero un conjunto de lenguajes ponderados, donde λ tiene su lenguaje único.

2. Sea el conjunto de operadores de compasión $\triangleright\triangleleft = \{\leq, \geq, >, <, =, \neq\}$, por lo tanto los puntos corte tiene todas la operaciones de comparación:

$$\mathcal{L}_A^{\triangleright\triangleleft}(\lambda) = \{w \in \mathcal{A}^* \mid \mathcal{A}(w) \triangleright \triangleleft \lambda\}$$

Example 2 Sea $\mathcal{A} = \{\Sigma, Q, S, \mu, T\}$ la maquina del Ejemplo 1.
Investigamos los dos problemas que pueden surgir cuando usamos los puntos de corte como en la Definicion 4:

1. Encuentra una palabra en $\mathcal{L}_A(\frac{1}{2})$:

$$\mu(bb) = \begin{pmatrix}1 & 0 & 0\end{pmatrix} \begin{pmatrix}\frac{1}{3} & 0 & \frac{2}{3}\\ 1 & 0 & 0\\ 0 & \frac{1}{4} & \frac{3}{4}\end{pmatrix} \begin{pmatrix}\frac{1}{3} & 0 & \frac{2}{3}\\ 1 & 0 & 0\\ 0 & \frac{1}{4} & \frac{3}{4}\end{pmatrix} \begin{pmatrix}0\\0\\1\end{pmatrix} = \frac{13}{18}$$

2. Encuentre las palabras en $\mathcal{L}_A(\lambda)$ para todo $\lambda > \frac{13}{18}$ que tengan como máximo 2 letras, osea esto nos dice que hay que encontrar: $\frac{13}{18} < \mathcal{A}(w) < 1$.

   Anteriormente, ya evaluamos que en $\mathcal{A}$ la letra a no puede llegar al estado de aceptación, ademas ab no es parte de este lenguaje estocástico, por otro lado sabemos que bb tiene probabilidad $\frac{13}{18}$ cual tampoco es parte de este lenguaje estocástico. Por lo que tendríamos que evaluar aa y ba, pero con ver la ilustración, nos damos cuenta que ba no puede ser parte de este lenguaje estocástico dado que el estado de aceptación q_3 solo tiene un bucle con b. Por lo que aa es el único candidato, entonces evaluamos $\mathcal{A}(aa)$:

$$\mathcal{A}(a) = \begin{pmatrix}1 & 0 & 0\end{pmatrix} \begin{pmatrix}0 & 1 & 0\\ 0 & \frac{1}{2} & \frac{1}{2}\\ 1 & 0 & 0\end{pmatrix} \begin{pmatrix}0 & 1 & 0\\ 0 & \frac{1}{2} & \frac{1}{2}\\ 1 & 0 & 0\end{pmatrix} \begin{pmatrix}0\\0\\1\end{pmatrix} = \frac{1}{2}$$

   Osea que aa tampoco es parte de este lenguaje estocástico. Osea que este lenguaje estocástico para palabras de dos letras no tiene ninguna palabra. Es mas en si no hay ninguna palabra que exista en este lenguaje estocástico, dado que la palabra mas aceptada en $\mathcal{A}$ es bb.

[2] Note, si hay situaciones donde puede existir una palabra completa en una maquina

Ahora supongamos que pasaría si en el ejercicio anterior se pediría buscar todas las palabras que están en un lenguaje estocástico, cuantas palabras pueden existir en un lenguaje estocástico. Y justamente los siguientes teoremas nos van a dar esa respuesta, específicamente el Teorema 3.

Propiedades de AP

Lo que sigue son teoremas importantes de la Teoría, en primer lugar vamos a evaluar la relación entre expresiones regulares y las AP (Teorema 1); después probamos que todo lenguaje estocástico están contenidos de lenguajes regulares (Teorema 2); seguido por un teorema que demuestra que existe un autómata 'universal' no regular AP (Teorema 3).

Teorema 1 (*Todo Lenguaje Regular es Estocástico*).

Demostración. Consideremos un lenguaje regular $\mathcal{L}_N$ que corresponde a una Autómata Finito Determinista (AFD), donde su maquina completa que le corresponde a $N = \{Q, \Sigma, \delta, q_0, F\}$; si es necesario incluya estados muertos para completar N. Tratemos a cada letra $a \in \Sigma$ como una matriz $\mu(a) =\mid Q \mid \times \mid Q \mid$, para cada entra de esta matriz le corresponda a una transición $m(a)$ donde

$$m(a)_{q_i,q_j} = \begin{cases} 1 & \text{si } \delta(q_i, a) = q_j \quad i, j \in Q \\ 0 & \text{De lo contrario} \end{cases}$$

Como N es determinista y completa, cada fila de $\mu(a)$ tiene exactamente un único 1, y todo otras entras en esa fila son 0, por ende $\mu(a)$ es renglón estocástico. Ahora definamos α como un vector fila, tal que tiene un 1 en q_0 y 0 en otro caso. Y sea β un vector columna que es 1 en el estado F, y 0 en los otros estados. Ahora considere una cadena $w = a_1 \cdot a_2 \cdots a_k$ entonces

$$\mu(w) = \mu(a_1) \cdot \mu(a_2) \cdots \mu(a_k), \text{ donde } \mu(\epsilon) = I$$

Por lo tanto notemos que

$$\alpha \cdot \mu(w) = \delta(q_0, w)$$

Entonces,

$$\alpha \cdot \mu(w) \cdot \beta = \begin{cases} 1 & \delta(q_0, w) \in F \quad (w \in L) \\ 0 & \text{De lo contrario} \end{cases}$$

□

Por si la demostración no lo dejo evidente, lo que el teorema dice es que las AFD son estocásticas porque se pueden definir como una AP, pero solo te van a dar probabilidad 1 o probabilidad 0.

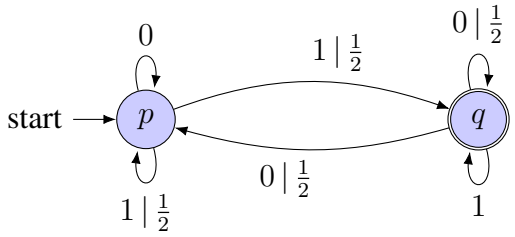
Teorema 2 (*Lenguajes Estocásticos contienen estrictamente lenguajes regulares*).

Demostración. Por el Teorema 1 sabemos que todo lenguaje reglar son estocástico, pero ahora vamos a dar un ejemplo de un lenguaje estocástico no-regular. Consideremos $\mathcal{A} = \{A, Q, S, \mu, T\}$ donde $A = \{0, 1\}$, $Q = \{p, q\}$ y

$$S = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad \mu(0) = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \quad \mu(1) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$\mathcal{A}$ se puede ilustrar como:

Figura 2: Maquina Probabilística No regular.



Ahora consideremos una palabra en A^* donde $[w] = \sum_{i=1}^{|w|} w_i 2^{i-|w|-1}$. Se quiere probar que si, $S\mu(w) = \begin{pmatrix} 1-[w] & [w] \end{pmatrix}$, Para ello evaluemos que,

$$[\epsilon] = 0 \quad [w \cdot 0] = \frac{[w]}{2} \quad [w \cdot 1] = \frac{1+[w]}{2}$$

Ahora por inducción, notemos que,

$$S = \begin{pmatrix} 1-[\epsilon] & [\epsilon] \end{pmatrix}, \begin{pmatrix} 1-[w] & [w] \end{pmatrix} \mu(0) = \begin{pmatrix} 1-\frac{1}{2}[w] & [w] \end{pmatrix}, \begin{pmatrix} 1-[w] & [w] \end{pmatrix} \mu(1) = \begin{pmatrix} \frac{1-[w]}{2} & \frac{1+[w]}{2} \end{pmatrix}$$

Por lo que la probabilidad de aceptación de w en $\mathcal{A}(w) = [w]$, pero notemos que $[w]$ es denso en $[0, 1]$, por lo que $w \in \mathcal{A}*$. Ahora si consideramos un $\lambda < \mu$, entonces $\mathcal{L}_A(\lambda) \supsetneq \mathcal{L}_A(\mu)$. Esto nos dice que como $[w]$ es denso, uno puede encontrar un w tal que $\lambda < \mathcal{A} \leq \mu$ para todo $\lambda < \mu$. Ahora, como hay infinitos no contables $\lambda < \mu$ entonces el conjunto de $\{\mathcal{L}_A(\lambda) \mid \lambda \in [0, 1]\}$ es un conjunto no contable; pero si el conjunto de expresiones regulares es contable, entonces tiene que existir un λ tal que $\mathcal{L}_A(\lambda)$ no es regular. □

El Teorema que sigue es una extensión del previo que se usara en un ejemplo para evaluar como se encuentran expresiones no-regulase en un AP. Para ello vamos a definir un concepto que se usara en la siguiente demostración,

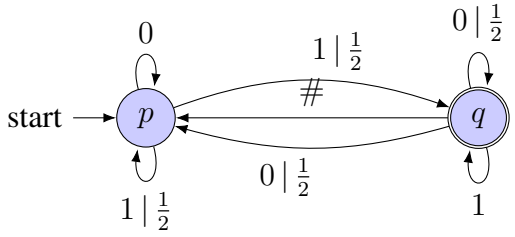
Teorema 3 (*Existe una Autómata Probabilístico universal no-regular*).

Demostración. Consideremos $\mathcal{B} = \{A', Q, S, \mu, T\}$ donde $A' = A \cup \{\#\} = \{0, 1, \#\}$, $Q = \{p, q\}$

$$S = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad \mu(0) = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \quad \mu(1) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$\mathcal{A}$ se puede ilustrar como:

Figura 3: Maquina Probabilística No regular con alfabeto extendido.



Notemos que para leer $\#$ se requiere que $\mathcal{B}$ este en el estado q, pero note que q es el estado de aceptación. Entonces para que $\mathcal{B}$ pueda leer $\#$, y aceptar una palabra que lo contenga. Para ello consideremos $\forall u, v \in A^*$,

$$\mathcal{B}(u\#v) = \mathcal{B}(p \xrightarrow{u} q)\mathcal{B}(p \xrightarrow{v} q) = \mathcal{A}(u)\mathcal{A}(v) = [u][v]$$

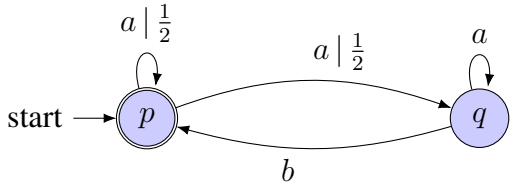
Pero si recordamos en Teorema 2 que el conjunto $[A^*] = \{[w] \mid w \in A*\}$ es denso en $[0,1]$. Ahora si fijamos un $\lambda \in (0,1)$ y tomamos $u, v \in A*$ tal que $\lambda < [u][v]$ entonces la densidad de $[A^*]$ nos asegura que $w \in A^*$ tal que,

$$\frac{\lambda}{[u]} > [w] > \frac{\lambda}{[v]}$$

Por ende $u \neq \mathcal{L}_B(\lambda)v$. Pero como $[A^*]$ es denso, entonces existen infinitas parejas de u, v tales que aseguran que $\mathcal{L}_B(\lambda)$ no sea regular. □

Osea lo que el teorema nos dice es que si consideremos un AP, tal que $\mathcal{L}_A(\lambda)$ va ser no regular $\forall \lambda \in (0,1)$. Entonces para terminar esta parte teórica, se usara el $Teorema\ 3$ para probar que la siguiente autómata $\mathcal{C}$, es universalmente no-regular como un ejemplo para tener una conceptualización mas rigurosa ante este mecanismo que no se puede evaluar en las AFD.

Figura 4: Illustracion de la Maquina $\mathcal{C}$ para el siguiente ejemplo.



Example 3 Sea $\mathcal{C} = \{C, Q, S, \mu, T\}$.
Donde, $C = \{a, b\}$, $Q = \{p, q\}$,

$$S = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad T = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \mu(a) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix} \quad \mu(b) = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

Demuestre que $\mathcal{C}$ es universalmente no-regular.[3]

Demostración. Primero notemos algo muy importante, y eso es que la matriz de la transición b no es estocástica dado que la primera fila no suma a 1, pero eso se debe a que esta matriz es substochastico, osea que las filas son menor o igual que 1.

Notemos lo siguiente sobre $\mathcal{C}$. Se empieza de p, y se puede leer a^n, y en teoría se puede aceptar la palabra a^n después de $n \in \mathbf{N}$ veces, lo que esto siempre va a dar una probabilidad de 2^{-n}. Pero si en algún caso se mueve a q, solo puede ser aceptada la palabra si regresa con b, pero notemos que q no tiene una transición de a para regresar a p. Por lo que si seguimos este patrón la palabra acepta de $\mathcal{C}$ tiene que ser de la siguiente forma:

$$x(n_1, \cdots, n_k; m) = a^{n_1} b \cdots a^{n_k} b a^m \quad (k \geq 0, n_i, m \in \mathbf{N})$$

Ahora si queremos verlo con sus probabilidades asociadas, notemos que, $\mathcal{C}(a^{n_i} b) = 1 - 2^{n_i}$ para cada $i \in k$. Pero como podemos tomar k veces esta palabra, notemos que esto implicaría,

$$\mathcal{C}(x(n_1, \cdots, n_k; m)) = 2^{-m} \Pi_{i=1}^{k} (1 - 2^{-n_i})$$

[3]Este problema tiene una pista que la dio Pouly (2022) pg. 5, la que dice lo siguente: Show that for every word $x(n_1, \ldots, n_k) := a^{n_1} b a^{n_2} \cdots a^{n_k} b$, we have $C\big(x(n_1, \ldots, n_k); m\big) = 2^{-m} \prod_{i=1}^{k} (1 - 2^{-n_i})$. Show that if $u = x(n_1, \ldots, n_k)$ and $w = x(n_{k+1}, \ldots, n_l)$ then $C(uw) = C(u)\, C(w)$. Show that $\big\{C(x(n_1, \ldots, n_k)) : n_1, \ldots, n_k \in \mathbb{N},\ k \in \mathbb{N}\big\}$ is dense in $[0,1]$. Conclude that C is universally non-regular. Use the same proof idea as Theorem-14 (osea Teorema 3 aqui).

Ahora recordemos que estamos evaluando que $\mathcal{C}$ es universalmente no-regular, entonces como lo define la demostración del Teorema 3, definamos u y w, en este caso sea

$$u = x(n_1, \cdots, n_k) = a^{n_1} \cdots a^{n_k} b \quad w = x(n_{k+1}, \cdots, n_l; m)$$

Entonces al evaluar estas palabras tenemos que

$$\mathcal{C}(u) = \Pi_{i=1}^{k}(1 - 2^{-n_i}), \quad C(w) = 2^{-m}\Pi_{i=k+1}^{l}(1 - 2^{-n_i})$$

Por lo tanto,

$$\mathcal{C}(uw) = \mathcal{C}(u) \cdot \mathcal{C}(w) = \Pi_{i=1}^{k}(1 - 2^{-n_i}) \cdot 2^{-m}\Pi_{i=k+1}^{l}(1 - 2^{-n_i}) = 2^{-m}\Pi_{i=1}^{k}(1 - 2^{-n_i})$$

Si seguimos el argumento de la demostración, hay que usar la propiedad sobre la densidad del cerrado $[0, 1]$. Para esto observemos cuando $m = 0$,

$$\Pi_{i=1}^{k}(1 - 2^{-n_i}) \in [0, 1]$$

Ahora como es $[0, 1]$ es cerrado y esto esta en $\mathbf{R}$, podemos partir el intervalo, en $0 \leq x(n_1, \cdots n_k) \leq \frac{1}{2}$. Si consideramos un cota logarítmica:

$$-x - x^2 \leq log(1 - x) \leq -x$$

Ahora evaluemos cuando $x = 2^{-n}$, para $n \in \mathbf{N}$ y observemos que

$$2^{-n} \leq -log(1 - 2^{-n}) \leq 2^{-n} + 2^{-2n}$$

Ahora consideremos un $y \in [0, 1]$ y un $\epsilon > 0$, para $t = -log(y)$. Tome una combinación finita de $\sum_j 2^{-n_j}$ tales que aproximen a t, esto no asegura que

$$\sum_j -log(1 - 2^{-n_j} \approx t$$

Para un j que se requiera. Por las propiedades de los logaritmos, tenemos que

$$\Pi_j(1 - 2^{-n_j} \approx e^{-t} = y$$

Pero en particular podemos llegar a que

$$\mid \Pi_j(1 - 2^{-n_j}) - y \mid < \epsilon$$

Por lo que llegamos a justificar que el conjunto

$$\{\mathcal{C}(x(n_1, \cdots, n_k)) \mid k \in \mathbf{N}, n_i \in \mathbf{N}\} \text{ es denso en } [0, 1]$$

Como llegamos que el conjunto de esas expresiones que se aceptan en $\mathcal{C}$ es denso en $[0, 1]$, consideremos el lenguaje estocástico $\lambda \in (0, 1)$, donde lo que estamos evaluando son los puntos de corte de

$$L_{>\lambda}(\mathcal{C}) = \{w \in \{a, b\}* \mid \mathcal{C}(w) > \lambda\}$$

Ahora si el lenguaje fuera regular, la palabra aceptada seria $(a^+b)^*a^*$ lo que en si esta palabra fue expresada como $x(n_1, \cdots n_k; m)$, y sabemos que esta expresión tiene la probabilidad $2^{-m}\Pi_{i=1}^{k}(1 - 2^{-n_i})$. Pero recordemos que podemos llegar a esa probabilidad si usamos las palabras uw, pero anteriormente llegamos a que esta expresión $(a^+b)^*a^*$ es densa en $[0, 1]$. Por lo que el Teorema 3 no asegura que $\mathcal{C}$ es universalmente no-regular. □

Ahora no voy a negar que me asistí de OpenAI (2025) para probar este ejemplo, especialmente cuando tenia que llegar a probar que la expresión es Densa. Aun así, este ejemplo es interesante en que probar que una expresión es universalmente no-regular tiene como mayor dificultad en que la expresión es densa. Y algo que no esperaba evaluar, es que se tuvo que usar un argumento de aproximación para justificar que es denso. Otro aspecto interesante, es que decir que la expresión es densa, requiere una elaboración sofisticada para concluirlo, por lo que esperaría que estudiar esta teoría me ayude en cursos que voy a llevar a futuro.

3. Simulador para Maquinas Probabilísticas

Para esta ultima parte de este informe presentamos un proyecto de programación que fue hecho para evaluar AP usando Python - versión 3.12.3. Con el siguiente link se puede acceder al repositorio: (repositorio de github). Como se observara, actualmente el programa hace lo siguiente:

- La estructura de datos del programa:
  1. Los AP están formateados en JSON[4], y después estructurados en una clase PFA que guarda específicamente las transiciones de pares de estado y letra (q_i, a), y con un retorno de los estados con transición y sus probabilidades asignadas.
  2. Ademas asegura que todo el alfabeto sea estocástico o substochastico.
- Los métodos de evaluar probabilidades en el programa:

  1. Método de matrices, toma la parejas de transiciones de la clase PFA para convertirlas a matrices, y también determina el vector inicial y el de aceptación. Y después evalúa la matriz como se demostró en el Ejemplo 1. Lo que retorna es el diccionario:

```
{
"word": word,
"exact_probability": result,
"time taken": time_end - start_time
}
```

  2. Monte Carlo, lo que hace es corre n veces la palabra evaluada, usando un indicador como el siguiente:

$$\mathbb{1}_w = \begin{cases} 1 & \text{si } w \in \mathcal{A} \\ 0 & \text{si } w \notin \mathcal{A} \end{cases}$$

     Y después evalúa el promedio de corridas por la suma del indicador. Lo que retorna esto es el diccionario:

```
{
"word": word,
"n_trial: n_trial,
"acceptance_probability": acceptance_prob,
"avg_path_prob": avg_path_prob,
"stddev_path_probability": std_path_prob,
"time_taken": end_time - start_time
}
```

[4]En el anexo se puede observar un ejemplo de como el json esta estructurado.

Algo importante a notar es que se están evaluando los tiempos de completitud para ambas simulaciones porque la idea es también comparar estos tiempos de complejidad.

- La operaciones en el programa:

Pestaña 1 Evaluar la probabilidad de una palabra que pueda ser aceptada, ver Anexo Figura 1, lo que se hace es que al insertar la palabra, el programa automáticamente evalúa ambas simulaciones y pone los resultados en un tabla, donde las filas rojas son las simulaciones de Monte Carlo, y las filas azules son las simulaciones de la Matriz. Adicionalmente, presenta una gráfica de que tan diferente es la probabilidad estimada ante la probabilidad del método de matrices.

Pestaña 2 Análisis de Punto de Corte, esta operación tiene dos sección. La primera es dado una palabra w se evalúa si $\mathcal{A}(w > \lambda)$ para $\lambda \in [0, 1]$, como en la Pestaña 1 también se hace de una vez ambos métodos y se ponen un una tabla (ver la imagen Anexo Figura 2). La segunda es dado un limite inferior o un intervalo cerrado, se quiere evaluar el numero de palabras que tienen cierta cantidad de letras cumplen con el punte de corte (ver la imagen Anexo Figura 3)[5].

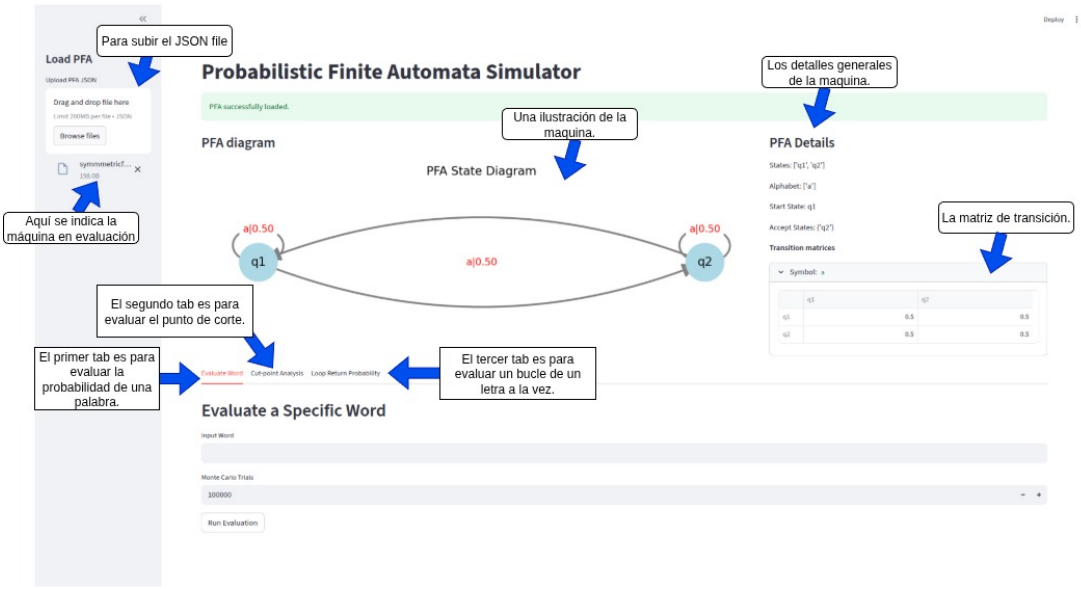
Pestaña 3 Evaluación de Bucle, lo que se hace aquí es bastante sencillo, dado un numero de repetición de una sola letra cual es la probabilidad que uno puede obtener (ver figura Anexo Figura 4), para ambos métodos y retorna un gráfico comparando el tiempo de completud contra el numero de repeticiones. Ahora note que el numero de repetición es finito, dado que las probabilidad en bucles para infinitas repeticiones usualmente no convergen (Rabin, 1963), lo que requieren métodos mas sofisticados, algo que en el siguiente proyecto esperaría cubrir teóricamente.

- La interfaz gráfica de programa: fue hecho con streamlit lo que facilita la interfaces tanto el backend como el frontend. Otro aspecto de Streamlit es que puedo usar la librería de networkx para hacer la ilustración visual que aparece al abrir una maquina, aunque siendo honesto no salen muy bien las transiciones de las maquinas, algo que en futuras entregas espero mejorar. Cambien quiero dejar dicho que aunque al principio propuse usar R como el lenguaje, rápidamente me di cuenta que la parte del interfaz del usuario salia mas eficiente en python. Pero lo interesante es que traducir código de Python a R es baste sencillo, lo que en futuras entregas posiblemente llegue hacer.

Abajo se presentara un complemento ilustrativo de como se mira el sistema, con indicaciones de que hace cada botón al ingresar una maquina:

[5] Ademas también le permite definir al usuario el tiempo para correr el programa.

Figura 5: Pagina de Inicio del Programa



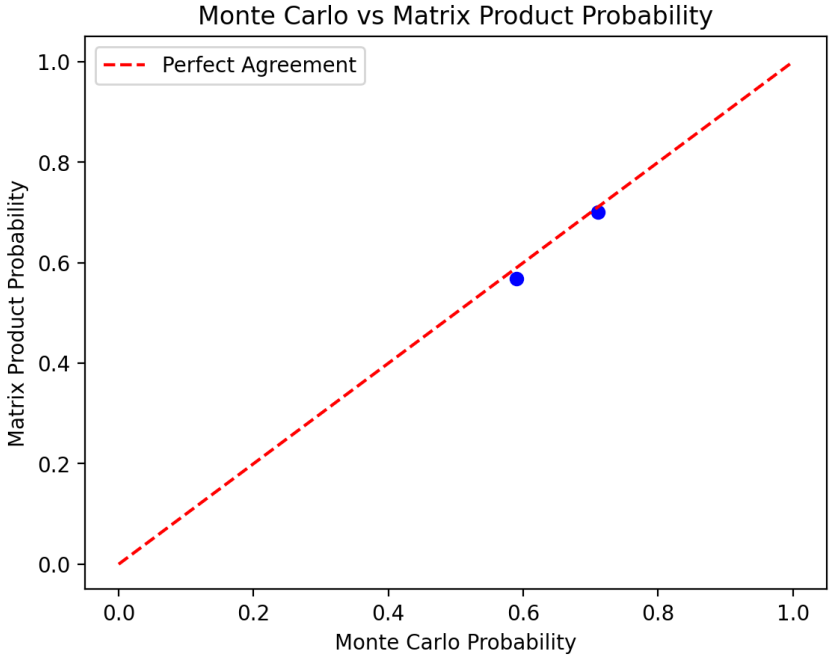
Teoría Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Resultados obtenidos por el programa

A continuación vamos analizar la maquina del *Ejemplo* 1, son dos aspectos importantes a comparar, primero la similitud de precisión en las probabilidades obtenidas por ambos métodos, el segundo es el tiempo de completud para ambos métodos. Notemos que el método de Matrices obtiene probabilidades exactas, mientras que el método de Monte Carlo son probabilidades estimadas. Por lo que continuación vamos a ver como el numero de simulaciones para el método de Monte Carlo mejora su estimación de probabilidad:

Tabla I: Ejemplo de Resultados de comparación entre Monte Carlo y Matriz

Word	Method	Probability	Avg. Path Prob	Stddev Path Prob	Time (s)	Trials
bb	Monte Carlo	0.711000	0.341393	0.170798	0.003062	1000
bb	Matrix Product	0.700000	–	–	0.000008	–
bbbb	Monte Carlo	0.590200	0.142804	0.092328	0.024961	5000
bbbb	Matrix Product	0.567861	–	–	0.000021	–



Teoría Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Otro aspecto a considerar es la estimación de los punto de cortes, se a observado de que en ciertos casos el método de Monte Carlo no concuerda con el mismo resultado obtenido con el método de matrices, esto es notable en el Anexo Figura 2 donde para la palabra aa la estimación de Monte Carlo obtuvo unos decimales de mas lo que hizo que contradijera con el método de matrices; a pesar que para ambas veces que se probo la palabra aa se hicieron 100000 repeticiones.

Pero lo mas interesante en las simulaciones de bucle, la siguiente imagen dice algo interesante, por lo que dice que

Tabla II: Ejemplo de simulaciones del Bucle

Symbol	k	Trials	Matrix Prob.	Monte Carlo Prob
b	811	5000	0.001342	0.6156
b	271	5000	0.078394	0.6150
b	1	5000	0.666667	0.6900

Teoría Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Notemos que las Probabilidades empiezan a diverge mientras mas repeticiones hagan en los bucles. Esto es debido a un error de programación de mi parte, algo que se va a evitar en futuras

entregas. El error aquí es que se programo las simulaciones de Monte Carlo para forzosamente siempre buscar la palabra adecuada, envés de correr cada simulación con diferentes caminos aunque esos caminos no lleguen a la palabra aceptada. Aun así otro aspecto interesante a notar es que la probabilidad de la palabra b empieza a converger a 0 por el numero de repeticiones, esto es algo que en futuras entregas se espera poder estudiar de manera mas sofisticada.

Por otro lado, el tema del tiempo de completud donde es mas evidente que el método de matrices es mas rápido es en los cálculos de bucles. Aunque teóricamente el método de matrices tiende a fallar para cálculos bastantes altos (Pouly, 2022), para los cálculos que se están evaluando en este programa. Las simulaciones de Monte Carlo se tardan significativamente mas que el Método de matrices, y esto claramente depende del numero de repeticiones que se usan.

4. Conclusión / Futuras Entregas

Ahora que se tiene una base de la Teoría de Maquinas Probabilísticas, esperaría poder seguir estudiando esta teoría para tener mas mecanismos que ofrece la teoría y así en la ultima entrega poder producir algún programa que pueda resolver una problemática. En la parte teórica lo que esperaría incluir en la siguiente entrega:

- Introducir mas operaciones para trabajar usando varias autómatas.
- Estudiar las siguen tas dos problemas de la Teoría:
  1. The Emptiness Problem
  2. The Isolation Problem
- Si el tiempo lo permite, incluir uno de los temas mas avanzados de la teoría[6]:
  1. The Value 1 problem
  2. The Value approximation problem
  3. Density Problem

En la parte computacional esperaría introducir al programa lo siguiente:

- Poder operar varias maquinas en el programa al mismo tiempo.
- Introducir computacionalmente los problemas elegidos en la teoría.

Anexo

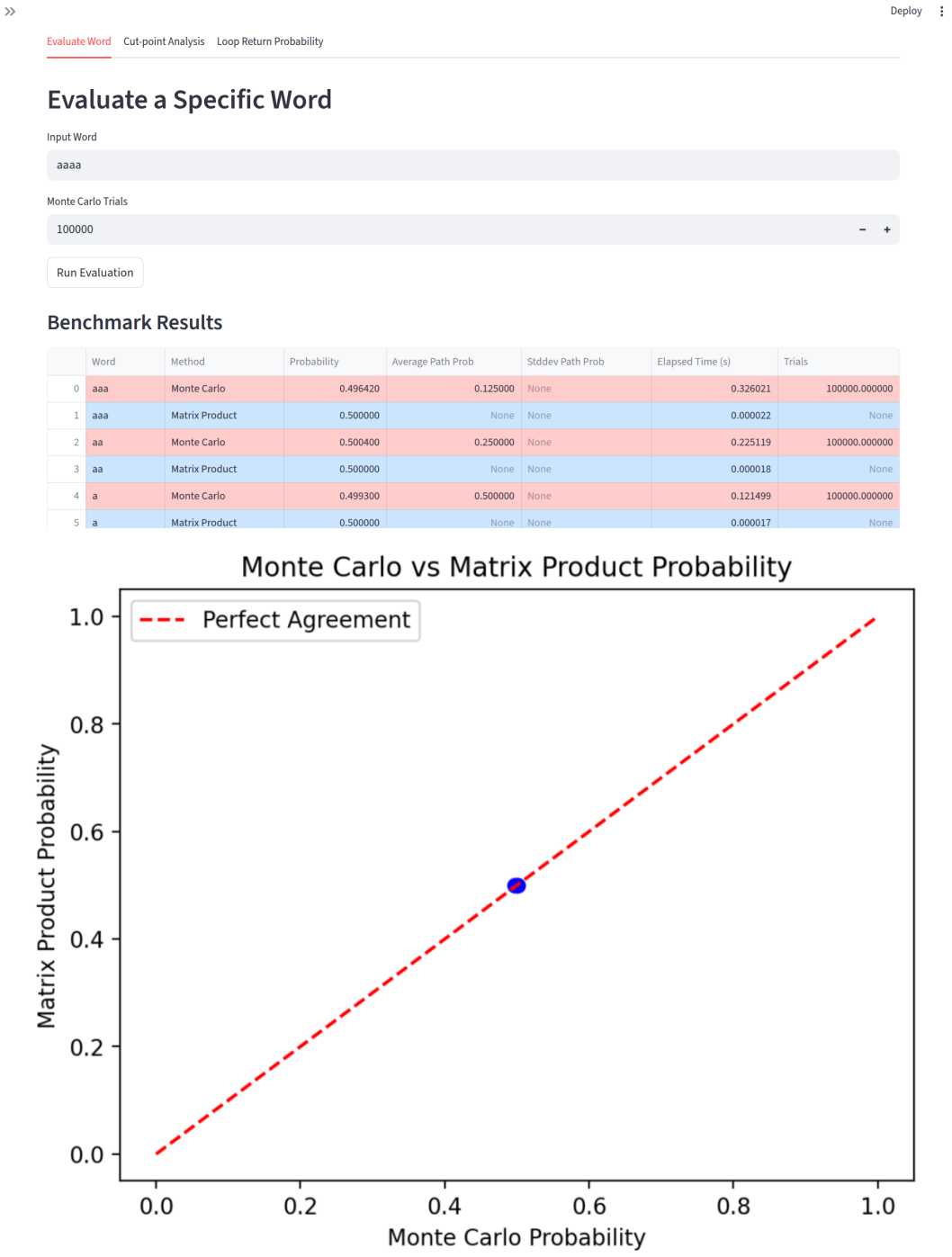
Estructura del JSON para guardar AP en el programa:

```
{
    "states": ["q0", "q1", "q2"],
    "alphabet": ["a", "b"],
    "transitions": {
        "(q0, 'a')": {"q0": 0.5, "q1": 0.5},
        "(q0, 'b')": {"q0": 1.0},
        "(q1, 'a')": {"q2": 1.0},
        "(q1, 'b')": {"q1": 0.7, "q2": 0.3},
        "(q2, 'a')": {"q2": 1.0},
        "(q2, 'b')": {"q2": 1.0}
    },
```

[6]Los siguientes problemas son generalizaciones de los problemas de Emptiness o Isolation.

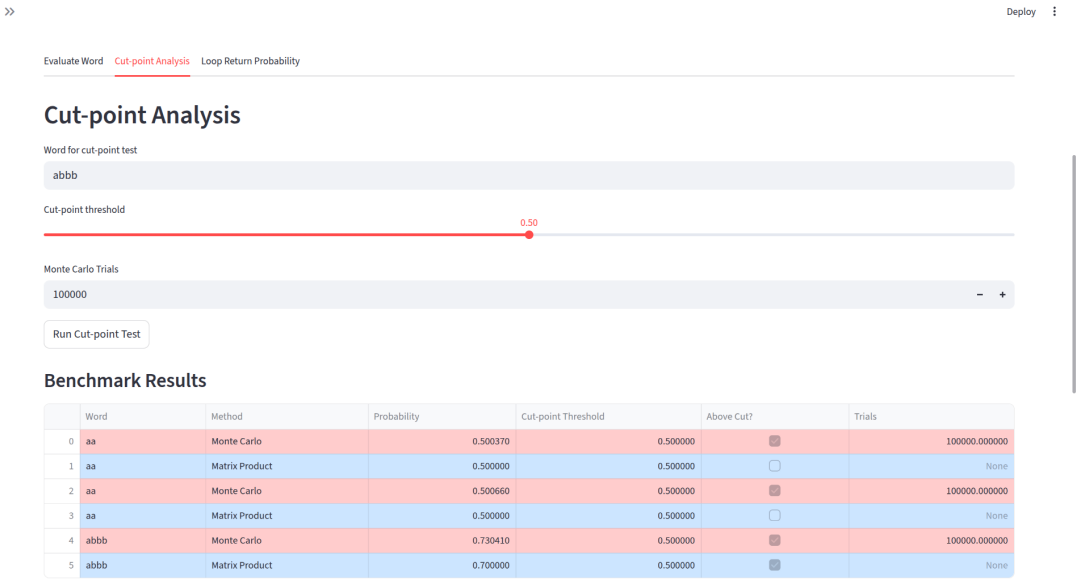
```
    "start_state": "q0",
    "accept_states": ["q2"]
}
```

Anexo Figura 1: Evaluación de Palabra



Teoría de la Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Anexo Figura 2: Evaluación de Punto de Corte por palabra



Teoría de la Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Anexo Figura 3: Evaluación de Punto de Corte todas las palabras

File change. Rerun Always rerun

Search Words by Cut-point or Interval

Cut-point threshold: 0.50

Interval low: 0.20

Interval high: 0.45

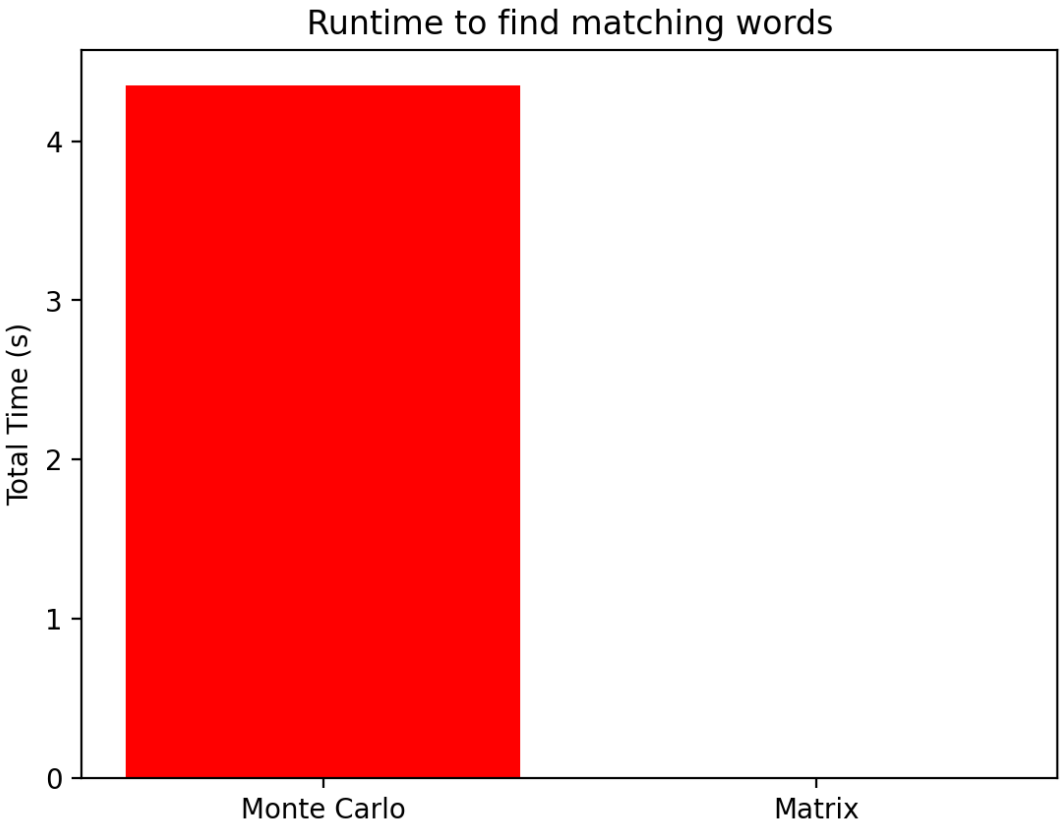
Max word length: 3

Time limit (seconds): 10

Search all Cut-points

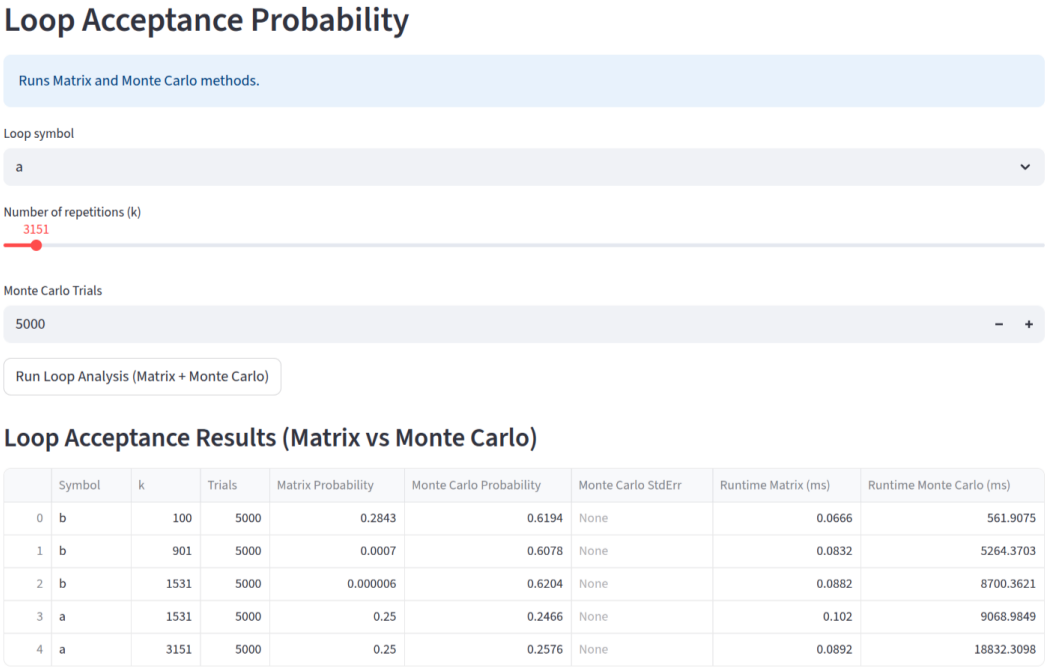
Found 3 words within criteria

	Word	Monte Carlo Prob	Matrix Prob	Cut-point	MC Time (s)	MM Time (s)
0	aaa	0.2521	0.25	0.2 0.45	0.3656	0.00004
1	aab	0.375	0.375	0.2 0.45	0.361	0.00003
2	bab	0.4749	0.4444	0.2 0.45	0.3828	0.00003



Teoría de la Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Anexo Figura 4: Evaluación de Bucles por letra



	Symbol	k	Trials	Matrix Probability	Monte Carlo Probability	Monte Carlo StdErr	Runtime Matrix (ms)	Runtime Monte Carlo (ms)
0	b	100	5000	0.2843	0.6194	None	0.0666	561.9075
1	b	901	5000	0.0007	0.6078	None	0.0832	5264.3703
2	b	1531	5000	0.000006	0.6204	None	0.0882	8700.3621
3	a	1531	5000	0.25	0.2466	None	0.102	9068.9849
4	a	3151	5000	0.25	0.2576	None	0.0892	18832.3098

Teoría de la Computación: Proyecto 1, UVG 2025; Python - versión 3.12.3

Referencias

OpenAI. (2025). Conversation with ChatGPT on Probabilistic Finite Automata [Accessed on September 2, 2025. Available from https://chat.openai.com].

Pouly, A. (2022). *Probabilistic Automata and Markov Chains: Lecture Notes for the Master Parisien de Recherche en Informatique, Course 2.16 – Finite Automata Based Computation Models, Academic Year 2021–2022* [Compiled on November 10, 2022, Available online as PDF]. Ver. 0.9993.

Rabin, M. O. (1963). Probabilistic Automata. *Information and Control*, *6*(3), 230-245. https://doi.org/10.1016/S0019-9958(63)90290-0