PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

# ANÁLISIS EXPERIMENTAL DE ALGORITMOS DE APROXIMACIÓN ALEATORIZADOS PARA CONTEO EN AUTÓMATAS

**RICARDO RAÚL RODRÍGUEZ REVECO**

Tesis para optar al grado de
Magíster en Ciencias de la Ingeniería

Profesor Supervisor:
MARCELO ARENAS, PH.D

Santiago de Chile, Diciembre 2023

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

# ANÁLISIS EXPERIMENTAL DE ALGORITMOS DE APROXIMACIÓN ALEATORIZADOS PARA CONTEO EN AUTÓMATAS

**RICARDO RAÚL RODRÍGUEZ REVECO**

Miembros del Comité:
MARCELO ARENAS, PH.D
CRISTIAN RIVEROS, PH.D
CLAUDIO GUTIÉRREZ PH.D
LEONARDO VANZI, PH.D

Tesis para optar al grado de
Magíster en Ciencias de la Ingeniería

Santiago de Chile, Diciembre 2023

*A mi querida madre que hubiese*
*estado orgullosa de leer esta tesis.*

## AGRADECIMIENTOS

Agradezco profundamente a mi supervisor Marcelo, a quien tuve la suerte de conocer durante mi segundo año de pregrado y me ha apoyado en muchos aspectos de la vida. También debo agradecer a Claudio Gutiérrez, Cristian Riveros y Leonardo Vanzi por ser parte del comité.

Le debo mucho a los profesores del grupo de teoría y datos de la UC, en particular a Juan Reutter y Domagoj Vroč por sus consejos y apoyo moral. También debo agradecerle a Pablo Barceló, Diego Arroyuelo y Miguel Romero por las conversaciones que hemos tenido, a Jorge Baier por su apoyo en situaciones complicadas, a Denis Parra y Marcelo Mendoza por su ayuda al lidiar con la burocracia de postgrado.

Le agradezco a la gente de la oficina del IMFD por recibirme en sus espacios, especialmente a Camila y Noemí.

Luego de una pandemia global, fue beneficioso volver a la presencialidad en el DCC, en la oficina O3 y en los pasillos, y agradezco mucho a Pavel Medina, Amaranta Salas y Francisca "@*gatochico*" Ibarra por colaborar a mantener mi salud mental estos meses. También estoy muy agradecido de compartir cotidianamente con otros estudiantes de postgrado, entre ellos Daniela Concha, Susana Figueroa, Julián García, Cristian Hinostroza, Alexander Pinto, Dante Pinto y Carlos Paredes.

Por último, quiero agradecer a mis amigos desde hace muchos años, Antonia, Pablo Vallejo, Simón, Poli, Pablito, Zamora, Jaime, Matías Villagra y Matías San Martín, por siempre estar presentes.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

## ABSTRACT

This thesis analyzes the performance of algorithms for the #NFA problem, which consists of counting the number of accepted words of a given length in nondeterministic finite automata (NFAs). This work contains the first experimental implementation and analysis of the randomized approximation algorithm proposed by Arenas et al. (2021). The algorithm was evaluated on three different families of NFAs, including the Tabakov-Vardi model of randomized NFAs.

Experimental results indicate that the algorithm of Arenas et al. fails to outperform traditional determinization-based algorithms in efficiency, even with relaxations in its parameters. This suggests that, although the algorithm presents a theoretical polynomial complexity, in reality it is not effective for realistic situations, thereby characterizing it as a possible 'galactic algorithm' (Lipton & Regan, 2013).

The thesis also recommends future research directions. This includes the development of alternative methods for generating randomized NFAs and improving the efficiency of the sampling subroutines of randomized approximation algorithms. In this way, the study breaks new ground in both the development of algorithms for the problem #NFA and in the methodologies for evaluating them.



**Keywords**: Randomized Algorithms, Computational Complexity, Automata, Experimental Analysis of Algorithms.

# RESUMEN

Esta tesis analiza el rendimiento de algoritmos para el problema #NFA, que consiste en contar la cantidad de palabras aceptadas, de un largo dado, en autómatas finitas no deterministas (NFAs). Este trabajo contiene la primera implementación y análisis empírico del algoritmo de aproximación aleatorizado propuesto por Arenas et al. (2021). Este estudio utiliza una metodología experimental para evaluar el algoritmo en tres familias diferentes de NFAs, incluyendo el modelo de Tabakov-Vardi de NFAs aleatorios.

Los resultados muestran que el algoritmo de Arenas et al. no logra superar en eficiencia a los algoritmos tradicionales de determinización, incluso con relajaciones en sus parámetros. Esto sugiere que, aunque el algoritmo presenta una complejidad polinomial teórica, en la realidad no resulta efectivo para situaciones realistas, caracterizándolo como un posible 'algoritmo galáctico' (Lipton & Regan, 2013).

Asimismo, la tesis recomienda direcciones de investigación futuras. Esto incluye el desarrollo de métodos alternativos para generar NFAs aleatorios y la mejora de la eficacia en las subrutinas de muestreo de los algoritmos de aproximación aleatorizados. De esta forma, el estudio abre nuevos caminos tanto en el desarrollo de algoritmos para el problema #NFA como en las metodologías de evaluación de estos.

**Palabras Claves**: Algoritmos Aleatorizados, Complejidad Computacional, Autómatas, Análisis Experimental de algoritmos.

# 1. INTRODUCCIÓN

## 1.1. Motivación

En esta tesis estudiaremos el problema de contar palabras en autómatas finitos no deterministas. Una de las principales razones para estudiar este problema viene de la teoría de bases de datos de grafos.

En abstracto, una base de datos de grafos es un grafo dirigido $G = (V, E)$ donde las aristas $E$ tienen etiquetas. Dado un camino $\pi$ en el grafo, compuesto por las aristas $e_1, \cdots, e_n$, llamamos etiqueta del camino a la palabra correspondiente a concatenar las etiquetas de las aristas que componen el camino.

Un modelo clásico de consultas en BBDD de grafos es el de *regular path queries* (Angles et al., 2017), donde recibimos una expresión regular y buscamos encontrar los caminos en el grafo cuya etiqueta pertenezca al conjunto definido por la expresión regular.

No es complicado argumentar que el problema de responder consultas es el problema más fundamental en bases de datos, y el enfoque clásico busca obtener todas las respuestas posibles a una consulta dada. Sin embargo, a medida que los volúmenes de datos aumentan, la cantidad de respuestas posibles también lo hace, de manera tal que obtener el conjunto completo de respuestas se vuelve poco factible en términos prácticos.

Por esta razón es que ha cobrado especial relevancia la idea de enumerar las soluciones con "small delay" (Segoufin, 2013). Esta idea consiste en proceder en dos fases, una primera de preprocesamiento donde se construyen estructuras de datos adecuadas para acelerar el proceso de computar las respuestas, y una segunda fase de enumeración. En esta última fase se entregan respuestas a las consultas con una demora pequeña entre ellas.

Ahora bien, surge la interrogante de cuando detener la enumeración, es para esto que necesitamos contar o bien estimar la cantidad de respuestas posibles.

Debido a lo anterior, cobra relevancia el problema #NFA, que recibe como entrada un NFA y un número en unario y entrega la cantidad de palabras de ese tamaño que son aceptadas por el NFA.

Otra motivación viene del mundo de la bioinformática (Kannan, Sweedyk, & Mahaney, 1995; Searls, 1993). Las secuencias de ADN se pueden modelar utilizando lenguajes regulares o libres de contexto y las expresiones regulares se utilizan para describir sitios de unión de proteínas en el ADN. En contextos biológicos, contar la cantidad de estos sitios corresponde a realizar una afirmación sobre la especificidad de una proteína.

En términos de la teoría de la complejidad computacional, el problema #NFA es catalogado como un problema difícil[1] (Álvarez & Jenner, 1993). Se considera inviable resolver de forma exacta los problemas pertenecientes a la clase de complejidad donde se encuentra #NFA (Valiant, 1979a), y por este motivo, el estudio de algoritmos que entreguen aproximaciones para el valor de #NFA resulta fundamental. Un reciente resultado teórico de (Arenas, Croquevielle, Jayaram, & Riveros, 2021) propone un algoritmo de aproximación aleatorizado que funciona en tiempo polinomial, sin embargo hasta este trabajo no existían análisis empíricos de este algoritmo, solo demostraciones en términos asintóticos del tiempo de ejecución, las que tienen la característica de ignorar factores constantes y solo observar el comportamiento para instancias a partir de un tamaño suficientemente grande.

## 1.2. Pregunta de investigación

En (Lipton & Regan, 2013) se introduce el concepto de algoritmos galácticos como algoritmos que tienen una complejidad asintótica mejor a los otros conocidos para un problema, pero el tamaño de instancia donde empiezan a superar a los otros algoritmos es enorme. Un ejemplo en el contexto de multiplicar números enteros es (Harvey & van der

[1] Se encuentra en la clase de complejidad #P, y de manera más precisa, en SPANL siendo completo para esta clase bajo reducciones en espacio logarítmico.

Hoeven, 2021), que ha sido publicado en la revista *Annals of Mathematics*. Este resultado entrega un algoritmo con complejidad asintótica $\mathcal{O}(n \log(n))$ para multiplicar números enteros. Sin embargo, el tamaño en bits de los números a partir del cual se tiene la cota asintótica corresponde a $2^{9^{12}} \approxeq 10^{85019762142}$. Lo anterior corresponde a números cuya cantidad de dígitos es mayor a $10^{10^{10}}$, y su valor es mayor a

$$2^{2^{9^{12}}}$$

En esta tesis exploramos la pregunta sobre si el algoritmo de (Arenas et al., 2021) es eficiente en la práctica o pertenece a la categoría de algoritmos galácticos. Para hacerlo realizaremos experimentos con diferencias familias de autómatas, una de ellas generada mediante un modelo aleatorio.

También exploramos la pregunta sobre si el algoritmo de (Arenas et al., 2021) funciona bien relajando los parámetros que son necesarios para la prueba de correctitud. Proponemos tres relajaciones del algoritmo y estudiamos su rendimiento.

### 1.3. Trabajos relacionados

El diseño de algoritmos de aproximación para problemas en la clase de complejidad #P ha sido estudiado desde la teoría y se han publicado varios resultados como por ejemplo (Kannan et al., 1995; M. Jerrum, Sinclair, & Vigoda, 2004; Bezáková, Štefankovič, Vazirani, & Vigoda, 2008; Karp, Luby, & Madras, 1989).

Cabe notar que el trabajo citado de (Kannan et al., 1995) propone un algoritmo de aproximación aleatorizado cuya complejidad asintótica es cuasi-polinomial [2] siendo el mejor algoritmo en estos términos, hasta la publicación de (Arenas et al., 2021), que consigue una complejidad asintótica polinomial.

[2] La complejidad asintótica de (Kannan et al., 1995) es $n^{\mathcal{O}(\log(n))}$

Salvo el caso del algoritmo de aproximación aleatorizado para #DNF, (Karp et al., 1989), que destaca por su simplicidad a la hora de ser implementado, prácticamente no existen trabajos donde se estudien estos algoritmos experimentalmente.

En efecto, el único análisis empírico de estos algoritmos que existe en la literatura, es un reciente trabajo por (Newman & Vardi, 2020) donde estudian un famoso FPRAS propuesto en 2004 por (M. Jerrum et al., 2004) para el cálculo del permanente de una matriz y las mejoras a este propuestas por (Bezáková et al., 2008).

Resulta sorprendente que a pesar de la gran influencia del artículo de Jerrum, Sinclair y Vigoda en la literatura posterior, el primer análisis empírico de este algoritmo haya aparecido 14 años después. Aún así el tamaño de las entradas con la cual se evalúa el algoritmo corresponde a matrices cuadradas de tamaños 6,8 y 10, siendo infactible en la práctica el poder ejecutar el algoritmo con matrices más grandes, en contraste con la afirmación de (Bezáková et al., 2008) sobre que las modificaciónes que proponen, harían posible calcular permanentes de matrices del orden de $100 \times 100$.

En otro orden de cosas, el estudio del rendimiento de algoritmos con generadores probabilistas de entradas bajo ciertos parámetros, es un tema interesante que no ha sido trabajado mucho en la literatura, más allá del caso del problema de satisficibilidad booleana, como por ejemplo en (Selman, Mitchell, & Levesque, 1996; Crawford & Auton, 1996; Gent & Walsh, 1996) y de algoritmos de determinización y universalidad para autómatas en el trabajo de (Tabakov & Vardi, 2005). En uno de los análisis que realizamos, utilizamos el modelo de autómatas aleatorios definido en (Tabakov & Vardi, 2005).

## 1.4. Contribuciones

Los algoritmos analizados en este trabajo no son originales del autor, pero hasta ahora no existía ningún análisis empírico de su rendimiento. Además, en general, los algoritmos

de aproximación para problemas en la clase de complejidad #P han sido estudiados principalmente mediante demostraciones teóricas de su complejidad asintótica, sin análisis experimentales detallados.

Los aportes de este trabajo son

(i) Realizamos la primera implementación del algoritmo publicado en (Arenas et al., 2021).
(ii) Propusimos e implementamos tres versiones modificadas del algoritmo de (Arenas et al., 2021).
(iii) Propusimos tres familias de autómatas a utilizar para evaluar el rendimiento de algoritmos que resuelvan el problema #NFA.
(iv) Comparamos, utilizando las familias propuestas, el tiempo de ejecución y la magnitud de los errores de aproximación del algoritmo de (Arenas et al., 2021) y sus variantes, con los de algoritmos clásicos para el problema #NFA.
(v) Concluimos a partir de los experimentos que el algoritmo de (Arenas et al., 2021) en su versión original no resuelve el problema #NFA dentro de un tiempo razonable.
(vi) Propusimos caminos para futuros desarrollos, tanto de algoritmos para el problema #NFA como de metodologías de evaluación para ellos.

## 2. PRELIMINARES

Para fijar notación, dados números naturales $n \leq m$ usaremos la notación $[n, m]$ para el conjunto $\{n, \ldots, m\}$. Además, usaremos $\log(x)$ para referirnos al logaritmo natural, es decir, en base $e$.

### 2.1. Relaciones y problemas

Sea $\Sigma$ un alfabeto finito con al menos dos símbolos. Por $\{0,1\}^*$ denotaremos el conjunto de todas las palabras finitas sobre el alfabeto binario $\{0,1\}$, $|x|$ indica la longitud de una palabra $x \in \{0,1\}^*$, $x_1 \cdot x_2$ corresponde a la concatenación de dos palabras, y $\{0,1\}^n$ denota el conjunto de todas las palabras $x \in \{0,1\}^*$ de largo $|n|$.

Representaremos un problema como una relación $R \subseteq \Sigma^* \times \Sigma^*$ donde en cada par $(x, y) \in R$ interpretaremos $x$ como la codificación de la entrada a algún problema e $y$ como la codificación de una solución a esa entrada, también llamada testigo o certificado.

Para cada $x \in \{0,1\}^*$, definimos el conjunto

$$W_R(x) = \{y \in \{0,1\}^* \mid (x, y) \in R\}$$

que llamaremos el conjunto de soluciones o de certificados de la instancia $x$

Para ejemplificar, podemos considerar el problema de satisfacibilidad booleana (SAT), dentro de este marco. Definiremos una relación RELATION-SAT como $(\varphi, \sigma)$ pertenece a la relación si y solo si $\varphi$ es la codificación de una fórmula en lógica proposicional y $\sigma$ es la codificación de una valuación que satisface a la fórmula.

Este marco es muy general, en esta tesis trabajaremos principalmente con lo que se conoce como $p$-relaciones (M. R. Jerrum, Valiant, & Vazirani, 1986). Diremos que una relación $R \subseteq \{0,1\}^* \times \{0,1\}^*$ es una $p$-relación si cumple dos condiciones. La

primera es que exista un polinomio $q$ tal que $(x, y) \in R$ implica que $|y| \leq q(|x|)$. La segunda es que exista una máquina de Turing determinista que reciba como entrada $(x, y) \in \{0, 1\}^* \times \{0, 1\}^*$, se ejecute en tiempo polinomial y acepte si y solo si $(x, y) \in R$. RELATION-SAT es un ejemplo de $p$-relación, pues cumple ambas condiciones. Dada una fórmula $\varphi$ de largo $|\varphi| = l$, sabemos que tiene a lo más $l$ variables proposicionales y el certificado $\sigma$ asociado a $\varphi$ es una valuación para $l$ variables, que podemos representar como una palabra binaria de largo $l$. Por lo anterior, tenemos que si $(\varphi, \sigma) \in$ RELATION-SAT, entonces $|\sigma| \leq |\varphi|$. La segunda condición también se cumple, pues dada una fórmula y una valuación, una maquina de Turing puede revisar en tiempo polinomial si la valuación satisface la fórmula.

Sin pérdida de generalidad asumiremos que para cualquier $p$-relación $R$ existe un polinomio $q$ tal que $|y| = q(|x|)$ para cualquier $(x, y) \in R$. Es importante notar que esta suposición implica que todos los certificados de la relación tienen el mismo largo, lo que se puede lograr rellenando con algún carácter especial hasta conseguir el largo adecuado.

### 2.2. Problemas de conteo

A partir de una relación $R$ podemos definir múltiples problemas asociados a ella. Uno de ellos es el problema de conteo asociado a una relación, que denotaremos como $\#R$

| | |
|---|---|
| **Problema:** | $\#R$ |
| **Entrada:** | Una palabra $x \in \{0, 1\}^*$ |
| **Salida:** | El tamaño $\lvert W_R(x)\rvert$ |

Dado que $|y| = q(|x|)$ para cada $(x, y) \in R$, tenemos que $W_R(x)$ es finito y por ende el problema está bien definido. Además, asumimos que $|W_R(x)|$ está codificado en binario y, por lo tanto, el tamaño de la salida es logarítmico en el tamaño de $W_R(x)$ y es polinomial en $|x|$.

En el caso de RELATION-SAT, el problema de conteo asociado, $\#$SAT corresponde a contar la cantidad de valuaciones que satisfacen una fórmula proposicional.

## 2.3. Clases de Complejidad de conteo

El problema abordado en esta tesis pertenece a la clase de complejidad #P. Esta clase se compone de funciones para las cuales existe una máquina de Turing no determinista que opera en tiempo polinomial tal que el valor de la función en una determinada entrada corresponde a la cantidad de caminos de aceptación de la máquina de Turing para dicha entrada.

La clase #P es por ende una clase de complejidad de funciones, es decir, sus elementos no corresponden a lenguajes como en el caso tradicional de clases de problemas de decisión, sino que a funciones, que generalmente representan el tamaño de un conjunto. Otra relevante clase de complejidad de funciones es la clase FP, que se considera la clase análoga a P en el contexto de complejidad de conteo.

Decimos que una función $f : \Sigma^* \mapsto \mathbb{N}$ pertenece a la clase FP si y sólo si existe una máquina de Turing determinista que opera en tiempo polinomial, que cuenta con dos cintas, una de lectura y otra de escritura, y al ser ejecutada con una palabra $w$ al detenerse, tiene el valor de la función, $f(w)$ en la cinta de escritura.

El artículo (Valiant, 1979a) presenta las primeras pruebas de problemas completos para la clase #P. Estos problemas incluyen el cálculo del permanente de una matriz y el conteo de las valuaciones que satisfacen una fórmula proposicional en forma normal conjuntiva (CNF). Posteriormente, se demostraron más problemas completos para #P, como se describe en (Valiant, 1979b). Uno de esos problemas es #DNF, es decir, el problema de contar las valuaciones que satisfacen una fórmula proposicional en forma normal disjuntiva (DNF). Recordemos que el problema de decidir si una fórmula en DNF

es satisfacible, se puede resolver en tiempo polinomial, a diferencia del problema para el caso general, que es NP-completo.

Esta clase es considerada difícil y la principal evidencia para conjeturar es el razonamiento de que si #P $\subseteq$ FP, podríamos contar en tiempo polinomial la cantidad de valuaciones que satisfacen una fórmula proposicional. Lo anterior implica que podríamos determinar en tiempo polinomial si una fórmula es satisfacible, con lo cual se tendría que P = NP.

Otra clase relevante, para el cual el problema central de esta tesis es completo, es la clase SPANL introducida en (Álvarez & Jenner, 1993). Antes de definir esta clase, recordaremos el concepto de transductor.

Una máquina de Turing no determinista se dice NL-transductor si cuenta con tres cintas, una solamente de lectura, otra solamente de escritura y una cinta de trabajo donde puede tanto leer como escribir. El cabezal de la cinta de escritura no puede moverse a la izquierda, por ende el transductor no puede leer lo que ha escrito en esa cinta. En la cinta de trabajo se puede utilizar una cantidad $O(\log(n))$ de celdas. Diremos que un NL-transductor $M$ calcula una función $f : \Sigma^* \mapsto \Sigma^*$ si con entrada $w$ en la cinta de lectura, el transductor se detiene en un estado final con $f(w)$ en la cinta de solo escritura.

Observemos que un NL- transductor es una máquina de Turing no determinista, por ende puede retornar distintos valores para ejecuciones distintas. A la cantidad de valores distintos que puede retornar un NL-transductor $M$ con entrada $w$ la llamaremos $\text{SPAN}_M(w)$.

Decimos que una función $f$ esta en SPANL si existe un NL- transductor $M$ con alfabeto de entrada $\{0, 1\}$ tal que el valor de la función $f(x)$ corresponde a la cantidad de valores que puede retornar un NL-transductor con entrada $x$. Esto quiere decir que SPANL

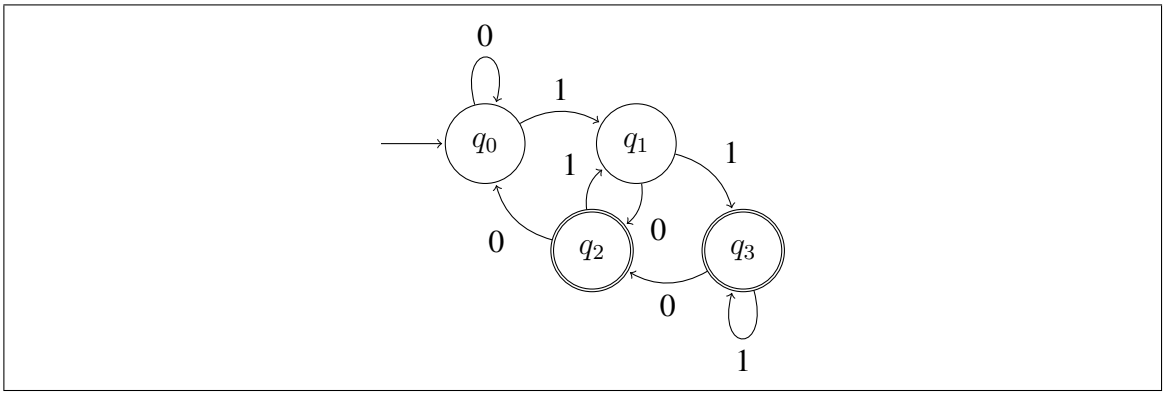


Figura 2.1. Un DFA que reconoce las palabras binarias cuyo penúltimo carácter es 1.

es el siguiente conjunto de funciones,

$$\text{SPANL} = \{f \mid f : \Sigma^* \mapsto \mathbb{N} \text{ y existe un NL-transductor } M \text{ tal que} f(x) = \text{SPAN}_M(x)\}$$

Esta clase esta contenida en la clase #P, y (Álvarez & Jenner, 1993) dan algunos indicios claves sobre la dificultad de esta clase. Uno de ellos es, que si SPANL $\subseteq$ FP entonces P = NP.

## 2.4. Autómatas y lenguajes regulares

Llamaremos autómata finito determinista, DFA por sus siglas en inglés, a una tupla $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ donde $Q$ es un conjunto finito de estados. $\Sigma$ es el alfabeto del automáta, $q_0 \in Q$ es el estado inicial. Los elementos de $F \subseteq Q$ se llaman estados finales y $\delta : Q \times \Sigma \mapsto Q$ es la función de transición.

En la figura 2.1 podemos ver un ejemplo de DFA, en este caso este DFA acepta las palabras binarias cuyo penúltimo símbolo es un 1.

De manera similar, llamaremos autómata finito no determinista, NFA por sus siglas en inglés, a una tupla $\mathcal{A} = (Q, \Sigma, I, F, \Delta)$ donde $Q$ es un conjunto finito de estados. $\Sigma$ es el

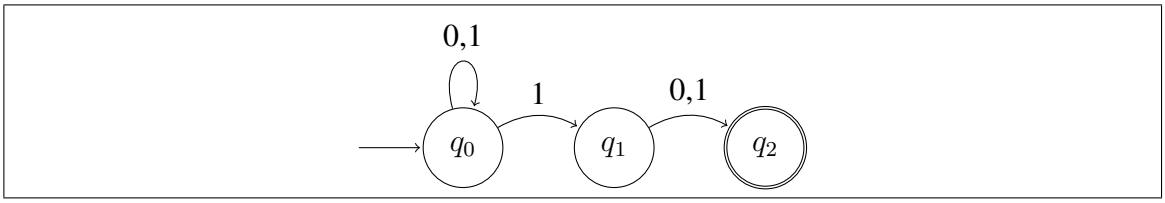


Figura 2.2. Un NFA que reconoce las palabras binarias cuyo penúltimo carácter es 1.

alfabeto del automáta, $I \subseteq Q$ es el conjunto de estados iniciales. Los elementos de $F \subseteq Q$ se llaman estados finales y $\Delta \subseteq Q \times \Sigma \times Q$ es la relación de transición.

En la figura 2.2 podemos ver un ejemplo de NFA que acepta el mismo lenguaje que el DFA anterior, podemos observar que el NFA tiene menos estados que el DFA.

Dado un autómata finito determinista, como la relación de transición es una función, cada palabra tiene a lo más una forma de ser ejecutada. Esto deja de ser cierto en el caso no determinista, debido a que la relación de transición no necesariamente es una función.

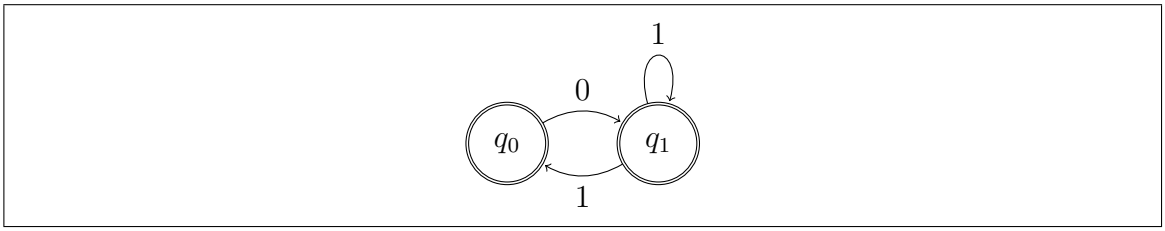


Figura 2.3. Un NFA con dos ejecuciones de aceptación para la palabra $01$.

En particular, en la figura 2.3 podemos ver un NFA que tiene dos ejecuciones de aceptación posibles para la palabra $01$.

Dado un autómata $\mathcal{A}$ y una palabra $w$ perteneciente a $\mathcal{L}(\mathcal{A})$, definimos la ambigüedad de $w$ en $\mathcal{L}(\mathcal{A})$, denotada por $\text{AMB}(\mathcal{A}, w)$, como la cantidad de ejecuciones de aceptación de la palabra en el autómata. En el autómata de la figura 2.4 la palabra $000$ tiene ambigüedad 2 y la palabra $0^6$ tiene ambigüedad 4. En efecto, en el caso de la figura, la

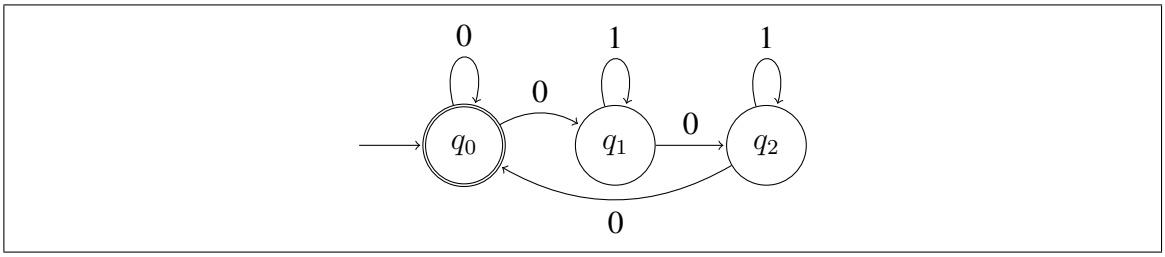


Figura 2.4. Un NFA con ambigüedad infinita

palabra $0^m$ tiene ambigüedad $2^{\lfloor \frac{m}{3} \rfloor}$, pues cada subpalabra $000$ tiene dos opciones para ser ejecutada.

A continuación, definimos la ambigüedad de un autómata $\text{AMB}(\mathcal{A})$ como el valor máximo, de existir, de $\text{AMB}(\mathcal{A}, w)$ para todas las palabras $w$ en $\mathcal{L}(\mathcal{A})$. En el caso de la figura 2.4, este máximo no existe, por ende diremos que el valor de $\text{AMB}(\mathcal{A})$ es infinito.

Si un autómata $\mathcal{A}$ tiene ambigüedad $\text{AMB}(\mathcal{A})$ igual a 1, decimos que es un autómata no ambiguo o UFA, por sus siglas en inglés. Claramente todos los DFAs son no ambiguos, sin embargo también existen NFAs no ambiguos.

# 3. CONTEO EN LENGUAJES REGULARES

El problema central de esta tesis es el de contar la cantidad de palabras de un largo determinado en un lenguaje regular dado por un autómata finito no determinista. Podemos formalizar este problema como el problema asociado a la siguiente relación.

$$\text{MEM-NFA} = \{((\mathcal{A}, 0^k), w) \mid \mathcal{A} \text{ es un NFA con alfabeto } \{0,1\}, w \in \{0,1\}^k \text{ y } w \text{ es aceptada por } \mathcal{A}\}$$

Llamaremos #NFA al problema de conteo asociado a MEM-NFA. Este problema consiste en, dados un NFA $\mathcal{A}$ y un valor $k$ dado en unario, codificado en la palabra $0^k$, calcular la cantidad de palabras $w$ de largo $k$ que son aceptadas por el NFA $\mathcal{A}$.

## 3.1. El caso determinista

El caso donde el autómata es determinista, que llamaremos #DFA, tiene un algoritmo polinomial sencillo.

Este algoritmo se basa en la observación de que en un autómata determinista cada palabra tiene exactamente una ejecución posible, por ende, existe una correspondencia entre contar palabras aceptadas por el autómata y contar caminos entre el estado inicial y algún estado final del autómata. Por lo anterior, en este caso el problema se reduce a contar caminos de largo $k$ en un grafo. Lo anterior es equivalente a calcular la $k-$ésima potencia de la matriz de adyacencia. Este cálculo se puede efectuar de manera eficiente a través de la exponenciación rápida. El algoritmo se puede ver en el pseudocódigo que sigue.

Este razonamiento se aplica también a los autómatas no ambiguos, en los cuales cada palabra que pertenece al lenguaje tiene una única ejecución de aceptación posible.

**Algoritmo 1** Algoritmo para contar la cantidad de palabras de largo $k$ aceptadas por un DFA

**Entrada:** Un DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ y un entero no negativo *k*

**Salida:** La cantidad de palabras de largo *k* aceptadas por el DFA $\mathcal{A}$

**function** CALCULARMATRIZADYACENCIA($\mathcal{A}$)
    $M \leftarrow$ matriz de adyacencia de $\mathcal{A}$
**return** $M$

**function** ELEVARMATRIZ($M, k$)
    $R \leftarrow$ matriz $M$ elevada a la potencia $k$
**return** $R$

**function** CONTARCAMINOS($M, q_{inicial}, F$)
    *caminos* $\leftarrow 0$
    **for all** $q_f \in F$ **do**
        *caminos* $\leftarrow caminos + M[q_{inicial}, q_f]$
    **return** *caminos*

$M \leftarrow$ CalcularMatrizAdyacencia($\mathcal{A}$)
$M^k \leftarrow$ ElevarMatriz($M, k$)
$q_{inicial} \leftarrow q_0$ estado inicial de $\mathcal{A}$
$F \leftarrow$ conjunto de estados finales de $\mathcal{A}$
*palabras* $\leftarrow$ ContarCaminos($M^k, q_{inicial}, F$)
**return** *palabras*

## 3.2. Salto de complejidad en el caso no determinista

La estrategia anterior que usamos para #DFA no funciona en el caso de autómatas finitos no deterministas, pues una palabra puede tener más de una ejecución posible. Esto rompe la correspondencia entre caminos entre el estado inicial y uno final en el grafo y ejecuciones de aceptación. En efecto, una palabra puede tener una cantidad exponencial de ejecuciones de aceptación, como se vio en el NFA de la figura 2.4.

La complejidad exacta de #NFA fue caracterizada en (Álvarez & Jenner, 1993) como completa para la clase SPANL. Nótese que esto implica que #NFA es difícil, puesto que si #NFA $\in$ FP, entonces se tendría que P $=$ NP.

## 3.3. Concepto de Fully Polynomial Randomized Approximation Scheme

Dada una relación $R$ y su problema de conteo asociado $\#R$, decimos que un algoritmo aleatorizado $\mathcal{A} : \{0,1\}^* \times (0,1) \to \mathbb{N}$ es un Fully Polynomial Randomized Approximation Scheme (FPRAS) (M. R. Jerrum et al., 1986) para el problema $\#R$ si existe un polinomio $q(u,v)$ tal que para cualquier $x \in \{0,1\}^*$ y $\delta \in (0,1)$, se tiene que:

$$\mathbb{P}(|\mathcal{A}(x,\delta) - |W_R(x)|| \leq \delta \cdot |W_R(x)|) \quad \geq \quad \frac{3}{4}$$

y el tiempo que se tarda en calcular $\mathcal{A}(x,\delta)$ es a lo más $q(|x|, \frac{1}{\delta})$. De este modo, $\mathcal{A}(x,\delta)$ aproxima el valor $|W_R(x)|$con un error relativo de $\delta$, y puede ser calculado en tiempo polinomial tanto en $x$ como en el valor $\frac{1}{\delta}$.

Un ejemplo clásico de FPRAS es el existente para #DNF, introducido en (Karp et al., 1989). Este algoritmo cuenta la cantidad de valuaciones que satisfacen una formula proposicional proporcionada en forma normal disyuntiva. Es relevante recordar que #DNF es un problema #P-Completo

Un hecho interesante es que el problema de determinar si una fórmula en forma normal disyuntiva es satisfacible, se puede resolver en tiempo polinomial, es decir, el problema de decisión asociado a este problema de conteo, es fácil.

No se conoce ningún FPRAS para $\#\mathrm{SAT}$ y se conjetura que no existen, pues un algoritmo de aproximación aleatorizado para contar valuaciones que satisfacen una fórmula proposicional, podría ser utilizado para construir un algoritmo aleatorizado de tiempo polinomial para $\mathrm{SAT}$.

Otro ejemplo relevante de FPRAS es el existente para calcular el permanente de una matriz con entradas no negativas, introducido en (M. Jerrum et al., 2004). En el caso de matrices con entradas 0-1, que sigue siendo #P-completo, podemos interpretar el permanente como la cantidad de perfect matchings en un grafo bipartito. En (Newman & Vardi, 2020) se analiza empíricamente el rendimiento de este algoritmo.

# 4. DESCRIPCIÓN DEL FPRAS

La entrada del problema #NFA es un NFA $\mathcal{A} = (Q, \{0,1\}, \Delta, I, F)$ con $m$ estados, una palabra $0^n$ que representa un número natural $n$ en unario, y un error relativo $\delta \in (0,1)$. El problema que buscamos resolver consiste en retornar un valor $N$ que sea una $(1 \pm \delta)$-aproximación de $|\mathcal{L}_n(\mathcal{A})|$, esto quiere decir que buscamos un valor que cumpla la siguiente desigualdad,

$$(1-\delta)|\mathcal{L}_n(\mathcal{A})| \ \leq \ N \ \leq \ (1+\delta)|\mathcal{L}_n(\mathcal{A})|$$

Además, esta aproximación debe ser calculada en tiempo polinomial para los parámetros $m$, $n$ y $\frac{1}{\delta}$.

En este capítulo, discutiremos el FPRAS presentado en (Arenas et al., 2021) para este problema, el cual consta de diversas partes que explicaremos por separado. En la primera sección, presentaremos una fase de preprocesamiento del algoritmo denominada "unroll" o "desenrollado", cuyo objetivo es crear un autómata sin ciclos que sea, desde cierto punto de vista, "equivalente" al autómata cuyas palabras de largo $n$ queremos contar.

En la segunda sección, abordaremos la idea principal en la que se basa la construcción del algoritmo. En la sección cuatro, aprenderemos a estimar el tamaño de ciertos lenguajes correspondientes a los caminos entre un estado inicial y un estado arbitrario del NFA. La penúltima sección se ocupa de la rutina de muestreo uniforme empleada por el FPRAS. Finalmente, proporcionaremos el pseudocódigo completo del algoritmo.

## 4.1. Unroll de un NFA

El algoritmo recibe tres entradas: un autómata $\mathcal{A}$, una palabra $0^n$, y un error relativo $\delta$ que se sitúa en el intervalo $(0,1)$. La primera acción del algoritmo es el "desenrollado" del autómata. Esta etapa implica la generación de un NFA sin ciclos, o visto de otra manera,

un DAG etiquetado. Este nuevo autómata tiene la característica de que el lenguaje que acepta coincide exactamente con $\mathcal{L}_n(\mathcal{A})$. Al autómata obtenido tras el desenrollado lo llamaremos $\mathcal{A}_{\text{unroll}}$. El propósito de construir un DAG proviene de la intención de aplicar la técnica de programación dinámica.

A partir de un autómata $\mathcal{A} = (Q, \Sigma, I, F, \Delta)$ y un valor $n$, llamaremos $\mathcal{A}_{\text{unroll}}$ al autómata obtenido mediante el siguiente procedimiento. Por cada estado $q$ en el conjunto de estados del autómata original $Q$, creamos $n+1$ copias $q^0, q^1 \cdots q^n$ de $q$ y las ponemos en el conjunto $Q_{\text{unroll}}$ que será el conjunto de estados del nuevo autómata. Por cada transición $(p, a, q)$ en el NFA original, crearemos transiciones $(p^\alpha, a, q^{\alpha+1})$, para $\alpha \in [0, n-1]$ en la relación de transición de $\mathcal{A}_{\text{unroll}}$, que llamaremos $\Delta_{\text{unroll}}$.

Nos referiremos al conjunto $Q^\alpha = \{q^\alpha \mid q \in Q\}$ como la capa $\alpha$ de $\mathcal{A}_{\text{unroll}}$. De forma similar, para cada conjunto $P \subseteq Q$, llamaremos $P^\alpha$ a la copia de $P$ contenida en la capa $\alpha$ de $\mathcal{A}_{\text{unroll}}$. De esta manera el conjunto $I^0$ corresponde a los estados iniciales del autómata original en la capa 0, que serán los estados iniciales de $\mathcal{A}_{\text{unroll}}$, y el conjunto $F^n$ a los estados finales del autómata original en la capa $n$, que serán los estados finales de $\mathcal{A}_{\text{unroll}}$.

Supondremos de ahora en adelante, que $\mathcal{A}_{\text{unroll}}$ se encuentra podado, es decir que se removieron todos los estados que no son parte de un camino entre un estado inicial y un estado final. Esto se puede hacer durante una fase de preprocesamiento en tiempo $O(nm)$, por lo que no afecta la complejidad del FPRAS.

## 4.2. Idea general

La idea general para construir el FPRAS para aproximar el valor #NFA, con entrada un NFA de $m$ estados, longitud de las palabras ser contadas $n$ y error relativo $\delta$ es como sigue. Partimos fijando el valor $\kappa = \lceil \frac{nm}{\delta} \rceil$. Luego para cada capa $\alpha$ y para cada estado $q$ tal que $q^\alpha \in Q_{\text{unroll}}$, almacenaremos un valor $N(q^\alpha)$ y un conjunto $S(q^\alpha) \subseteq \mathcal{L}(q^\alpha)$ de forma tal que se cumplan las dos condiciones que siguen:

(i) $N(q^\alpha)$ es una $(1 \pm \kappa^{-2})^\alpha$ aproximación de $|\mathcal{L}(q^\alpha)|$.

(ii) $S(q^\alpha)$ es una muestra de tamaño $2\kappa^7$, tomada de manera uniforme, de $\mathcal{L}(q^\alpha)$.

La primera condición quiere decir que

$$(1-\kappa^{-2})^\alpha |\mathcal{L}(q^\alpha)| \leq \ N(q^\alpha) \leq (1+\kappa^{-2})^\alpha |\mathcal{L}(q^\alpha)|$$

Cabe notar que si $\alpha = 0$, estamos pidiendo que $N(q^\alpha) = \mathcal{L}(q^\alpha)$. La segunda condición pide que $S(q^\alpha)$ sea una muestra uniforme de $\mathcal{L}(q^\alpha)$, por ende, es posible que se obtengan elementos repetidos al muestrear, debido a esto, $S(q^\alpha)$ es un multiconjunto (también llamado *bag*). En efecto, si $|\mathcal{L}(q^\alpha)| < 2\kappa^7$, necesariamente habrán elementos repetidos. Llamaremos "sketch" de $\mathcal{L}(q^\alpha)$ al par $(N(q^\alpha), S(q^\alpha))$. El algoritmo calcula mediante programación dinámica los sketches $(N(q^\alpha), S(q^\alpha))$ para cada estado $q^\alpha$ de $\mathcal{A}_{\text{unroll}}$ en un orden correspondiente a recorrer el DAG mediante búsqueda en amplitud. Es decir, antes de calcular el sketch de un estado en capa $\alpha + 1$, ya hemos calculado los sketches de cada estado de la capa $\alpha$. El valor que buscamos, es decir la estimación de $|\mathcal{L}(F^n)|$ es $N(F^n)$.

### 4.3. Estimación para un conjunto de vértices

Supongamos que conocemos $\text{SKETCH}[\alpha] = \{(N(q^\beta), S(q^\beta)\}_{\beta \leq \alpha \ q \in Q}$ para algún valor $\alpha$ en $[0, m-1]$. Recordemos que $N(q^\beta)$ es una $(1 \pm \kappa^{-2})^\beta$ aproximación de $|\mathcal{L}(q^\beta)|$ y $S(q^\beta)$ es una muestra uniforme de $\mathcal{L}(q^\beta)$ de tamaño $2\kappa^7$.

Nuestro objetivo es calcular $N(q^{\alpha+1})$ para todos los $q \in Q$. Para lograrlo, primero mostraremos como calcular una estimación de $|\mathcal{L}(P^\alpha)|$ para conjuntos de vértices $P \subseteq Q$ arbitrarios. Llamaremos $N(P^\alpha)$ a esta estimación. Posteriormente mostraremos como usar la estimación $N(P^\alpha)$ para calcular los valores $N(q^{\alpha+1})$.

Dado un subconjunto no vacío $P \subseteq Q$ buscamos una estimación $N(P^\alpha)$ de $|\mathcal{L}(P^\alpha)|$. Si $\mathcal{A}$ es un autómata determinista, los conjuntos $\{\mathcal{L}(p^\alpha) \mid p \in P\}$ son disjuntos, por ende

el valor $|\mathcal{L}(P^\alpha)|$ es simplemente igual a la suma de cada uno de estos conjuntos, es decir:

$$|\mathcal{L}(\mathcal{P}^\alpha)| = \sum_{p \in P} |\mathcal{L}(p)|$$

En el caso en que $\mathcal{A}$ no es determinista, como una palabra puede tener más de una ejecución, esta suma puede contar dos veces algunas palabras, por ende es necesario considerar las intersecciones de los conjuntos $\mathcal{L}(p^\alpha)$. El primer desafío a la hora de considerar esas intersecciones, es que no podemos calcular explícitamente todas ellas, pues requeriría una cantidad exponencial de tiempo. Para poder tratar con las intersecciones de estos conjuntos, fijamos una relación de orden total $\prec$ sobre el conjunto $P$ y consideramos la siguiente manera para calcular $|\mathcal{L}(P^\alpha)|$,

$$|\mathcal{L}(P^\alpha)| = \sum_{p \in P} |\mathcal{L}(p^\alpha)| \cdot \frac{|\mathcal{L}(p^\alpha) \setminus \bigcup_{q \in P \,:\, q \prec p} \mathcal{L}(q^\alpha)|}{|\mathcal{L}(p^\alpha)|}$$

Observemos que en el numerador de la fracción quitamos de $\mathcal{L}(p^\alpha)$ su intersección con todos los conjuntos $\mathcal{L}(q^\alpha)$ donde $q \prec p$. En efecto,

$$|\mathcal{L}(P^\alpha)| = \sum_{p \in P} |\mathcal{L}(p^\alpha) \setminus \bigcup_{q \in P \,:\, q \prec p} \mathcal{L}(q^\alpha)|$$

Llamaremos al valor,

$$\frac{|\mathcal{L}(p^\alpha) \setminus \bigcup_{q \in P \,:\, q \prec p} \mathcal{L}(q^\alpha)|}{|\mathcal{L}(p^\alpha)|}$$

la fracción de intersección del estado $p^\alpha$.

Inspirándonos en la ecuación de arriba, podemos estimar $|\mathcal{L}(P^\alpha)|$ utilizando el valor $N(p^\alpha)$ como estimador de $|\mathcal{L}(p^\alpha)|$ y usando el multiconjunto $S(p^\alpha)$ para calcular una estimación de la fracción de intersección de $p^\alpha$. Definiremos la estimación $N(P^\alpha)$ de $|\mathcal{L}(P^\alpha)|$ de la siguiente manera,

$$N(P^\alpha) = \sum_{p \in P} |N(p^\alpha)| \cdot \frac{|S(p^\alpha) \setminus \bigcup_{q \in P \,:\, q \prec p} \mathcal{L}(q^\alpha)|}{|S(p^\alpha)|}$$

Esta estimación puede ser calculada en tiempo polinomial respecto al tamaño del SKETCH$[\alpha]$. De hecho, el conjunto $S(p^\alpha) \setminus \bigcup_{q \in P \,:\, q \prec p} \mathcal{L}(q^\alpha)$ puede ser calculado iterando sobre las palabras $\omega \in S(p^\alpha)$ y revisando si la palabra $\omega$ pertenece a $\mathcal{L}(\{q^\alpha \mid q \in P \text{ y } q \prec p\})$. Debido a que verificar si una palabra pertenece a este lenguaje se puede realizar en tiempo polinomial, y la cantidad de palabras a testear es $2\kappa^7$, podemos calcular la estimación $N(P^\alpha)$ en tiempo polinomial. Llamaremos estimación de la fracción de intersección a la fracción

$$\frac{|S(p^\alpha) \setminus \bigcup_{q \in P \,:\, q \prec p} \mathcal{L}(q^\alpha)|}{|S(p^\alpha)|}$$

Para mostrar que $N(P^\alpha)$ es una buena aproximación de $|\mathcal{L}(\mathcal{P}^\alpha)|$, necesitamos que el estimador de de la fracción de intersección sea una buena aproximación de la fracción de intersección real en cada capa. Por buena aproximación entendemos que se debe cumplir la siguiente condición en cada capa $\alpha$,

$$\mathcal{E}(\alpha) \;:=\; \forall q \in Q \; \forall P \subseteq Q. \quad \left| \frac{|\mathcal{L}(q^\alpha) \setminus \bigcup_{p \in P} \mathcal{L}(p^\alpha)|}{|\mathcal{L}(q^\alpha)|} - \frac{|S(q^\alpha) \setminus \bigcup_{p \in P} \mathcal{L}(p^\alpha)|}{|S(q^\alpha)|} \right| \;<\; \frac{1}{\kappa^3}$$

El siguiente resultado, cuya demostración se encuentra en la proposición 6.4 de (Arenas et al., 2021) muestra que si $\mathcal{E}(\alpha)$ es una buena aproximación y tenemos una $(1 \pm \kappa^{-2})^\alpha$ aproximación de $|\mathcal{L}(p^\alpha)|$ para cada estado $p$, la estimación $N(P^\alpha)$ es buena.

**Proposición 4.1.** *Supongamos que la condición $\mathcal{E}(\alpha)$ se cumple y que $N(p^\alpha)$ es una $(1 \pm \kappa^{-2})^\alpha$-aproximación de $|\mathcal{L}(p^\alpha)|$ para cualquier $p \in Q$. Entonces se tiene que $N(P^\alpha)$ es una $(1 \pm \kappa^{-2})^{\alpha+1}$-aproximación de $|\mathcal{L}(P^\alpha)|$ para cualquier $P \subseteq Q$.*

Una vez que conocemos como calcular las estimaciones de $|\mathcal{L}(P^\alpha)|$ para cualquier subconjunto $P \subseteq Q$ en la capa $\alpha$, podemos calcular una estimación de $|\mathcal{L}(q^{\alpha+1})|$ de un estado $q$ en la capa $\alpha + 1$. Sea $q^{\alpha+1}$ un estado arbitrario en la capa $\alpha + 1$. Para un símbolo $b$ en el alfabeto $\{0, 1\}$ podemos definir el conjunto de estados $R_b = \{p^\alpha \in Q^\alpha \mid (p^\alpha, b, q^{\alpha+1}) \in \Delta_{\text{unroll}}\}$. En otras palabras, el conjunto $R_b$ es el conjunto de estados de la capa $\alpha$ desde el cual podemos llegar a $q^{\alpha+1}$ leyendo el símbolo $b$. Una observación

relevante es que los conjuntos $R_0$ y $R_1$ forman una partición de $\mathcal{L}(q^{\alpha+1})$ de la manera que sigue.

$$\mathcal{L}(q^{\alpha+1}) = \mathcal{L}(R_0) \cdot \{0\} \uplus \mathcal{L}(R_1) \cdot \{1\}$$

Donde interpretamos la concatenación de dos conjuntos $S_1, S_2$, denotada como por $S_1 \cdot S_2$ como el conjunto resultante de todas las concatenaciones de un elemento de $S_1$ con un elemento del segundo $S_2$.

A partir de la proposición 4.1 podemos deducir las siguientes dos proposiciones.

**Proposición 4.2.** *Supongamos que la condición $\mathcal{E}(\beta)$ se cumple para todos los $\beta \leq \alpha$. Entonces $N(p^{\alpha+1})$ es una $(1 \pm \kappa^{-2})^{\alpha+1}$-aproximación de $|\mathcal{L}(p^{\alpha+1})|$ para cada $p \in Q$.*

**Proposición 4.3.** *Si se cumple $\mathcal{E}(\beta)$ para todos los $\beta \leq n$, entonces $N(F^n)$ es una $(1 \pm \delta)$-aproximación para $|\mathcal{L}_n(\mathcal{A})|$.*

De esta manera, a partir del SKETCH$[\alpha]$ logramos calcular $N(q^{\alpha+1})$ para cualquier $q^{\alpha+1}$ arbitrario en la capa $\alpha + 1$. Para poder completar la programación dinámica, resta calcular los conjuntos $S(q^{\alpha+1})$, es decir, generar una muestra aleatoria, bajo la distribución uniforme, del conjunto $\mathcal{L}(q^{\alpha+1})$. Esto será el tema a verá en la sección que sigue.

## 4.4. Muestreo

El paso faltante para completar el algoritmo consiste en generar, de manera uniforme, palabras de $\mathcal{L}(q^{\alpha+1})$. Para esto los autores se inspiran en una técnica propuesta por (M. R. Jerrum et al., 1986) adaptándola a nuestro contexto.

Sea $q \in Q$ un estado y $\alpha$ una capa tal que $\alpha \leq n$, supongamos que para cada capa $\beta < \alpha$ se cumple la condición $\mathcal{E}(\beta)$. Mediante el procedimiento de la sección anterior, tenemos las estimaciones $N(p^{\alpha})$ para todos los estados en la capa $\alpha$. Buscamos generar de manera aleatoria, bajo la distribución uniforme, elementos de $\mathcal{L}(q^{\alpha})$. Para lograrlo,

procedemos de la siguiente manera. Primero definiremos a la palabra $w^\alpha$ como la palabra vacía, que denotaremos como $\lambda$, posteriormente construiremos una sucesión de palabras $w^\alpha, w^{\alpha-1}, \cdots, w^1, w^0$ donde cada elemento $w^b$ tiene la forma $b_\beta \cdot w^{\beta+1}$ donde $b_\beta \in \{0,1\}$, definiremos como resultado de este procedimiento a la palabra $w^0$. Dicho de otra manera, construiremos una palabra $w^0$ del lenguaje $\mathcal{L}(q^\alpha)$ construyendo un sufijo de la muestra, bit por bit.

Necesitamos asegurar que la palabra $w^0$ haya sido elegida de $\mathcal{L}(q^\alpha)$ con probabilidad uniforme, para conseguir esto, consideraremos una sucesión de conjuntos $P^\alpha, P^{\alpha-1}, \cdots, P^1, P^0$ que construiremos de la siguiente forma.

El primer conjunto es $P^\alpha = \{q^\alpha\}$, luego consideraremos el conjunto de vértices en la capa anterior, $\alpha - 1$, desde donde podemos llegar al conjunto $P^\alpha$ al leer el carácter $b \in \{0,1\}$. Dicho de otra manera definimos el conjunto $P_b^\alpha$ como,

$$P_b^\alpha = \{p^{\alpha-1} \in Q^{\alpha-1} \mid \text{existe un } r^\alpha \in P^\alpha \text{ tal que } (p^{\alpha-1}, b, r^\alpha) \text{ es una transición en } \mathcal{A}_{\text{unroll}}.$$

Una observación relevante es que $P_b^\alpha$ son estados en la capa $\alpha - 1$, aunque el superíndice sea $\alpha$. De forma similar a la sección anterior estos conjuntos inducen una partición del lenguaje $\mathcal{L}(P^\alpha)$ de la manera que sigue:

$$\mathcal{L}(P^\alpha) \;=\; \mathcal{L}(P_0^\alpha) \cdot \{0\} \;\uplus\; \mathcal{L}(P_1^\alpha) \cdot \{1\}$$

El algoritmo procede primero calculando las estimaciones, de acuerdo a la sección anterior, $N(P_b^\alpha)$ de los tamaños de los conjuntos $\mathcal{L}(P_b^\alpha)$ para $b \in \{0,1\}$. Una vez conocidas estas estimaciones, elige uno de los conjuntos de la partición de acuerdo al tamaño estimado. Esto es con probabilidad $\dfrac{N(P_0^\alpha)}{N(P_0^\alpha) + N(P_1^\alpha)}$ escogemos el conjunto $P_0^\alpha$ y con probabilidad $\dfrac{N(P_1^\alpha)}{N(P_0^\alpha) + N(P_1^\alpha)}$ escogemos el conjunto $P_1^\alpha$.

Supongamos que escogimos el conjunto $P_b^\alpha$, entonces añadimos b al sufijo que estamos construyendo. Esto es, definimos $b_{\alpha-1} = b$, agregamos el bit $b_{\alpha-1}$ como un prefijo

de $w^\alpha$ para obtener $w^{\alpha-1} = b_{\alpha-1} \cdot w^\alpha$, definimos $P^{\alpha-1}$ como $P_b^\alpha$, y continuamos con la recursión en $w^{\alpha-1}$ y $P^{\alpha-1}$.

Observemos que $P^\beta$ es el conjunto de vértices tal que existe un camino etiquetado por $w^\beta$ que conecta algún estado de $P^\beta$ con $q^\alpha$. Debido a que la estimación del tamaño de las particiones puede no ser exacta, podría ser el caso de que elegimos algunas palabras con una probabilidad ligeramente mayor que otras. Para evitar esto y obtener un muestreo perfectamente uniforme, en cada paso almacenamos la probabilidad de la partición que escogimos. Con esta información podemos calcular la probabilidad exacta $\varphi$ con la cual fue elegida la palabra $w$ que generamos. Finalmente, rechazamos esta muestra con probabilidad proporcional a $\varphi$, con lo que obtenemos un muestreo perfectamente uniforme. Mientras ninguna palabra sea significativamente más probable de muestrear que otra, la probabilidad de rechazar una muestra será constante y para obtener una palabra con probabilidad $1 - \mu$, para una parámetro de falla $\mu > 0$, simplemente podemos ejecutar el muestreo $\mathcal{O}(\log(\frac{1}{\mu}))$ veces.

**Algoritmo 2** Algoritmo para generar de manera uniforme una palabra de $P^\alpha$

**Entrada:** $\beta, P^\beta, w^\beta, \varphi$

**Salida:** Retorna $w^0$ con probabilidad $\varphi$, o retorna **fail**

1: **function SAMPLE**($\beta, P^\beta, w^\beta, \varphi$)
2: **if** $\beta = 0$ **then**
3: con probabilidad $\varphi$, **return** $w^0$, en otro caso, **return fail**
4: **else**
5: **for all** $b \in \{0, 1\}$ **do**
6: $P_b^\alpha \leftarrow \{p^{\alpha-1} \in Q^{\alpha-1} \mid$ existe $r^\alpha \in P^\alpha$ tal que $(p^{\alpha-1}, b, r^\alpha)$ es una transición en $A_{unroll}\}$
7: $p_b \leftarrow \dfrac{N(P_b^\beta)}{N(P_0^\beta) + N(P_1^\beta)}$
8: Con probabilidad $p_i$ escogemos $b \leftarrow i$, con $i \in \{0, 1\}$
9: $P^{\beta-1} \leftarrow P_b^\beta$
10: $w^{\beta-1} \leftarrow b \cdot w^\beta$
11: **return SAMPLE**($\beta - 1, P^{\beta-1}, w^{\beta-1}, \frac{\varphi}{p_b}$)

Podemos ver el pseudocódigo del algoritmo en la figura. Realizamos la primera llamada con los parámetros **Sample**$(\alpha, \{q^\alpha\}, \lambda, \varphi_0)$ donde $\lambda$ es la palabra vacía, pues el objetivo es obtener de manera uniforme un elemento de $\mathcal{L}(P^\alpha) = \mathcal{L}(q^\alpha)$. El valor $\varphi_0$ es un valor que determinaremos más adelante. Notemos que en cada paso de este algoritmo, $|\mathcal{L}(P^\beta)|$ es exactamente la cantidad de palabras en $\mathcal{L}(q^\alpha)$ que tienen el sufijo $w^b$, y $\mathcal{L}(P^\beta)$ es el conjunto de strings $x$ tal que $x \cdot w^\beta \in \mathcal{L}(q^\alpha)$. Observemos que este conjunto $P^\beta$ depende de la palabra generada aleatoriamente $w^\beta$, pero por simplicidad escribimos $P^\beta$ en vez de $P^\beta_{\omega^\beta}$.

Para entender la intuición detrás de este algoritmo, supongamos por un momento que podemos calcular de manera exacta las probabilidades asociadas a las particiones, esto

es $p_b = \frac{|\mathcal{L}(P_b^\beta)|}{|\mathcal{L}(P^\beta)|}$. Descartemos por el momento la probabilidad de retornar **fail**. Dada una palabra $x \in \mathcal{L}(q^\alpha)$, ¿Cuál es la probabilidad de elegir como $w^0$ a la palabra $x$? Como descartamos el caso de un error en el muestreo, se tiene que $w^0 = \textbf{Sample}(\alpha, \{q^\alpha\}, \lambda, \varphi_0)$. La probabilidad de haber elegido $w^0 = b_0 \cdots b_l$ corresponde al producto de las probabilidades de haber elegido, en cada paso, al bit $b_i$, $i \in [l]$. Esto es,

$$\mathbf{Pr}(\{w^0 = x\}) = \frac{|\mathcal{L}(P^{\alpha-1})|}{|\mathcal{L}(P^\alpha)|} \cdot \frac{|\mathcal{L}(P^{\alpha-2})|}{|\mathcal{L}(P^{\alpha-1})|} \cdot \frac{|\mathcal{L}(P^{\alpha-3})|}{|\mathcal{L}(P^{\alpha-2})|} \cdot \cdots \cdot \frac{|\mathcal{L}(P^1)|}{|\mathcal{L}(P^2)|} \cdot \frac{1}{|\mathcal{L}(P^1)|} = \frac{1}{|\mathcal{L}(P^\alpha)|}$$

que es exactamente la probabilidad de elegir un elemento bajo la distribución uniforme.

Ya vimos que si pudiésemos calcular de manera exacta el tamaño de las particiones, podríamos generar muestras de manera perfectamente uniforme. Con alta probabilidad esto también se cumple cuando aproximamos las probabilidades $p_b$ con $N(P_b^\beta)/(N(P_0^\beta) + N(P_1^\beta))$. Esto se ve en la siguiente afirmación que corresponde a la proposición 6.7 en (Arenas et al., 2021).

**Proposición 4.4.** *Supongamos que para todas las capas $\beta < \alpha$ se cumple la condición $\mathcal{E}(\beta)$.*

$$\mathbf{Pr}(\{w = x\}) = \frac{e^{-5}}{N(q^\alpha)}.$$

*Además, el algoritmo entrega* **fail** *con probabilidad a los más $1 - e^{-9}$. Es decir, condicionado en no retornar un error muestral, el algoritmo* ***Sample***$(\alpha, \{q^\alpha\}, \lambda, \frac{e^{-5}}{N(q^\alpha)})$ *entrega un elemento $x \in \mathcal{L}(q^\alpha)$ elegido desde la distribución uniforme.*

### 4.5. Algoritmo completo

Finalmente juntando, los procedimientos vistos en las secciones anteriores, podemos ver el pseudocódigo del algoritmo completo.

En la línea 18 del algoritmo tenemos un valor $c(\kappa)$ que corresponde a la cantidad de intentos que realiza la subrutina de muestreo, y retorna error de muestreo si no consigue

**Algoritmo 3** FPRAS para #NFA

**Entrada:** Un NFA $\mathcal{A}$, un valor $n$ dado en unario como $0^n$, un parámetro de error relativo $\delta \in (0,1)$

**Salida:** Una $(1 \pm \delta)$ aproximación de $|\mathcal{L}_n(\mathcal{A})|$

**function FPRAS**$(\mathcal{A}, 0^n, \delta)$
  **if** $\mathcal{L}_n(\mathcal{A}) = \emptyset$ **then return** $0$
  Construimos el autómata desenrollado $\mathcal{A}_{\text{unroll}}$ y lo podamos, removiendo los estados inalcanzables y los estados desde los cuáles no se puede llegar a un estado final.
  **for all** $q^0 \in I^0$ **do**
    $N(q^0) \leftarrow 1$
    $S(q^0) \leftarrow \{\lambda\}$
  **for all** $\alpha \in \{1, \cdots, n\}$ **do**
    **for all** $q^\alpha \in \mathcal{A}_{\text{unroll}}$ **do**
      **for all** $b \in \{0,1\}$ **do**
        $R_b \leftarrow \{p^{\alpha-1} \in Q^{\alpha-1} \mid (p^{\alpha-1}, b, q^\alpha)$ es una transición en $\mathcal{A}_{\text{unroll}}\}$
      $N(q^\alpha) \leftarrow N(R_0) + N(R_1)$
      $S(q^\alpha) \leftarrow \emptyset$
      **while** $|S(q^\alpha)| < 2\kappa^7$ **do**
        $i = 0$
        **while** $i < c(\kappa)$ **do**
          $w = \textsc{Sample}(\alpha, q^\alpha, \lambda, \frac{e^{-5}}{N(q^\alpha)})$
          **if** $w \neq$ **fail then**
            **break**
        **if** $w =$ **fail then**
          **return sampling error**
        **else**
          $S(q^\alpha) \leftarrow S(q^\alpha) \cup \{w\}$
  **return** $N(F^n)$

una palabra en esa cantidad de intentos. $\Theta(\log(\kappa))$ Para que el tiempo de ejecución del algoritmo sea polinomial, basta que este valor sea. Elegimos como valor,

$$c(\kappa) \quad = \quad \left\lceil \frac{2 + \log(4) + 8\log(\kappa)}{\log((1 - e^{-9})^{-1})} \right\rceil$$

el cuál es $\Theta(\log(\kappa))$.

# 5. MARCO EXPERIMENTAL

Este capítulo está compuesto por tres partes; en la primera sección de este capítulo veremos la descripción de tres familias de autómatas que elegimos para llevar a cabo las pruebas. Después, se detallarán los algoritmos que buscamos evaluar, y finalmente se especificará la metodología con la que se ejecutaron los experimentos.

## 5.1. Familias de autómatas

Elegimos tres familias de autómatas, cada una con características particulares, para evaluar el rendimiento de los distintos algoritmos de conteo. La primera familia que elegimos corresponde a $\text{NID}_m$, elegida por ser una familia de NFAs cuyo NFA no ambiguo más pequeño tiene una cantidad exponencial de estados respecto al NFA. La segunda familia corresponde a una sucesión de NFA cuya densidad va decayendo, elegida para ver como la densidad afecta el comportamiento de la subrutina de muestreo. Finalmente, la tercera familia corresponde a un modelo de NFA aleatorios introducidos por (Tabakov & Vardi, 2005).

### 5.1.1. Autómatas para NID

Dado un número entero $m$, consideremos el lenguaje regular:

$$\text{NID}_m = \{u \in \{0,1\}^* | \exists i : u_i \neq u_{i+m}\}$$

Este lenguaje corresponde a las palabras binarias en las cuales existen dos caracteres diferentes a una distancia de $m$ posiciones. De lo anterior podemos notar que toda palabra en $\text{NID}_m$ tiene longitud al menos $m+1$ y para todo largo mayor a $m+1$ existe al menos una palabra de ese largo en el lenguaje.

En efecto, la cantidad exacta de palabras de largo $k$ en el lenguaje $\text{NID}_m$ es:

$$\text{NID}_m \cap \{0,1\}^k = \begin{cases} 2^{k-m} & \text{si } k \geq m+1 \\ 0 & \text{si } k < m+1 \end{cases} \tag{5.1}$$

El NFA minimal para este lenguaje tiene exactamente $2m+2$ estados y su ambigüedad es $O(m)$. En la figura 5.1 podemos ver un NFA minimal para $\text{NID}_2$.

Este lenguaje fue introducido en (Hromkovič, Seibert, Karhumäki, Klauck, & Schnitger, 2002). Cuenta con la característica de ser difícil de desambiguar. En efecto, todo NFA con ambigüedad $k$ para $\text{NID}_m$ tiene tamaño al menos $2^{\frac{m}{k}} - 1$ y en particular, todo NFA no ambiguo para este lenguaje tiene $2^m - 1$ estados.

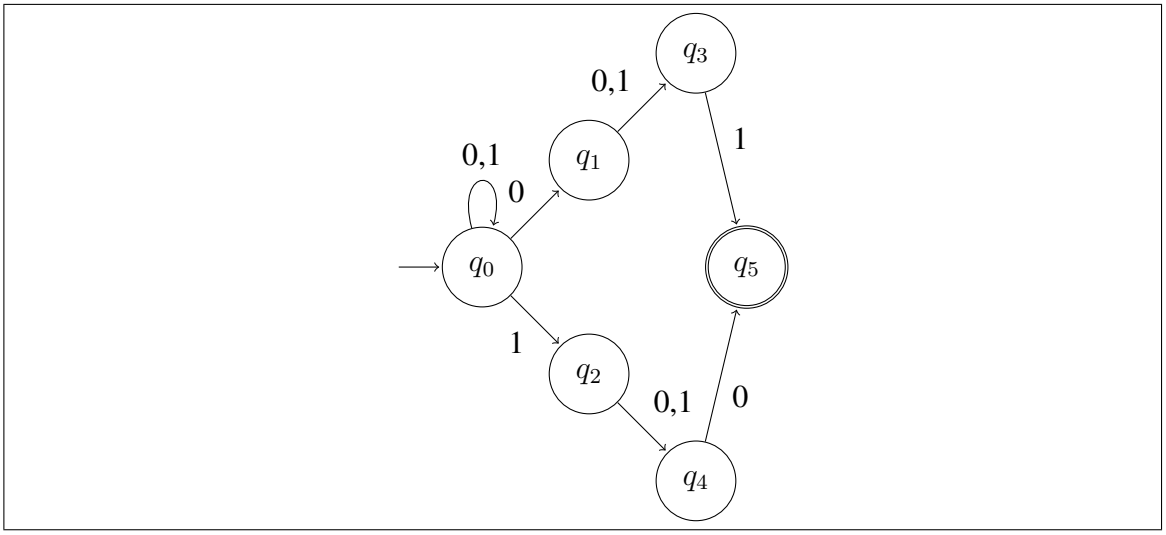


Figura 5.1. El NFA minimal para $\text{NID}_2$, de 6 estados

### 5.1.2. Autómatas de densidad $\frac{1}{2^k}$

Para el análisis del FPRAS, dado que implica muestreo aleatorio de un lenguaje, resulta interesante preguntarse que sucede a medida que consideramos lenguajes cada vez menos densos. Podemos construir un autómata que de las $2^k$ palabras binarias de

largo $k$ acepta $\frac{1}{2^k}$ fijando un valor determinado para las primeras $k-1$ posiciones y luego permitiendo cualquier carácter en la última posición. Dicho de otra manera, podemos considerar el lenguaje de las expresiones regulares de forma $0^{k-1}(0+1)$ y DFA $\mathcal{D} = (Q, \{0,1\}, \delta, q_0, q_f)$ donde $Q = \{q_0, \cdots, q_k\}$, el estado final es $q_f = q_k$ y la función de transición se define como:

$$\delta(p,a) = \begin{cases} q_{i+1} & \text{si } p = q_i, i \in \{0, \cdots, k-2\} \text{ y } a = 0 \\ q_k & \text{si } p = q_{k-1} \text{ y } a \in \{0,1\} \end{cases}$$

A modo de ejemplo, en la figura 5.2 podemos ver un autómata de densidad $\frac{1}{2^4}$ para el lenguaje de la expresión $0^3(0+1)$

De esta manera, la cantidad de palabras de longitud $l$ pertenecientes al lenguaje es cero si $k$ es distinto de $l$, y $2^{k-1}$ si $k$ es igual a $l$.

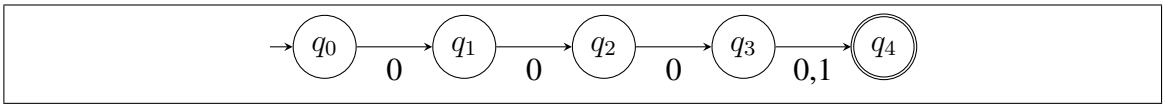


Figura 5.2. Un DFA que acepta un lenguaje de densidad $\frac{1}{2^4}$

### 5.1.3. Autómatas aleatorios de Tabakov-Vardi

En contraste con los modelos de grafos aleatorios, la generación aleatoria de autómatas ha sido un tema al que se le ha dedicado escasa atención. Moshe Vardi y Deian Tabakov propusieron en (Tabakov & Vardi, 2005) un modelo probabilístico para crear autómatas, que utilizaron para analizar el desempeño de algoritmos tradicionales en problemas de minimización de autómatas y verificación de universalidad. Aunque ambos problemas son relevantes en la práctica, resultan ser completos para la clase PSPACE.

Ellos señalan el estudio de instancias aleatorizadas de SAT (Selman et al., 1996) como la inspiración para el modelo.

Además del trabajo en el que se presentó el modelo, este también ha sido empleado para comparar el rendimiento de algoritmos en autómatas en otros estudios, incluyendo (Tabakov & Vardi, 2007), (Doyen & Raskin, 2009) y (Holík, Meyer, & Muskalla, 2016).

Describiremos el proceso mediante el cual el modelo aleatorio de Tabakov-Vardi genera un autómata aleatorio $A = (Q, \Sigma, \Delta, I, F)$ con $n$ estados. En primer lugar, el conjunto de estados iniciales $I$ consta únicamente de un estado, denominado $q_0$, y se emplea el alfabeto $\Sigma = \{0, 1\}$.

Para cada letra $a$ del alfabeto, se crea un grafo dirigido aleatorio (siguiendo el modelo de Erdős–Rényi) $D_a$ en $Q$ con $k$ aristas, que representan las transiciones $(p, a, q) \in \Delta$. Denominaremos densidad de transiciones para $a$ a la fracción $r = \frac{k}{|Q|}$. En este modelo, los autores consideran la misma densidad para ambos caracteres del alfabeto, por lo que nos referimos a $r$ como la densidad de transiciones del autómata $A$. Este constituye el primer parámetro del modelo.

El segundo parámetro corresponde a la cantidad de estados finales, determinada por una densidad de estados finales $f = \frac{m}{|Q|}$. Una vez establecida la cantidad de estados finales, se seleccionan aleatoriamente estados como finales, excepto el primero, que siempre es $q_0$. De esta manera, nos aseguramos de no generar un NFA cuyo lenguaje aceptado sea vacío.

Tabakov y Vardi examinan cómo la modificación de los parámetros del modelo influye en la probabilidad de que un NFA generado mediante dicho modelo acepte todas las palabras sobre el alfabeto $\{0, 1\}$. Ellos observan que esta probabilidad experimenta un cambio de forma continua entre cero y uno, ya sea que se modifique la densidad de estados finales $f$ o la densidad de transiciones $r$. Este cambio es suave, no existen transiciones de fase.

De la misma manera, Tabakov y Vardi ilustran que, cuando el parámetro $r$ adquiere un valor de $1{,}25$; el modelo genera autómatas con DFAs minimales que contienen un mayor número de estados, existe una pequeña transición de fase en este punto. Sin embargo, este valor máximo se ve afectado de forma marginal por cambios en el valor de $r$.

## 5.2. Algoritmos

En esta sección, describiremos los siete algoritmos con los que llevamos a cabo pruebas.

**Fuerza Bruta (FB):**

El algoritmo de fuerza bruta que utilizamos en los experimentos, consiste en generar on-the-fly cada una de las $2^k$ palabras de largo $k$ en el alfabeto $\{0, 1\}$ e ir probando si pertenecen al lenguaje aceptado por el autómata. Lo abreviaremos **FB** en adelante.

**Determinización sin minimización (DET):**

El segundo algoritmo que utlizamos, consiste en simplemente determinizar el autómata y utilizar el algoritmo para #DFA. Lo abreviaremos **DET** en adelante.

**Determinización con minimización (DET-M):**

Otro algoritmo a considerar, es además de determinizar el automata, minimizarlo mediante el algoritmo de Hopcroft, y luego utilizar el algoritmo para #DFA. Lo abreviaremos **DET-M** en adelante.

**FPRAS original (FPRAS):**

Implementamos el FPRAS exactamente como en el algoritmo 3 del capítulo anterior. Lo abreviaremos **FPRAS** en adelante.

**FPRAS con criterio de parada para el muestreo (FPRAS-S):**

Podemos observar que el tamaño del muestreo necesario en cada etapa del FPRAS, crece muy rápidamente. Para nodos en las primeras capas del DAG, el tamaño de la muestra es mayor a la cantidad de palabras de ese largo sobre

el alfabeto $\{0, 1\}$. Una optimización sencilla al FPRAS es en estos nodos, es simplemente detener el muestreo cuando ya se tienen tantas palabras como las palabras posibles de ese largo sobre el alfabeto $\{0, 1\}$.

Abreviaremos este algoritmo como **FPRAS-S**

**FPRAS con muestreo de tamaño lineal (FPRAS-L):**

El mayor cuello de botella en el FPRAS, es el tamaño de las muestras $S(q^\alpha)$ obtenidas de manera uniforme de $\mathcal{L}(q^\alpha)$. En el algoritmo original esta cantidad es $2\kappa^7$. Implementamos una versión donde el tamaño de estos conjuntos es solamente $2\kappa$, sumado a la optimización anterior de generar por fuerza bruta todas las palabras si el tamaño a muestrear es mayor a la cantidad de palabras del largo de la capa. Usaremos **FPRAS-L** para referirnos a este algoritmo.

**FPRAS con muestreo de tamaño lineal y sin independencia (FPRAS-NI):**

La prueba de correctitud del FPRAS provista por los autores en (Arenas et al., 2021) utiliza la desigualdad de Hoeffding. Para poder aplicar esta desigualdad, una de las condiciones necesarias es que la sucesión de variables aleatorias sea independiente e idénticamente distribuida (i.i.d).

Surge de manera natural la pregunta sobre si en este algoritmo se requiere realmente la independencia o se podría obtener otra prueba de correctitud bajo condiciones más laxas, sin requerir independencia entre las variables aleatorias asociadas a cada paso del muestreo. Una potencial herramienta para reemplazar la desigualdad de Hoeffding en la demostración es la desigualdad de Azuma (Azuma, 1967) que no requiere la hipótesis de independencia.

Para evaluar experimentalmente esta idea, implementamos una version del FPRAS con muestreo reducido a una cantidad lineal $2\kappa$ y donde la subrutina de muestreo recicla conjuntos de palabras, volviendo a empezar desde cero cada diez llamadas, lo que introduce dependencia en las variables aleatorias. Esta modificación se suma, al igual que en el caso anterior a generar en las primeras

Tabla 5.1. Selección de autómatas a utilizar

| Familia | NID | Densidad | Tabakov-Vardi |
|---|---|---|---|
| Cantidad de autómatas utilizados | 29 | 29 | 960 |
| Parámetros | $m$ | $k$ | $f, r, \|Q\|$ |
| Valores de los parámetros usados | $m \in \{2, \cdots, 30\}$ | $k \in \{1, \cdots, 29\}$ | $f = r = 1, 25$, $\|Q\| \in \{2, \cdots, 49\}$ |
| Autómatas por combinación de parámetros | 1 | 1 | 20 |

capas por fuerza bruta en vez de muestreo. Este algoritmo será abreviado como **FPRAS-NI**.

## 5.3. Experimentos

Los experimentos fueron realizados utilizando procesadores Intel Xeon Silver 4110 con frecuencia de reloj de 2,10 GHz y 29 GB de RAM. De cada familia se tomaron autómatas de la manera descrita en la tabla 5.1.

Después de haber definido los autómatas a emplear, podemos describir los experimentos efectuados. Consistieron en dos grupos, El primero de ellos consiste en calcular la cantidad de palabras aceptadas por un autómata con largo entre 1 y 50, mediante uno de los siete algoritmos. Se estableció el parámetro de error relativo $\delta$ en $0{,}1$ en el caso de los algoritmos aleatorizados. En cada ejecución, si un algoritmo no proporciona un resultado en un plazo de 300 segundos, se detiene la ejecución y se registra como una falla. En la tabla 5.2 podemos ver la cantidad de experimentos hechos. Este grupo fue el que empleamos para calcular el tiempo de ejecución de los algoritmos, y contiene experimentos con autómatas de las tres familias descritas anteriormente.

Asimismo, de los 960 autómatas de la familia Tabakov-Vardi, se seleccionaron 48 de ellos, uno por cada cantidad de estados comprendida entre 2 y 49. En cada uno de ellos realizamos un experimento adicional, que consiste en repetir cada intento con cada

Tabla 5.2. Cantidad de veces que cada algoritmo fue testeado en el grupo principal

| Familia | NID | Densidad | Tabakov-Vardi |
|---|---|---|---|
| Veces que se ejecutó cada algoritmo | 1450 | 1450 | 48000 |

algoritmo aleatorizado 30 veces. La motivación de lo anterior radica en la evaluación del error de los algoritmos aleatorizados. De esta manera, los algoritmos aleatorizados fueron ejecutados 72.000 cada uno, adicionalmente a los 48.000 ensayos anteriores. Este grupo, donde iteramos las ejecuciones de los algoritmos aleatorizados sobre el mismo autómata, es el que usamos para analizar el error de los algoritmos.

# 6. RESULTADOS

En las tres secciones de este capítulo mostraremos los resultados de nuestros experimentos. En la sección inicial, proporcionaremos información sobre el comportamiento de los errores de aproximación en los algoritmos aleatorizados. Posteriormente, en la segunda sección, compararemos el tiempo que demora cada algoritmo en resolver el problema. Finalmente, en la última sección, expondremos algunas ideas que surgen desde los resultados experimentales.

## 6.1. Errores de los algoritmos aleatorizados

Para analizar los errores de los algoritmos aleatorizados, debemos primero tener en cuenta un par de factores. El primero es que en las familias NID y Densidad, al momento de calcular el unroll, obtenemos autómatas no ambiguos. Es por esta razón que los cuatro algoritmos aleatorizados dan el resultado exacto, pues corresponde al caso descrito en la sección 3.1. Para analizar el error solamente utilizaremos la familia de Tabakov-Vardi.

Recordemos que hemos generado 960 autómatas de manera aleatoria, a partir del modelo de Tabakov-Vardi con parámetros $r = 1{,}25$ y $f = 0{,}25$. Generamos 20 autómatas por cada cantidad de estados entre 2 y 49. Una pregunta interesante es cuán común es para este modelo aleatorio producir autómatas deterministas o no ambiguos. En la tabla 6.1 podemos ver ese dato dentro de la muestra que generamos.

Tabla 6.1. Ambigüedad de autómatas generados desde el modelo de Tabakov-Vardi

| | Cantidad de autómatas | Porcentaje de autómatas |
|---|---|---|
| Deterministas | 20 | $2{,}08\%$ |
| No ambiguos | 27 | $2{,}81\%$ |
| Ambiguos | 933 | $97{,}18\%$ |

El segundo factor a considerar a la hora de analizar los errores de los algoritmos aleatorizados es que el algoritmo original FPRAS sin modificaciones da timeout en cerca del 92% de los experimentos realizados en la familia Tabakov-Vardi, por ende no reportamos los datos del error de este algoritmo.

Recordemos que el FPRAS original y sus modificaciones reciben como entrada un autómata, un valor $k$ dado en unario, correspondiente al largo de las palabras que deseamos contar, y un parámetro de error relativo $\delta \in (0{,}1)$. En el caso del FPRAS sin modificar, (Arenas et al., 2021) prueba una cota teórica para el error y esta garantía se extiende a FPRAS-S. En todos los experimentos que realizamos, asignamos al parámetro $\delta$ el valor $\delta = 0{,}1$.

Tabla 6.2. Errores del algoritmo FPRAS-S

| | |
|---|---|
| Casos con ejecución terminada | 7075 |
| Promedio del error | 0,013912 |
| Desviación estándar del error | 0,036582 |
| Máximo del error | 0,33 |
| Casos con error cero | 78 % |
| Casos fuera de la cota teórica | 284 |
| % fuera de la cota | 4,01% |
| Error medio en los casos fuera de la cota | 15,44% |

En las tablas siguientes, podemos observar los datos del error observados al contar la cantidad de palabras de largos entre 1 y 50, iterando 30 veces cada cálculo para 48 NFAs de tamaños entre 2 y 49 estados, como se describió en la sección 5.3. En este caso, con un valor del parámetro de error relativo $\delta = 0{,}1$, la garantía teórica se traduce en que existe una probabilidad mayor o igual a $\frac{3}{4}$ de que el algoritmo entregue un valor cuyo error relativo sea menor o igual a $0{,}1$.

Como se ve en la tabla 6.2, solamente el $4{,}01\%$ de los casos en el algoritmo FPRAS-S tienen un error relativo fuera de la cota de $0{,}1$; lo que es consistente con la teoría. En el caso del algoritmo FPRAS-L, no tenemos una cota teórica que entregue garantías sobre

Tabla 6.3. Errores del algoritmo FPRAS-L

| | |
|---|---|
| Casos con ejecución terminada | 8333 |
| Promedio del error | 0,000456 |
| Desviación estándar del error | 0,042937 |
| Máximo del error | 0,42 |
| Casos con error cero | 78 % |
| Casos fuera de la cota teórica | 188 |
| % fuera de la cota | 2,25% |
| Error medio en los casos fuera de la cota | 16,56% |

el error, sin embargo, como se ve en la tabla 6.3 obtenemos un excelente desempeño, con casi el $98\%$ de los casos con un error relativo menor al $10\%$.

Tabla 6.4. Errores del algoritmo FPRAS-NI

| | |
|---|---|
| Casos con ejecución terminada | 11458 |
| Promedio del error | 0,307775 |
| Desviación estándar | 0,291347 |
| Mediana | 0,285714 |
| Máximo | 0,897641 |
| Casos con error cero | 36 % |
| Casos fuera de la cota teórica | 6982 |
| % fuera de la cota | 61,43% |
| Error medio en los casos fuera de la cota | 49,91% |
| Fallos de muestreo | 93 |
| % fallos de muestreo | 0,81 |

Finalmente, en el caso del algoritmo FPRAS-NI, donde tampoco tenemos garantías teóricas, observamos a partir de la tabla 6.4 que, dentro del límite de tiempo, se calculan significativamente más casos. Sin embargo, el error promedio supera el $30\%$ y obtenemos fallos de muestreo, un evento que nunca ocurrió en los FPRAS que muestrean de manera independiente. La diferencia entre FPRAS-NI y los otros algoritmos radica en la forma de realizar el muestreo.

Recordemos cómo funciona el procedimiento de muestreo descrito en la sección 4.4 cuyo pseudocódigo corresponde al algoritmo 2 de la sección. Esta subrutina recibe un subconjunto de estados del unroll del autómata, que corresponde al DAG hasta una capa $\beta$. La salida de esta función es una palabra $w$, obtenida bajo la distribución uniforme, del lenguaje inducido por el subconjunto de estados, o nos indica que falló en obtener una palabra bajo esta distribución.

Esta función es utilizada para obtener los sketches $S(p^\alpha)$ que corresponden a muestras de los lenguajes $\mathcal{L}(p^\alpha)$ correspondientes a estados $p$ en la capa $\alpha$ del NFA desenrollado. Como señala la proposición 4.4, este procedimiento de muestreo, tiene una probabilidad de error que se encuentra acotada superiormente por $1 - e^{-9} \approx 99,98\%$. Por esta razón, cada llamada a esta rutina se repite $\mathcal{O}(\log(\kappa))$ veces.

En las demás versiones del FPRAS, al construir cada sketch $S(p^\alpha)$ estamos volviendo a muestrear palabras de largo $\alpha - 1$ y menores, es decir repitiendo una computación que fue realizada a la hora de construir sketches anteriores. Nuestra modificación en este caso consiste en reutilizar los sketches anteriores, volviendo a muestrear de manera independiente en uno de cada 10 sketches.

El resultado de este cambio se encuentra en la tabla 6.4. Si bien esta modificación permite aumentar de forma relativamente significativa el total de casos que se resuelven antes del tiempo límite, también conlleva problemas. Aparecen fallas muestrales y el error promedio de nuestras estimaciones aumenta significativamente. La mediana del error es del $28\%$, pero puede llegar hasta casi el $90\%$.

## 6.2. Tiempo de ejecución de los algoritmos

En esta sección realizaremos un análisis del tiempo de ejecución de los diferentes algoritmos en cada una de las tres familias de autómatas que utilizamos en los experimentos.

### 6.2.1. Familia Tabakov-Vardi

En las tablas 6.5 y 6.6 podemos ver, dentro del conjunto de los 960 autómatas de esta familia, para cada cantidad de estados, el mayor largo de palabra (parámetro $k$) para el cual cada algoritmo aleatorizado concluyó su ejecución dentro del tiempo de 300 segundos.

Tabla 6.5. Mayor largo de palabras contado dentro del límite de tiempo para cantidad de estados entre 2 y 25

| Estados | FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|---|
| 2 | 10 | 10 | 19 | 19 |
| 3 | 5 | 7 | 9 | 9 |
| 4 | 10 | 10 | 19 | 19 |
| 5 | 6 | 7 | 9 | 9 |
| 6 | 5 | 8 | 10 | 10 |
| 7 | 4 | 6 | 9 | 9 |
| 8 | 6 | 9 | 9 | 50 |
| 9 | 5 | 7 | 9 | 9 |
| 10 | 6 | 8 | 9 | 9 |
| 11 | 5 | 7 | 9 | 9 |
| 12 | 6 | 8 | 10 | 50 |
| 13 | 4 | 6 | 9 | 9 |
| 14 | 5 | 7 | 9 | 50 |
| 15 | 14 | 11 | 9 | 50 |
| 16 | 5 | 7 | 9 | 50 |
| 17 | 6 | 8 | 9 | 9 |
| 18 | 5 | 6 | 9 | 50 |
| 19 | 5 | 7 | 9 | 9 |
| 20 | 6 | 9 | 9 | 9 |
| 21 | 4 | 6 | 9 | 9 |
| 22 | 5 | 7 | 9 | 9 |
| 23 | 5 | 7 | 9 | 50 |
| 24 | 5 | 7 | 9 | 9 |
| 25 | 5 | 7 | 9 | 50 |

Para ver como escalan los resultados, analizaremos más detalladamente los casos de autómatas con 5, 30 y 49 estados.

Tabla 6.6. Mayor largo de palabras contado dentro del límite de tiempo para cantidad de estados entre 26 y 49

| Estados | FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|---|
| 26 | 5 | 7 | 9 | 9 |
| 27 | 4 | 7 | 9 | 50 |
| 28 | 5 | 8 | 9 | 9 |
| 29 | 4 | 6 | 9 | 9 |
| 30 | 4 | 7 | 9 | 9 |
| 31 | 4 | 6 | 9 | 9 |
| 32 | 5 | 8 | 9 | 9 |
| 33 | 3 | 6 | 9 | 50 |
| 34 | 3 | 8 | 9 | 50 |
| 35 | 4 | 8 | 9 | 9 |
| 36 | 4 | 6 | 9 | 50 |
| 37 | 4 | 7 | 9 | 9 |
| 38 | 3 | 7 | 9 | 9 |
| 39 | 4 | 7 | 9 | 50 |
| 40 | 4 | 7 | 9 | 9 |
| 41 | 4 | 6 | 9 | 9 |
| 42 | 3 | 6 | 9 | 50 |
| 43 | 3 | 7 | 9 | 9 |
| 44 | 4 | 8 | 9 | 50 |
| 45 | 4 | 6 | 9 | 9 |
| 46 | 3 | 6 | 9 | 9 |
| 47 | 4 | 8 | 9 | 50 |
| 48 | 4 | 7 | 9 | 50 |
| 49 | 4 | 6 | 9 | 9 |
| 48 | 4 | 7 | 9 | 50 |
| 49 | 4 | 6 | 9 | 9 |

**Caso de 5 estados**

En la tabla 6.7 podemos ver el mayor valor para el cual cada algoritmo logró contar dentro de los 300 segundos las palabras que tienen como largo ese valor. Este dato corresponde a ejecutar cada algoritmo en 20 autómatas distintos de 5 estados, cada uno de la familia de Tabakov-Vardi.

Tabla 6.7. Mayor valor para el cuál cada algoritmo logró contar las palabras de ese largo, antes del timeout, en autómatas de 5 estados.

| FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|
| 6 | 7 | 9 | 9 |

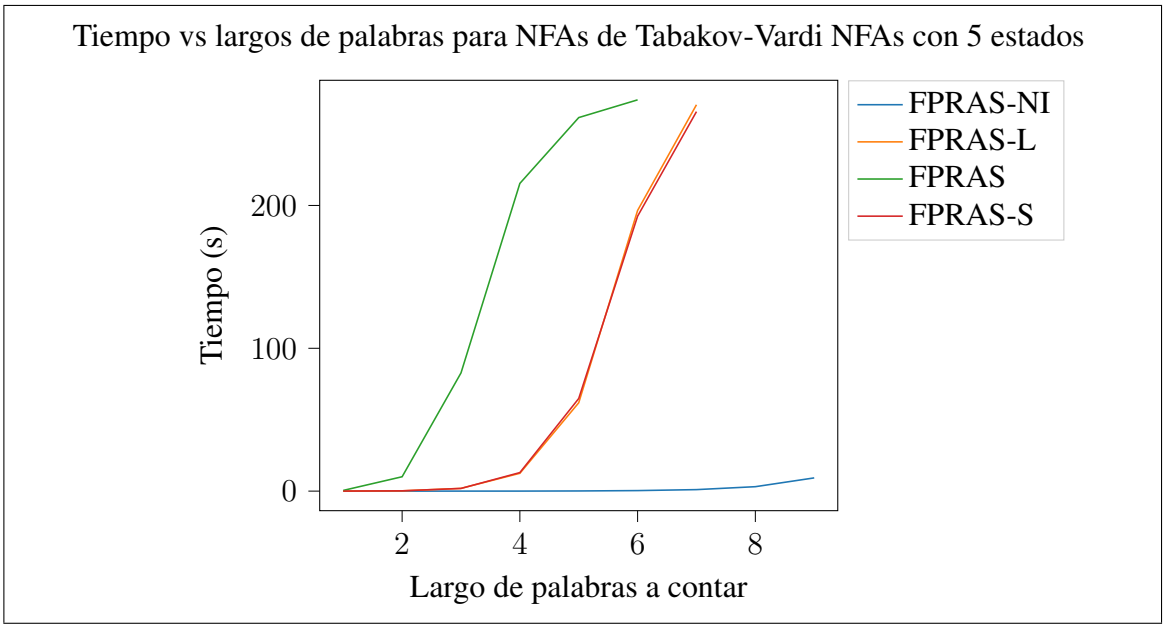


Figura 6.1. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 5 estados.

**Caso de 30 estados**

En la tabla 6.8 podemos observar el mayor valor con ejecución terminada para cada algoritmo en un conjunto de 20 autómatas distintos de 30 estados, provenientes de la familia de Tabakov-Vardi.

Tabla 6.8. Mayor valor para el cuál cada algoritmo logró contar las palabras de ese largo, antes del timeout, en autómatas de 30 estados.

| FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|
| 4 | 7 | 9 | 9 |

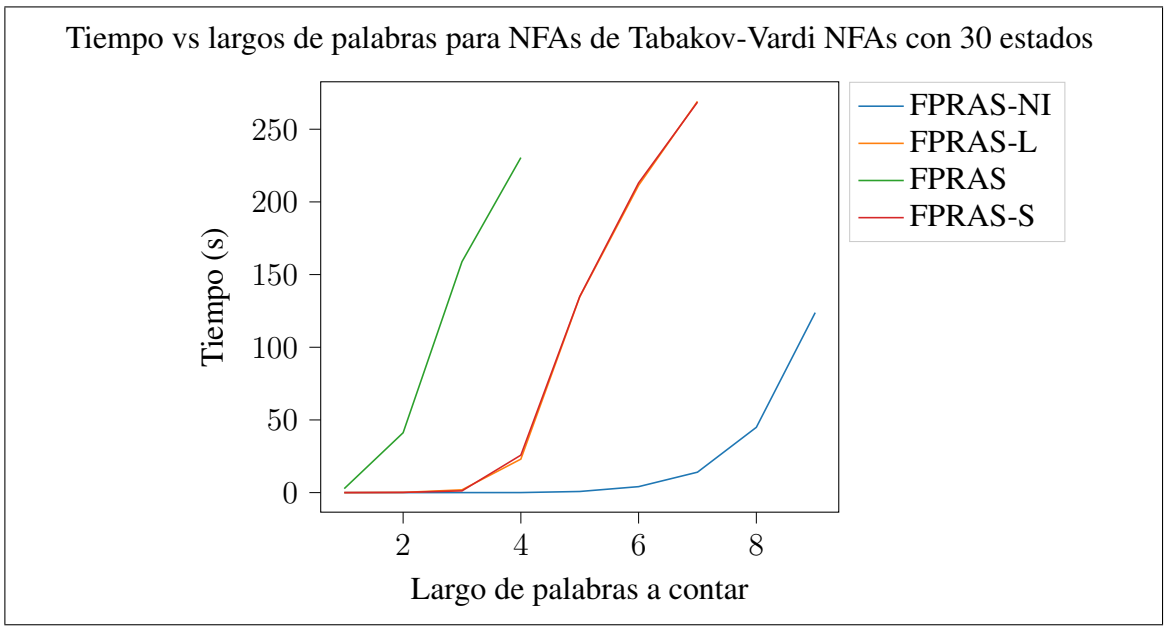


Figura 6.2. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 30 estados.

**Caso de 49 estados**

En la tabla 6.9 vemos el mayor valor que no dio timeout para cada algoritmo logró contar dentro de los 300 segundos. Este dato corresponde a ejecutar cada algoritmo en 20 autómatas distintos de 5 estados, cada uno de la familia de Tabakov-Vardi.

Tabla 6.9. Mayor valor para el cuál cada algoritmo logró contar las palabras de ese largo, antes del timeout, en autómatas de 49 estados.

| FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|
| 4 | 6 | 9 | 9 |

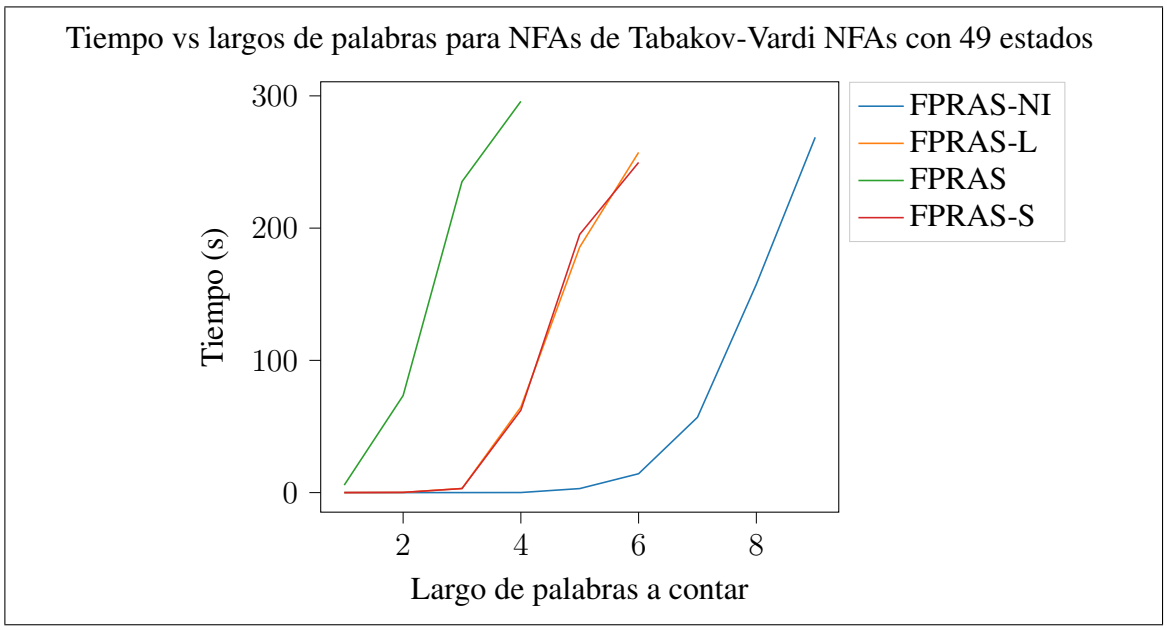


Figura 6.3. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 49 estados.

### 6.2.2. Familia NID

En esta subsección analizaremos los tiempos de ejecución de los diferentes algoritmos en la familia $\text{NID}_m$, que fue descrita anteriormente en la subsección 5.1.1. Recordemos que esta familia fue elegida debido a que con la característica de ser difícil de determinizar y desambiguar. Podemos calcular fácilmente la cantidad de palabras de largo $k$ en el lenguaje $\text{NID}_m$ usando la formula siguiente, que ya habíamos visto como la ecuación 5.1.

$$\text{NID}_m \cap \{0,1\}^k = \begin{cases} 2^{k-m} & \text{si } k \geq m+1 \\ 0 & \text{si } k < m+1 \end{cases}$$

Nos enfocaremos solamente en las combinaciones de parámetros $m$ y $k$ tal que el lenguaje $\text{NID}_m \cap \{0,1\}^k \neq \varnothing$, esto sucede cuando $k \geq m+1$. La razón de esta restricción es que nuestros algoritmos detectan si un lenguaje no tiene palabras de largo $k$ en el preprocesamiento, sin recurrir a la rutina principal.

Podemos ver los valores máximos del parámetro $k$, correspondiente al largo de las palabras a contar, para los cuales cada algoritmo finalizó dentro de los 300 segundos, en la tabla 6.10. Es destacable que los algoritmos deterministas dieron timeout en todos los casos con 38 o más estados.

Observaremos en el gráfico 6.4 el caso correspondiente a un valor de $4$ para el parámetro $m$, i.e un autómata de 10 estados. Este es el caso más grande en esta familia donde todos los algoritmos aleatorizados finalizaron su ejecución dentro del limite de tiempo. Una observación relevante en esta familia, es que con valores de $m$ mayores a 18, es decir con 39 estados o más, incluso los algoritmos deterministas dieron timeout.

Tabla 6.10. Mayores valores para los cuales cada algoritmo logró contar las palabras de ese largo, una cruz indica timeout en todos los casos. En los casos con m mayor a 18, todos los algoritmos fallaron para todos los largos

| m | Cantidad de Estados | DET | FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|---|---|---|
| 2 | 6 | 11 | 7 | 9 | 9 | 10 |
| 3 | 8 | 11 | 6 | 9 | 9 | 10 |
| 4 | 10 | 10 | 5 | 9 | 9 | 10 |
| 5 | 12 | 10 | × | 9 | 9 | 10 |
| 6 | 14 | 11 | × | 9 | 9 | 10 |
| 7 | 16 | 11 | × | 9 | 9 | 10 |
| 8 | 18 | 11 | × | 9 | 9 | 10 |
| 9 | 20 | 11 | × | × | × | 10 |
| 10 | 22 | 11 | × | × | × | × |
| 11 | 24 | 12 | × | × | × | × |
| 12 | 26 | 12 | × | × | × | × |
| 13 | 28 | 14 | × | × | × | × |
| 14 | 30 | 15 | × | × | × | × |
| 15 | 32 | 16 | × | × | × | × |
| 16 | 34 | 17 | × | × | × | × |
| 17 | 36 | 18 | × | × | × | × |
| 18 | 38 | 19 | × | × | × | × |

### 6.2.3. Familia Densidad

En esta subsección analizaremos los tiempos de ejecución de los diferentes algoritmos en la familia de autómatas de densidad $\frac{1}{2^k}$, cuya descripción se encuentra en la subsección 5.1.2. La razón para elegir esta familia es la pregunta sobre la influencia de la densidad del lenguaje en el rendimiento de la subrutina de muestreo. Los autómatas asociados a esta familia son deterministas, están parametrizados por un valor $k$ y aceptan $2^{k-1}$ palabras de largo $k$, y cero palabras de largo distinto a $k$. Realizamos pruebas con valores de $k$ en el conjunto $\{2, \ldots, 15\}$. Los mayores valores del parámetro para el cual logramos contar de manera exitosa se encuentran en la tabla 6.11

Una observación relevante es que el algoritmo FPRAS-NI retornó errores de muestreo en todos los casos con $k$ mayor a 9.

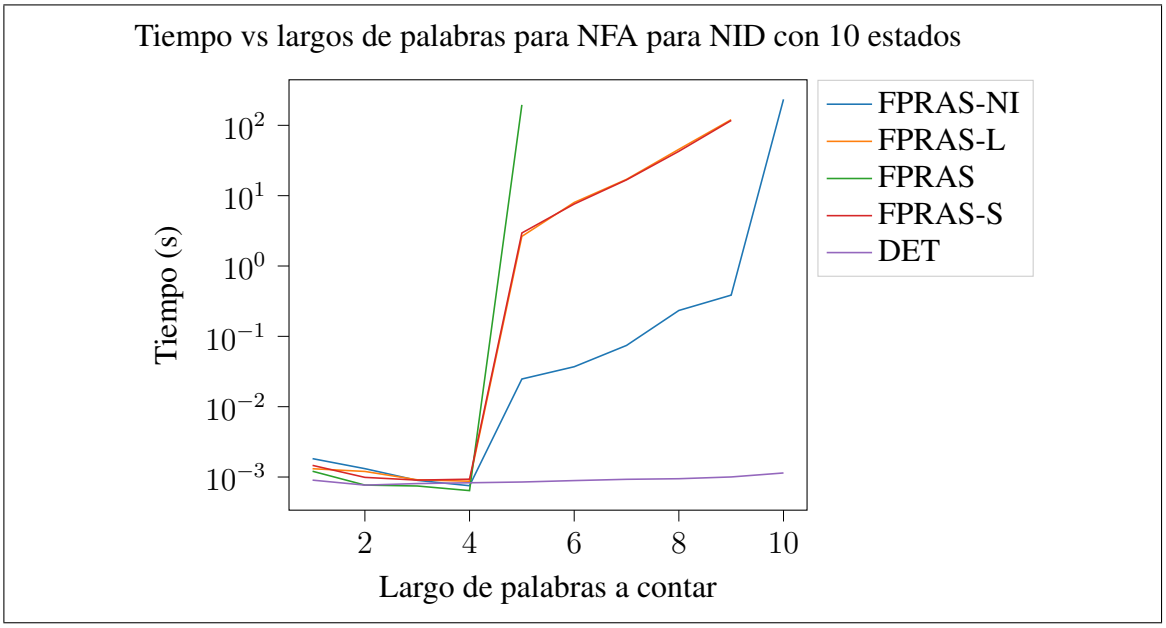


Figura 6.4. Tiempos de ejecución para el NFA correspondiente a NID con $m = 4$

Tabla 6.11. Mayor valor del parámetro $k$ para el cual cada algoritmo logró contar las palabras de ese largo. Los algoritmos deterministas lo lograron para todos los valores

| FPRAS | FPRAS-S | FPRAS-L | FPRAS-NI |
|---|---|---|---|
| 8 | 11 | 11 | 9 |

## 6.3. Análisis de los resultados

Para concluir este capítulo, una vez vistos los resultados experimentales en estas tres familias en las cuales hemos realizado experimentos, expondremos lo que hemos aprendido desde esta perspectiva de evaluación de algoritmos. Hemos visto constantemente que el rendimiento de los FPRAS ha sido malo en términos de tiempo. Esto coincide con él único análisis experimental existente de un algoritmo de este tipo, que fue realizado por (Newman & Vardi, 2020) para el problema de calcular permanentes de matrices.

Identificamos tres consideraciones cruciales:

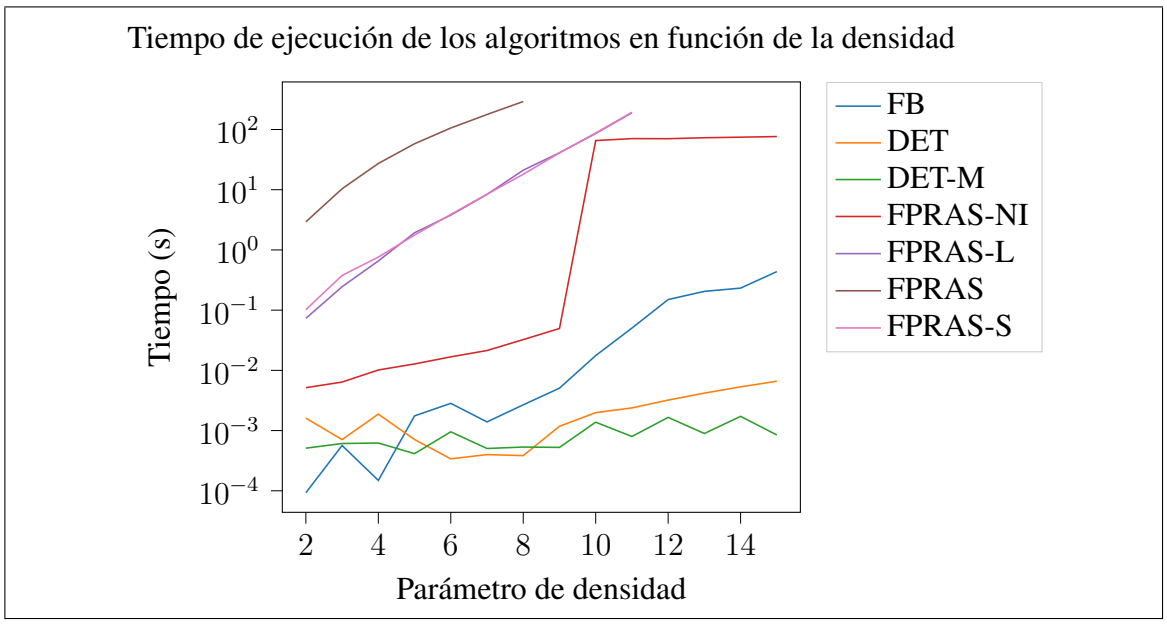


Figura 6.5. Tiempos de ejecución en las pruebas de densidad

### 6.3.1. Los FPRAS no son factibles en la práctica

Los FPRAS constituyen un avance teóricamente significativo en el ámbito de la complejidad computacional. No obstante, en la práctica, su eficacia para resolver problemas es limitada. El algoritmo en su versión original no resolvió instancias de tamaño pequeño en un tiempo razonable de 300 segundos, y las versiones con parámetros relajados solo tuvieron éxito con instancias de tamaño pequeño a mediano.

### 6.3.2. Los FPRAS tienen espacio para mejorar y relajar algunos parámetros de correctitud

La demostración de correctitud del FPRAS realizada por sus autores en (Arenas et al., 2021) requiere de supuestos bastante fuertes para poder utilizar herramientas de concentración de la medida. Estos supuestos son dos: en primer lugar, que la subrutina de muestreo debe obtener una muestra con una cantidad de elementos de orden $\mathcal{O}(n^7)$ en cada iteración, y en segundo lugar, que en cada iteración $k$ se parte el muestreo de palabras de

largo $k$ de forma independiente, sin poder reusar el trabajo realizado durante el muestreo de palabras de largo $k-1$. La evidencia experimental indica que el primer supuesto parece ser mucho menos influyente que el segundo, por lo cual un desafió interesante es lograr rehacer la prueba de correctitud con un muestreo más acotado.

### 6.3.3. Necesidad de un marco robusto para evaluar el rendimiento de algoritmos sobre autómatas

La complejidad asintótica puede no reflejar adecuadamente el rendimiento práctico de un algoritmo. Esto es evidente al comparar los algoritmos de conteo basados en determinización, que, aunque exponenciales en teoría, superaron a los FPRAS polinomiales en nuestras pruebas.

Es raro realizar análisis experimentales para algoritmos de este tipo, y no podemos asegurar que las familias de autómatas que elegimos utilizar sean representativas de los problemas prácticos. En el campo de la satisfacibilidad booleana (SAT), existen baterías de pruebas para evaluar el rendimiento que han debidamente estudiadas, y cuyo grado de correlación con el rendimiento en la práctica está bien fundamentado. Este enfoque ha estado ausente en problemas relacionados con autómatas.

# 7. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo implementamos por primera vez el algoritmo de aproximación aleatorizado de (Arenas et al., 2021) y realizamos un primer análisis experimental de su rendimiento utilizando tres familias distintas de autómatas. No es común evaluar el rendimiento de algoritmos como este usando nuestro enfoque, sino que, se suele solamente realizar un análisis teórico en términos de comportamiento asintótico y notación $\mathcal{O}$.

Este enfoque metodológico parece ser bastante singular y permite evaluar la factibilidad de utilizar estos algoritmos desde un punto de vista más realista. Una de las familias de autómatas que utilizamos para evaluar corresponde al modelo de Tabakov-Vardi de NFAs aleatorios. Esta manera de generar autómatas de manera aleatoria es bastante natural, sin embargo, sorprendentemente, en la literatura no existen otros usos de este modelo hasta ahora, distintos a los que realizaron los autores de este. Tampoco existen análisis teóricos de los NFAs generados por este modelo, algo que contrasta bastante con el mundo de los grafos aleatorios, en particular con el modelo de grafos aleatorios de Erdős-Rényi (Erdős & Rényi, 1959), sobre el cual se inspira el modelo de Tabakov-Vardi. Esto propone de inmediato un área interesante para trabajos futuros.

Dentro de la idea de generar modelos aleatorios de entradas con la cual evaluar el rendimiento de algoritmos, surge otra línea factible de trabajo futuro, correspondiente a proponer otros métodos para generar autómatas de manera aleatoria, una primera idea podría ser modificar Tabakov-Vardi cambiando el modelo subyacente de grafos aleatorios por otro ya estudiado como (Barabási & Albert, 1999) o (Watts & Strogatz, 1998). Resulta interesante preguntarse cual de estos modelos genera entradas más similares a las reales en diversos contextos.

Nuestros resultados entregan bastante evidencia experimental de la ineficiencia del FPRAS de (Arenas et al., 2021) para varias familias de automátas. Esta evidencia se extiende a algunas relajaciones que hemos estudiado, considerando un tamaño menor de muestreo o no requiriendo que cada intento en el muestreo sea independiente. Ejecutamos el FPRAS original y sus variantes cerca de 123.000 veces y con nuestro timeout de 300 segundos por llamada, no encontramos ningún caso en el que alguno de los algoritmos aleatorizados haya podido resolver #NFA más rápido que los algoritmos que determinizan, los que tienen una complejidad asintótica de tiempo exponencial en comparación a una polinomial de los algoritmos aleatorizados.

Vale recordar el concepto de algoritmos galácticos planteado en (Lipton & Regan, 2013), como algoritmos cuyo comportamiento asintótico es mejor a lo mejores conocidos en el momento de su publicación, pero no son más eficientes que los que ya existen con tamaños de entradas existentes en la realidad. (Lipton & Regan, 2013) señalan que este tipo de algoritmos suelen contener nuevas técnicas que pueden ser utilizadas para crear otros algoritmos que si tengan aplicaciones prácticas. En esta linea, está abierto el desafío de demostrar que se puede reducir el tamaño del muestreo en cada paso del FPRAS para #NFA. Otro cuello de botella presente en el algoritmo, consiste en la subrutina de muestreo, la cuál tiene una probabilidad de error constante (por ende en el análisis asintótico requiere ser ejecutada solamente una cantidad polinomial de veces), pero es un valor muy alto, mayor al 99,99%. Un avance, aunque sea menor, en mejorar el rendimiento de esta rutina de muestreo, podría tener un gran impacto en el tiempo de ejecución de estos algoritmos.

# REFERENCIAS


Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2017, sep). Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, *50*(5). Retrieved from https://doi.org/10.1145/3104031 doi: 10.1145/3104031

Arenas, M., Croquevielle, L. A., Jayaram, R., & Riveros, C. (2021). #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, *68*(6), Art. 48, 40. Retrieved from https://doi.org/10.1145/3477045 doi: 10.1145/3477045

Azuma, K. (1967). Weighted sums of certain dependent random variables. *Tohoku Math. J. (2)*, *19*, 357–367. Retrieved from https://doi.org/10.2748/tmj/1178243286 doi: 10.2748/tmj/1178243286

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *science*, *286*(5439), 509–512.

Bezáková, I., Štefankovič, D., Vazirani, V. V., & Vigoda, E. (2008). Accelerating simulated annealing for the permanent and combinatorial counting problems. *SIAM J. Comput.*, *37*(5), 1429–1454. Retrieved from https://doi.org/10.1137/050644033 doi: 10.1137/050644033

Crawford, J. M., & Auton, L. D. (1996, March). Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, *81*(1), 31–57. Retrieved 2023-04-03, from https://www.sciencedirect.com/science/article/pii/0004370295000461 doi: 10.1016/0004-3702(95)00046-1

Doyen, L., & Raskin, J.-F. (2009, February). *Antichains for the Automata-Based Approach*

*to Model-Checking* (Tech. Rep.). Retrieved 2023-03-19, from https://ui.adsabs.harvard.edu/abs/2009arXiv0902.3958D (Publication Title: arXiv e-prints ADS Bibcode: 2009arXiv0902.3958D Type: article) doi: 10.48550/arXiv.0902.3958

Erdős, P., & Rényi, A. (1959). On random graphs i. *Publ. Math. Debrecen*, *6*(290-297), 18.

Gent, I. P., & Walsh, T. (1996, March). The satisfiability constraint gap. *Artificial Intelligence*, *81*(1), 59–80. Retrieved 2023-04-03, from https://www.sciencedirect.com/science/article/pii/000437029500047X doi: 10.1016/0004-3702(95)00047-X

Harvey, D., & van der Hoeven, J. (2021). Integer multiplication in time $O(n \log n)$. *Ann. of Math. (2)*, *193*(2), 563–617. Retrieved from https://doi.org/10.4007/annals.2021.193.2.4 doi: 10.4007/annals.2021.193.2.4

Holík, L., Meyer, R., & Muskalla, S. (2016). Summaries for Context-Free Games. In A. Lal, S. Akshay, S. Saurabh, & S. Sen (Eds.), *36th iarcs annual conference on foundations of software technology and theoretical computer science (fsttcs 2016)* (Vol. 65, pp. 41:1–41:16). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2016/6876 doi: 10.4230/LIPIcs.FSTTCS.2016.41

Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H., & Schnitger, G. (2002, January). Communication Complexity Method for Measuring Nondeterminism in Finite Automata. *Information and Computation*, *172*(2), 202–217. Retrieved 2021-09-24, from https://www.sciencedirect.com/science/article/pii/S089054010193069X doi: 10.1006/inco.2001.3069

Jerrum, M., Sinclair, A., & Vigoda, E. (2004). A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, *51*(4), 671–697. Retrieved from https://doi.org/10.1145/1008731.1008738 doi: 10.1145/1008731.1008738

Jerrum, M. R., Valiant, L. G., & Vazirani, V. V. (1986). Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, *43*, 169-188. Retrieved from https://www.sciencedirect.com/science/article/pii/030439758690174X doi: https://doi.org/10.1016/0304-3975(86)90174-X

Kannan, S., Sweedyk, Z., & Mahaney, S. (1995, January). Counting and random generation of strings in regular languages. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms* (pp. 551–557). USA: Society for Industrial and Applied Mathematics.

Karp, R. M., Luby, M., & Madras, N. (1989). Monte Carlo approximation algorithms for enumeration problems. *J. Algorithms*, *10*(3), 429–448. Retrieved from https://doi.org/10.1016/0196-6774(89)90038-2 doi: 10.1016/0196-6774(89)90038-2

Lipton, R. J., & Regan, K. W. (2013). David johnson: Galactic algorithms. In *People, problems, and proofs: Essays from gödel's lost letter: 2010* (pp. 109–112). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-642-41422-0_20 doi: 10.1007/978-3-642-41422-0_20

Newman, J. E., & Vardi, M. Y. (2020). FPRAS approximation of the matrix permanent in practice. *CoRR*, *abs/2012.03367*. Retrieved from https://arxiv.org/abs/2012.03367

Searls, D. B. (1993). The computational linguistics of biological sequences. In *Artificial intelligence and molecular biology* (p. 47–120). USA: American Association for Artificial

Intelligence.

Segoufin, L. (2013). Enumerating with constant delay the answers to a query. In W. Tan, G. Guerrini, B. Catania, & A. Gounaris (Eds.), *Joint 2013 EDBT/ICDT conferences, ICDT '13 proceedings, genoa, italy, march 18-22, 2013* (pp. 10–20). ACM. Retrieved from https://doi.org/10.1145/2448496.2448498 doi: 10.1145/2448496.2448498

Selman, B., Mitchell, D. G., & Levesque, H. J. (1996). Generating hard satisfiability problems. In (Vol. 81, pp. 17–29). Retrieved from https://doi.org/10.1016/0004-3702(95)00045-3 (Frontiers in problem solving: phase transitions and complexity) doi: 10.1016/0004-3702(95)00045-3

Tabakov, D., & Vardi, M. Y. (2005). Experimental evaluation of classical automata constructions. In *Proceedings of the 12th international conference on logic for programming, artificial intelligence, and reasoning* (p. 396–411). Berlin, Heidelberg: Springer-Verlag. Retrieved from https://doi.org/10.1007/11591191_28 doi: 10.1007/11591191_28

Tabakov, D., & Vardi, M. Y. (2007). Model checking buechi specifications. In R. Loos, S. Z. Fazekas, & C. Martín-Vide (Eds.), *LATA 2007. proceedings of the 1st international conference on language and automata theory and applications* (Vol. Report 35/07, pp. 565–576). Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona.

Valiant, L. G. (1979a). The complexity of computing the permanent. *Theoret. Comput. Sci.*, *8*(2), 189–201. Retrieved from https://doi.org/10.1016/0304-3975(79)90044-6 doi: 10.1016/0304-3975(79)90044-6

Valiant, L. G. (1979b). The complexity of enumeration and reliability problems. *SIAM*

*J. Comput.*, *8*(3), 410–421. Retrieved from https://doi.org/10.1137/0208032 doi: 10.1137/0208032

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, *393*(6684), 440–442.

Álvarez, C., & Jenner, B. (1993). A very hard log-space counting class. *Theoretical Computer Science*, *107*(1), 3-30. Retrieved from https://www.sciencedirect.com/science/article/pii/030439759390252O doi: https://doi.org/10.1016/0304-3975(93)90252-O

# ANEXO

# A. GRÁFICOS DEL TIEMPO DE EJECUCIÓN

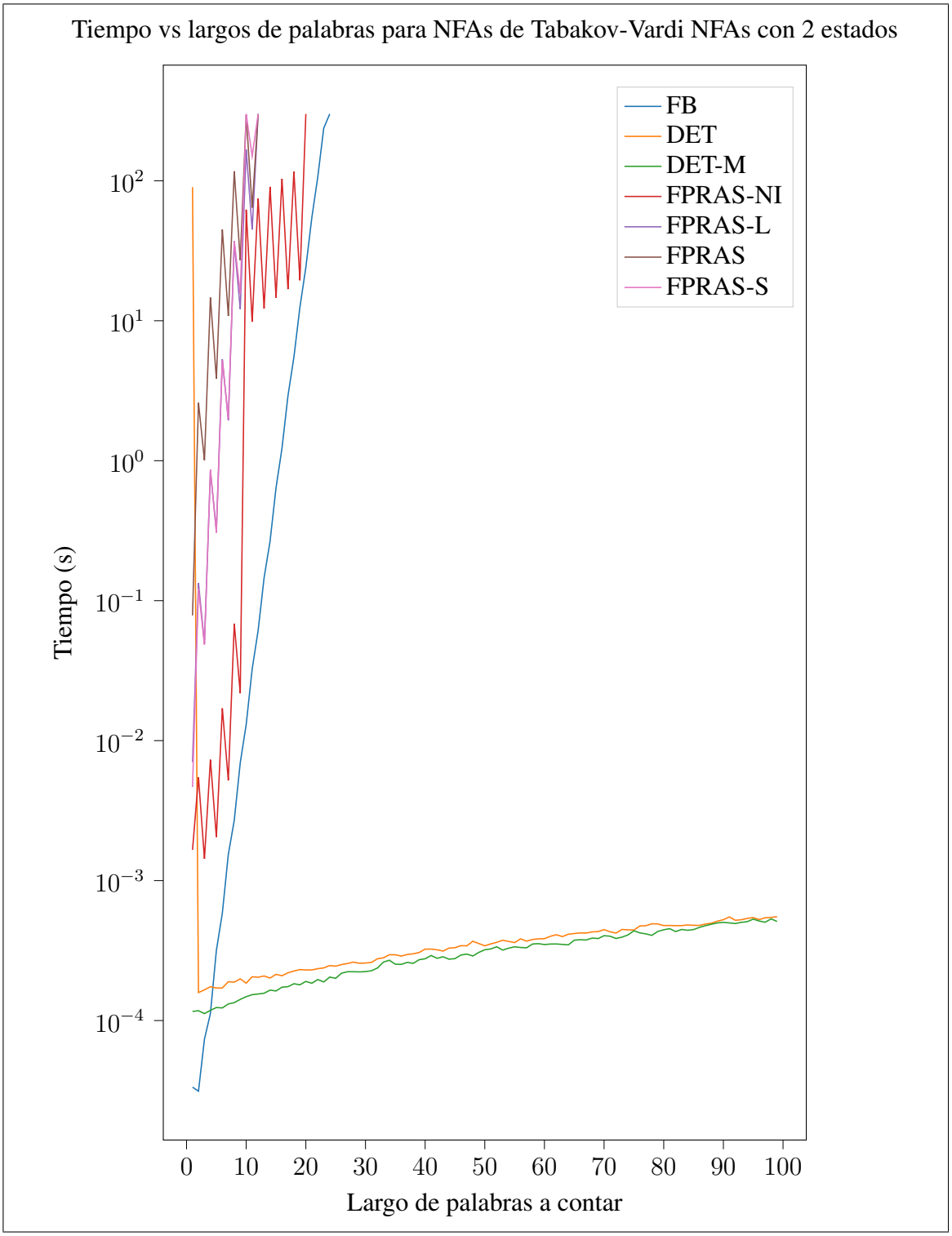


Figura A.1. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 2 estados.

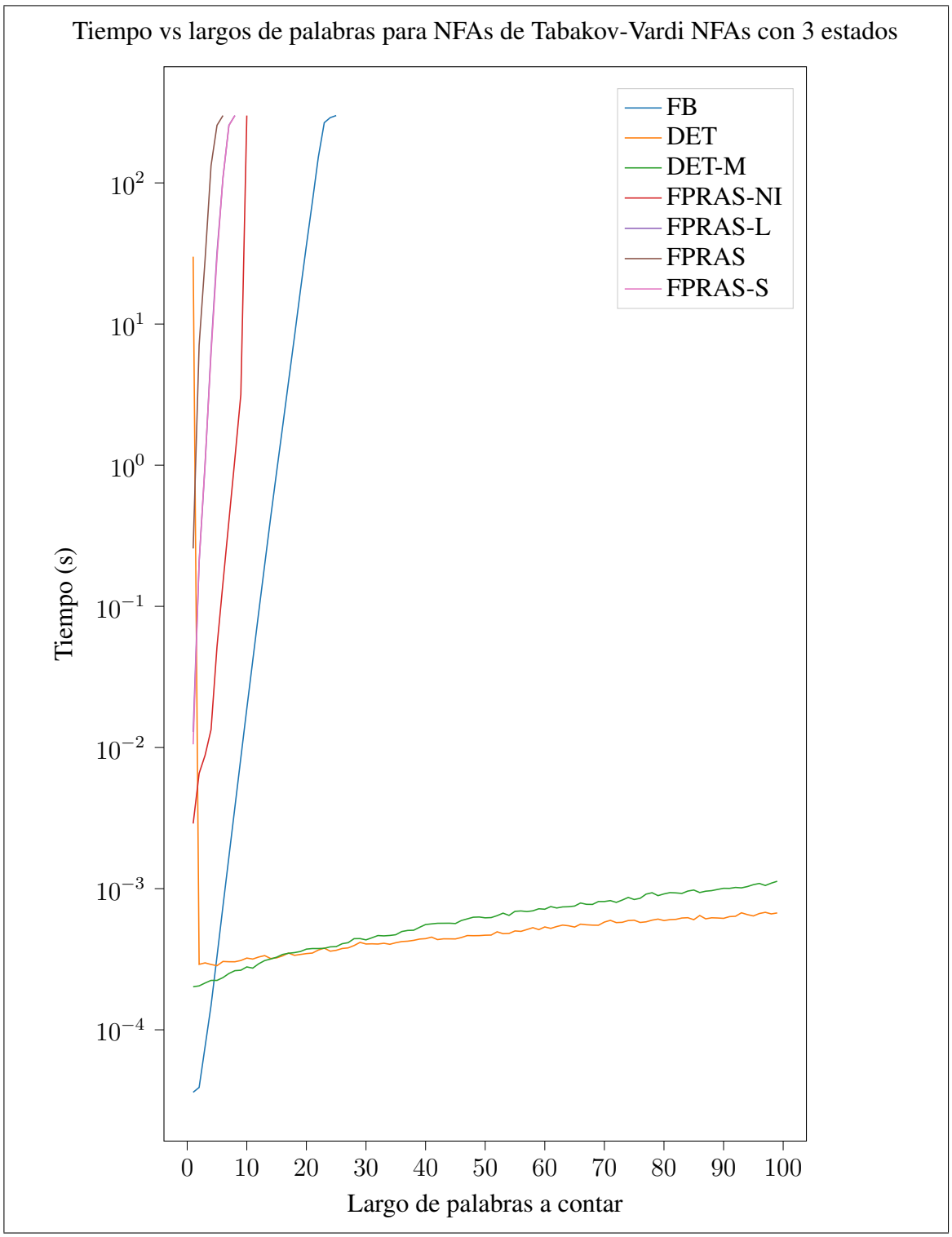


Figura A.2. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 3 estados.

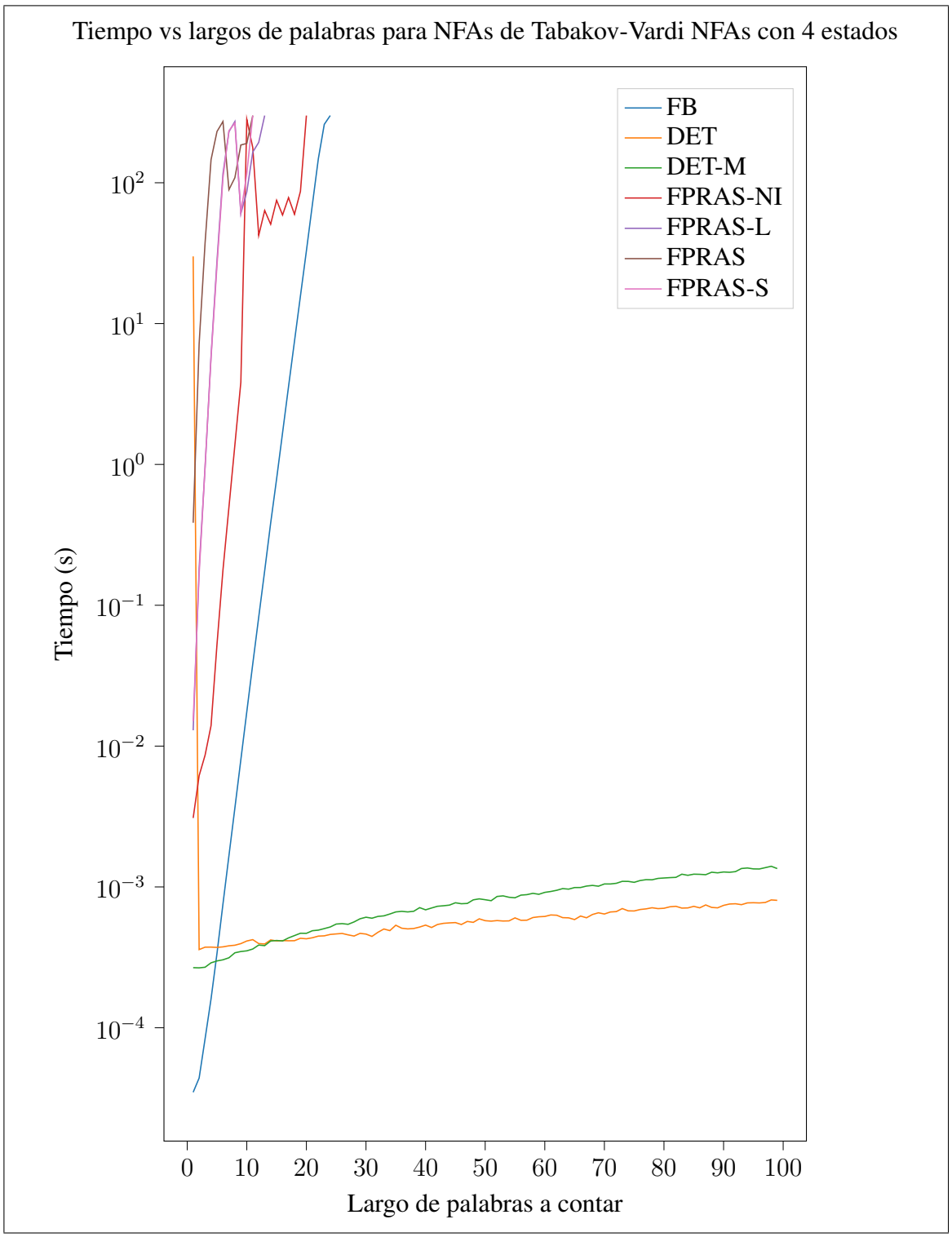


Figura A.3. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 4 estados.

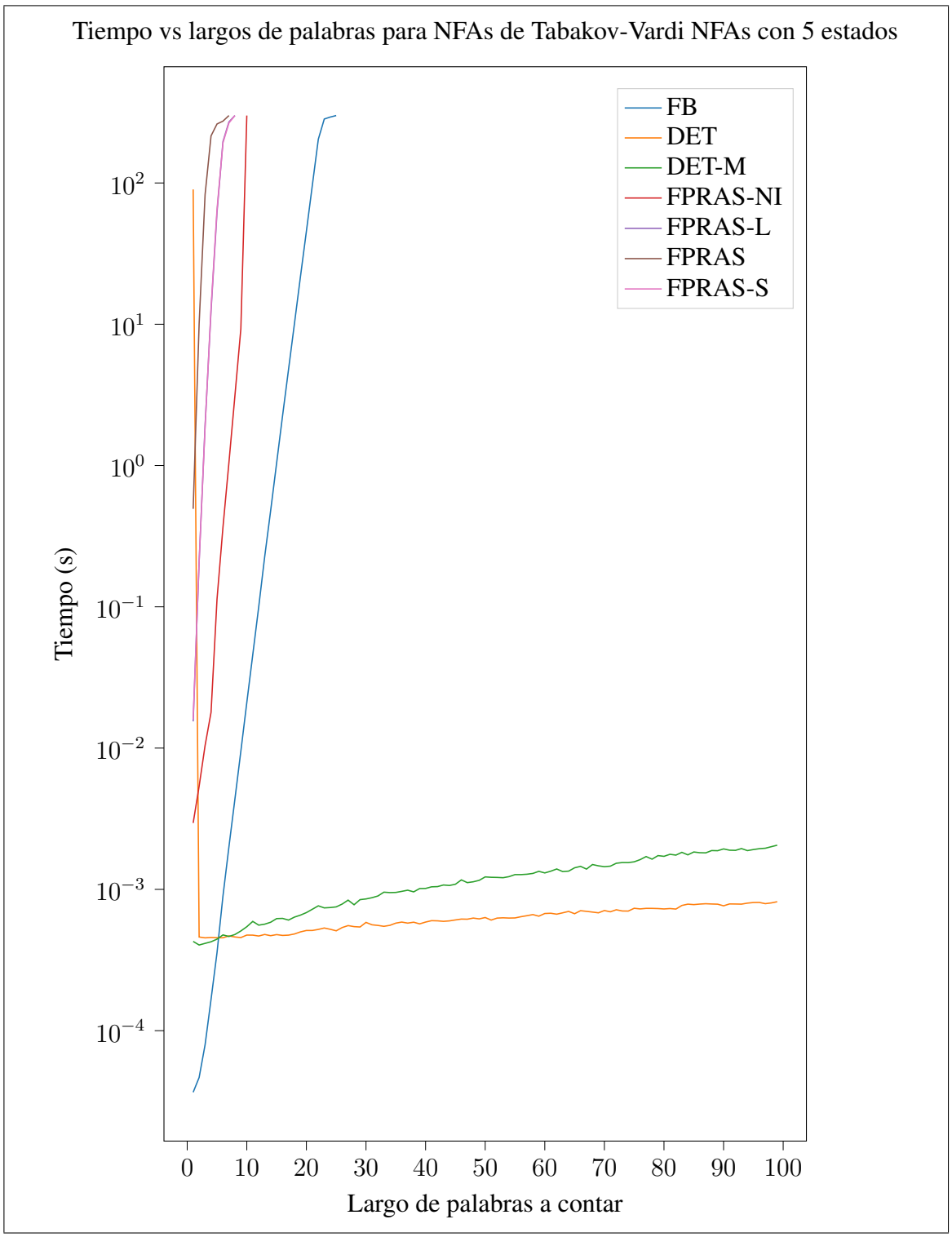


Figura A.4. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 5 estados.

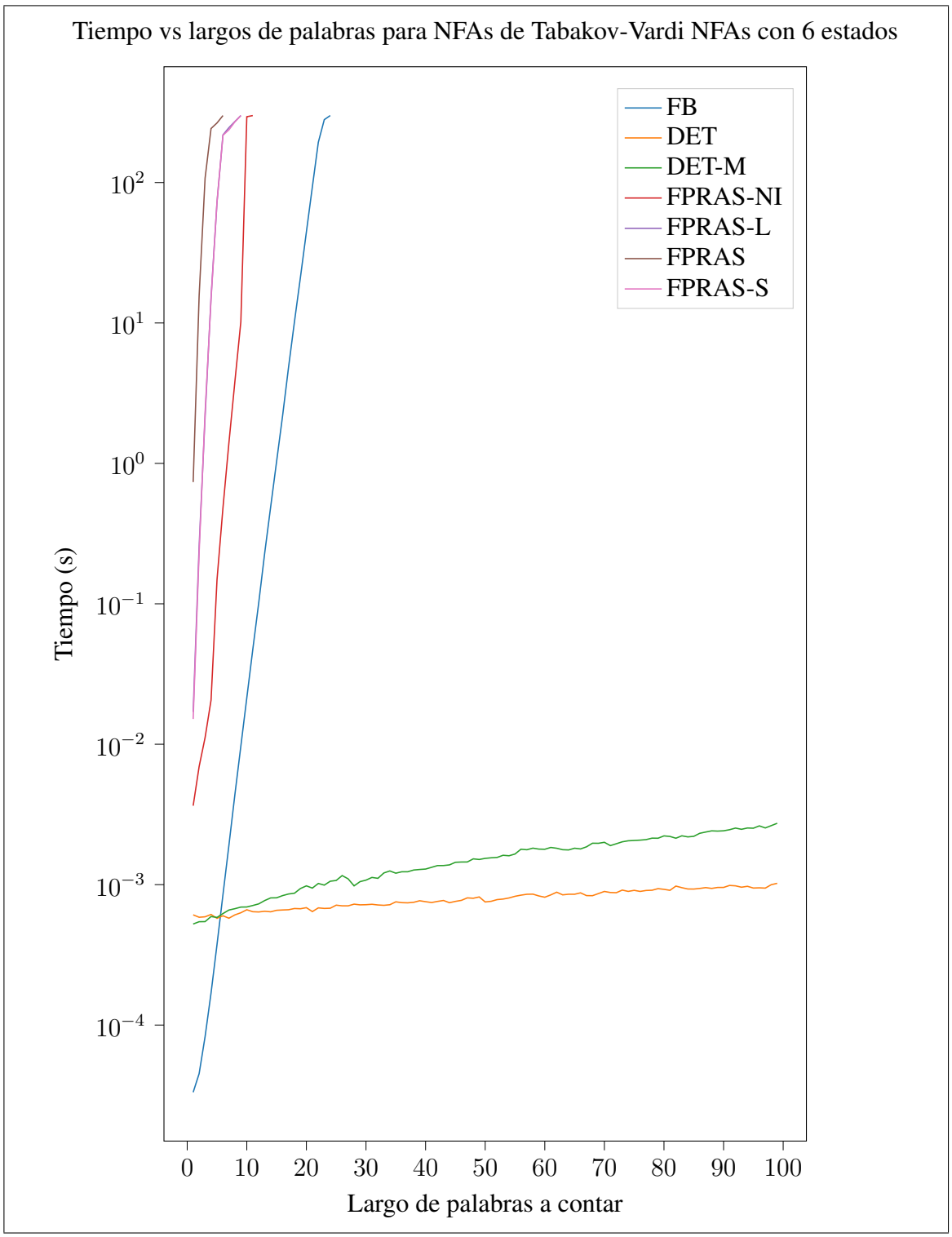


Figura A.5. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 6 estados.

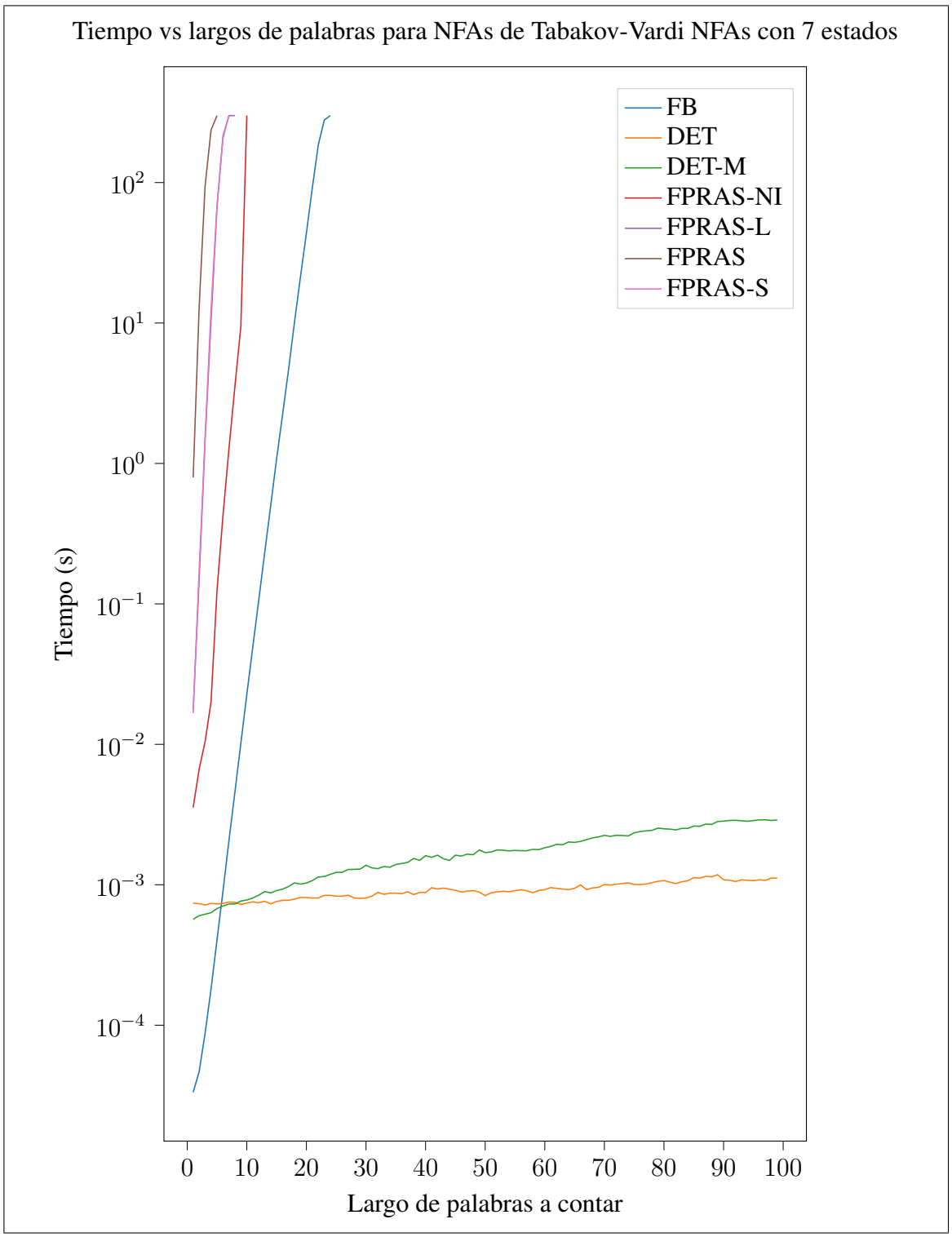


Figura A.6. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 7 estados.

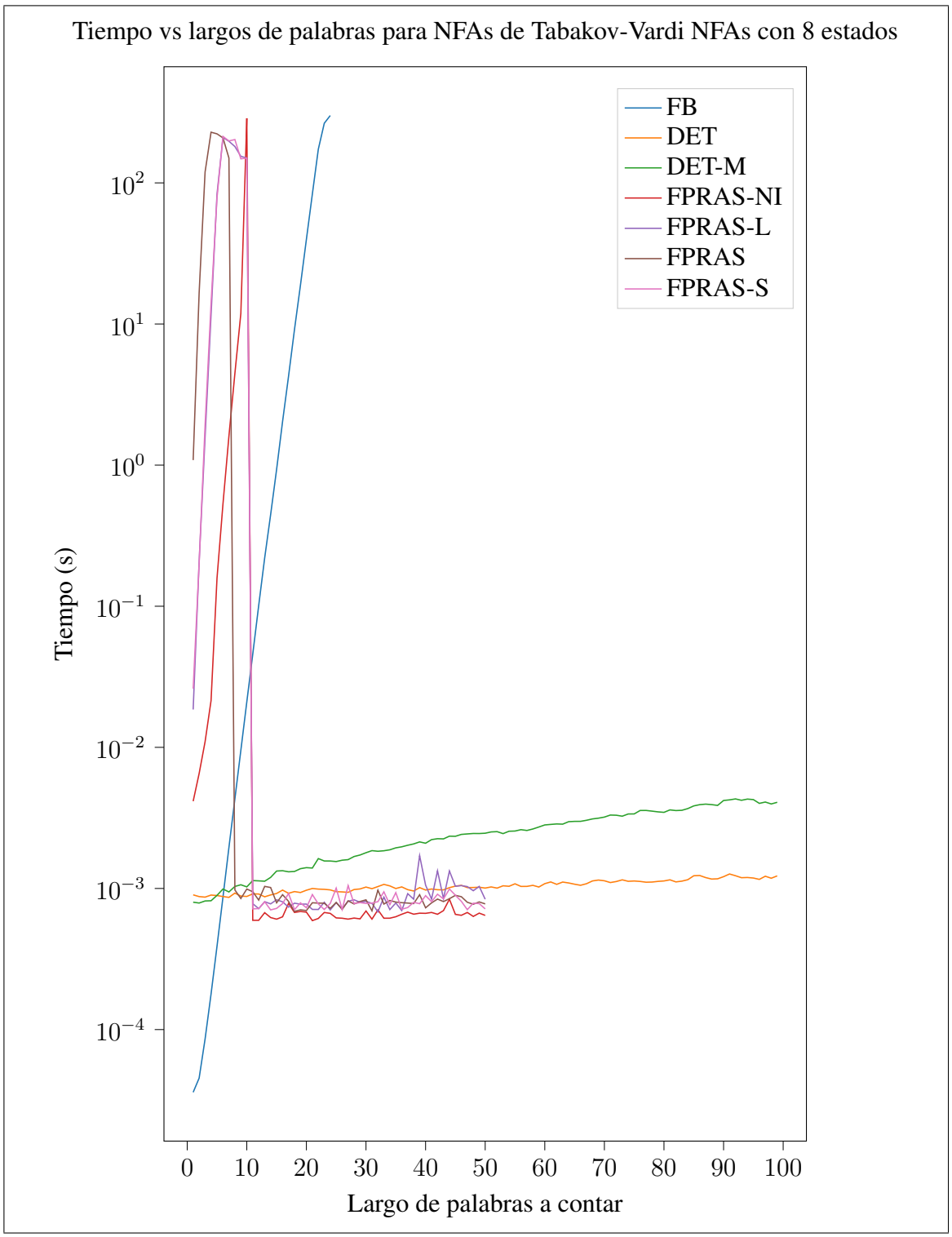


Figura A.7. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 8 estados.

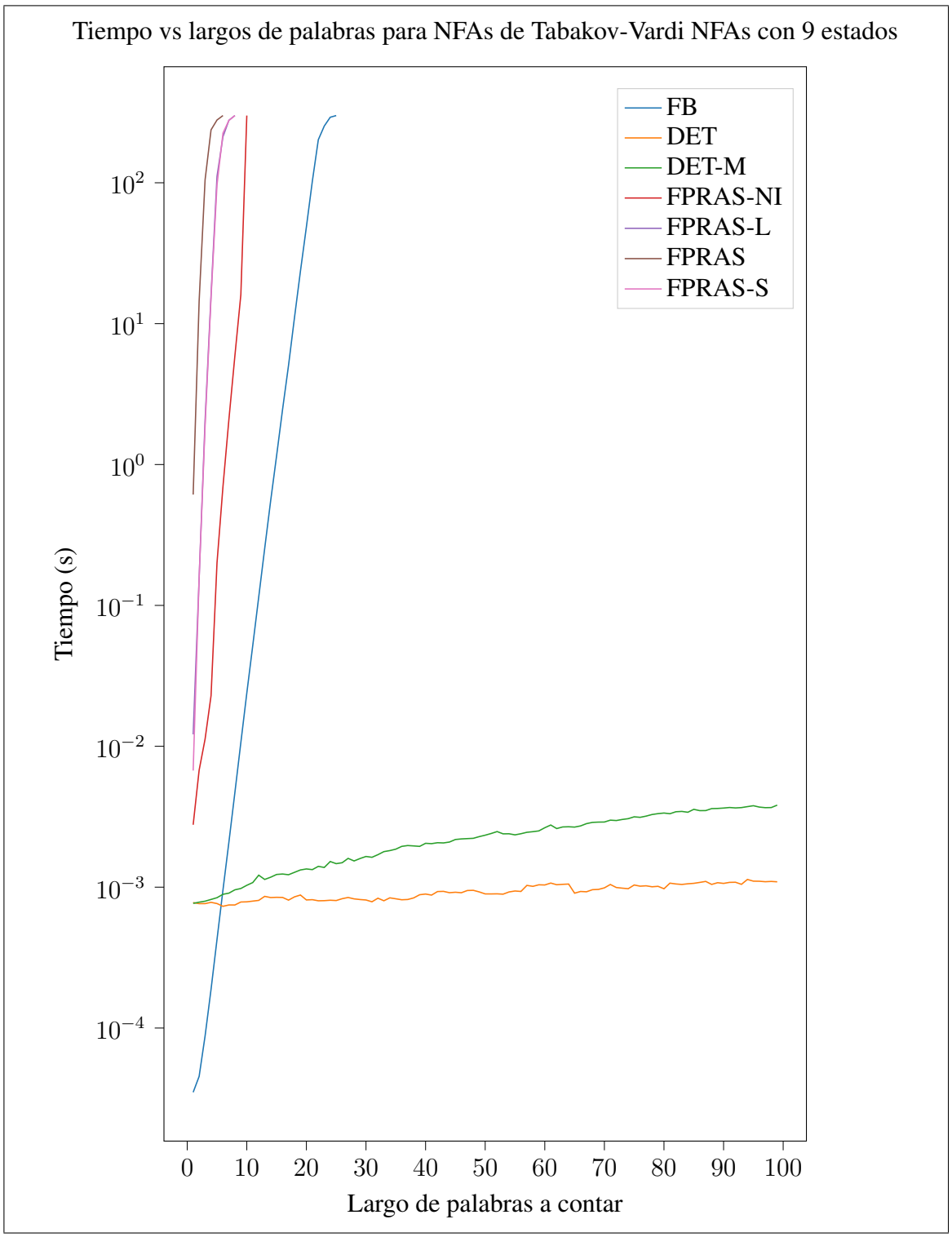


Figura A.8. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 9 estados.

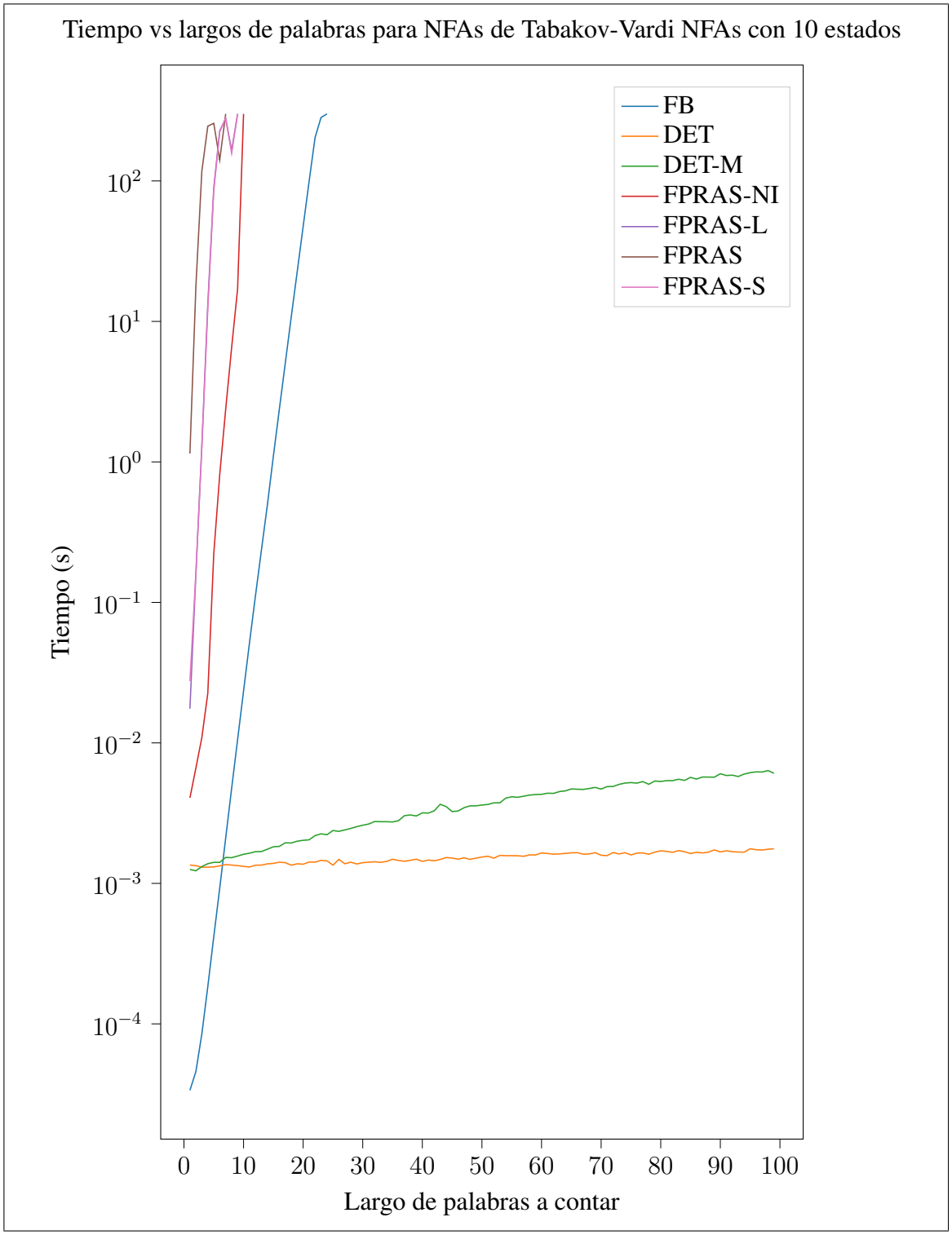


Figura A.9. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 10 estados.

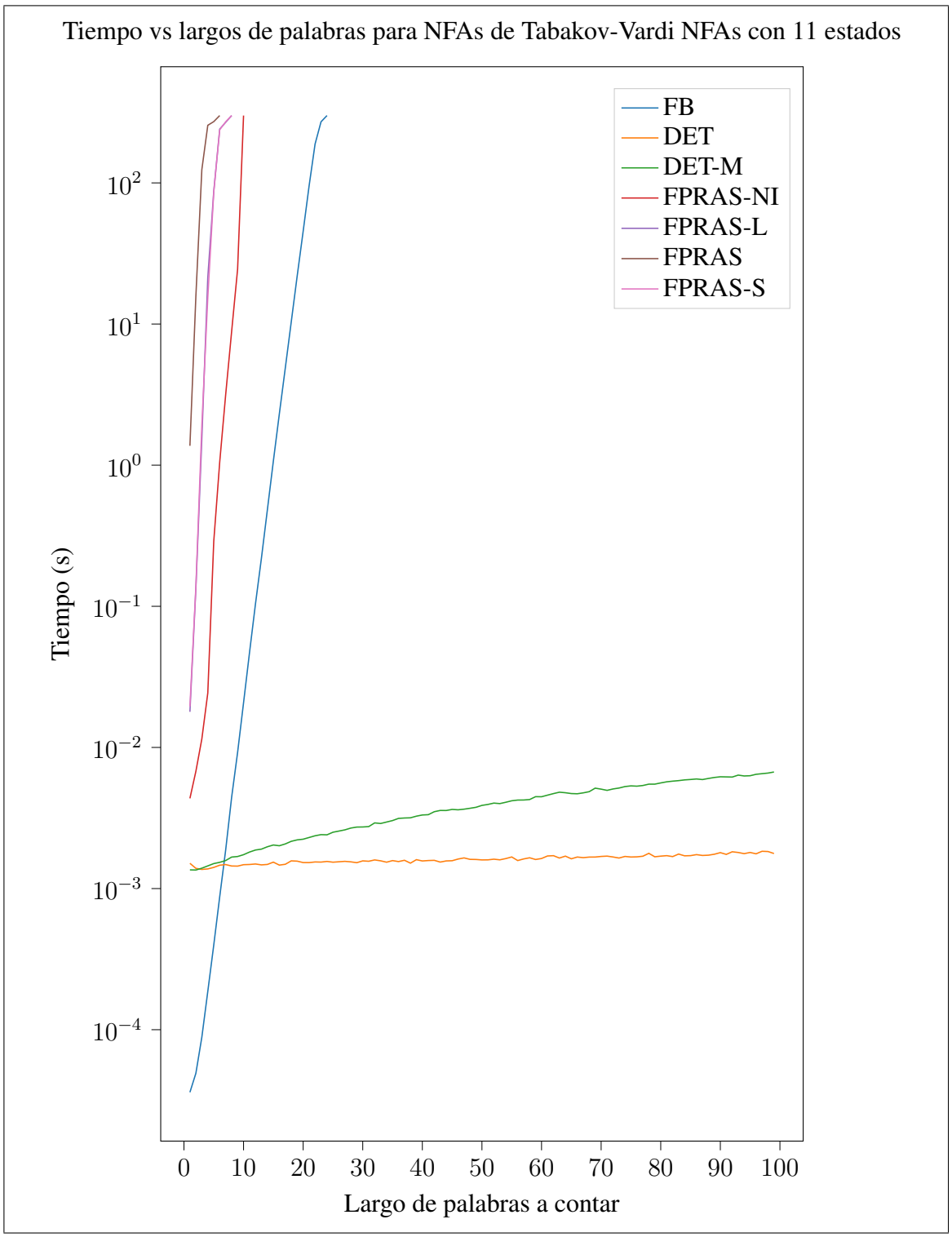


Figura A.10. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 11 estados.

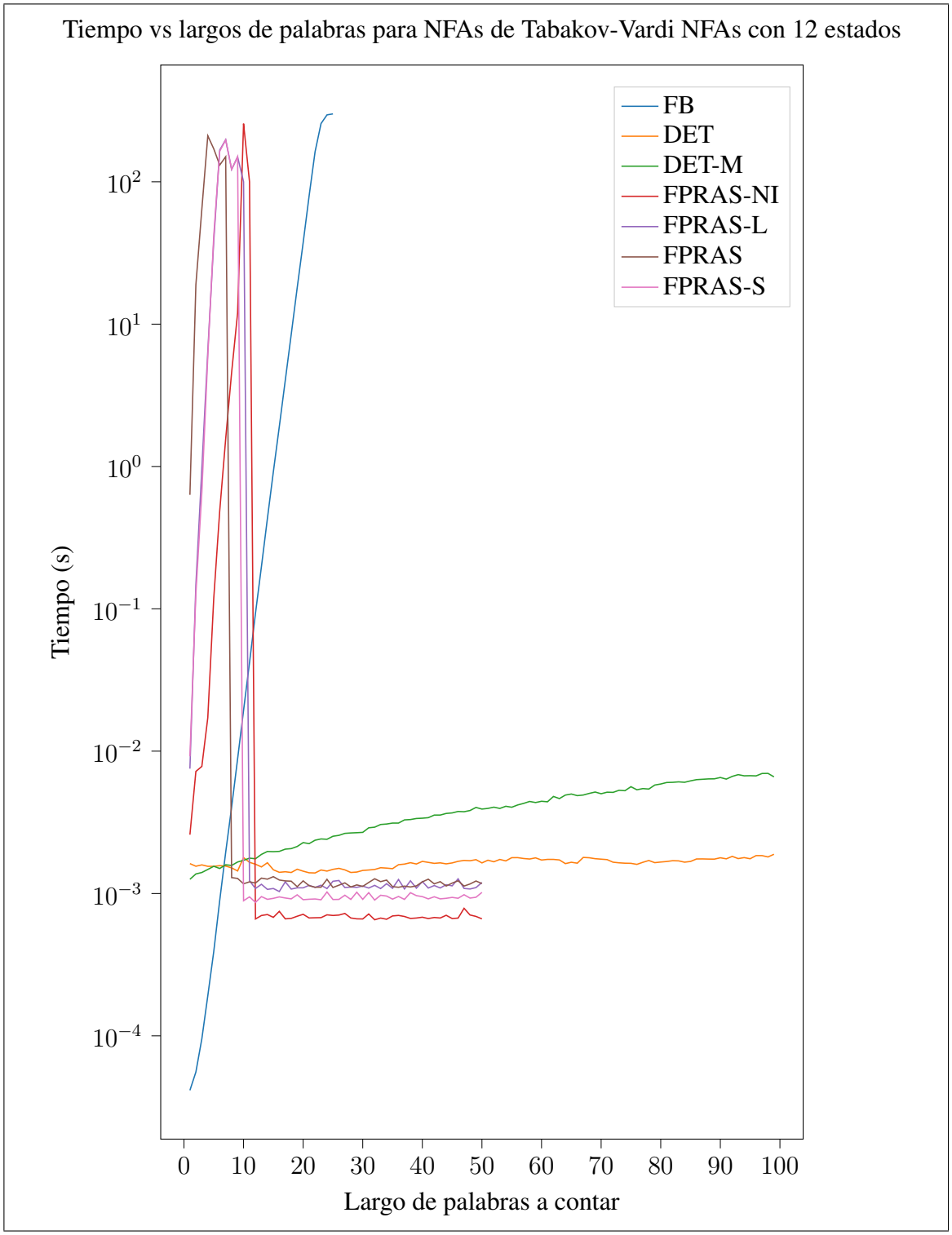


Figura A.11. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 12 estados.

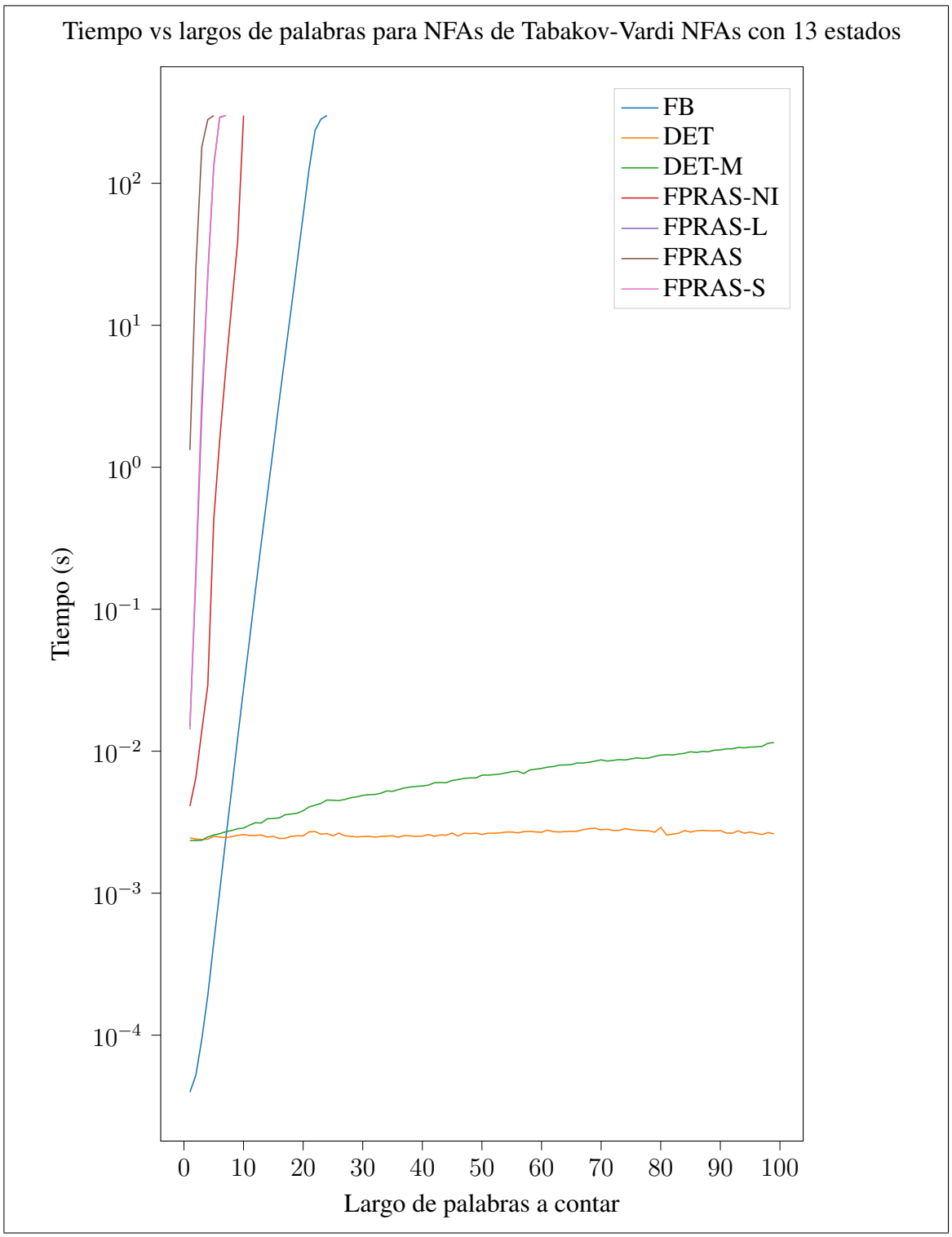


Figura A.12. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 13 estados.

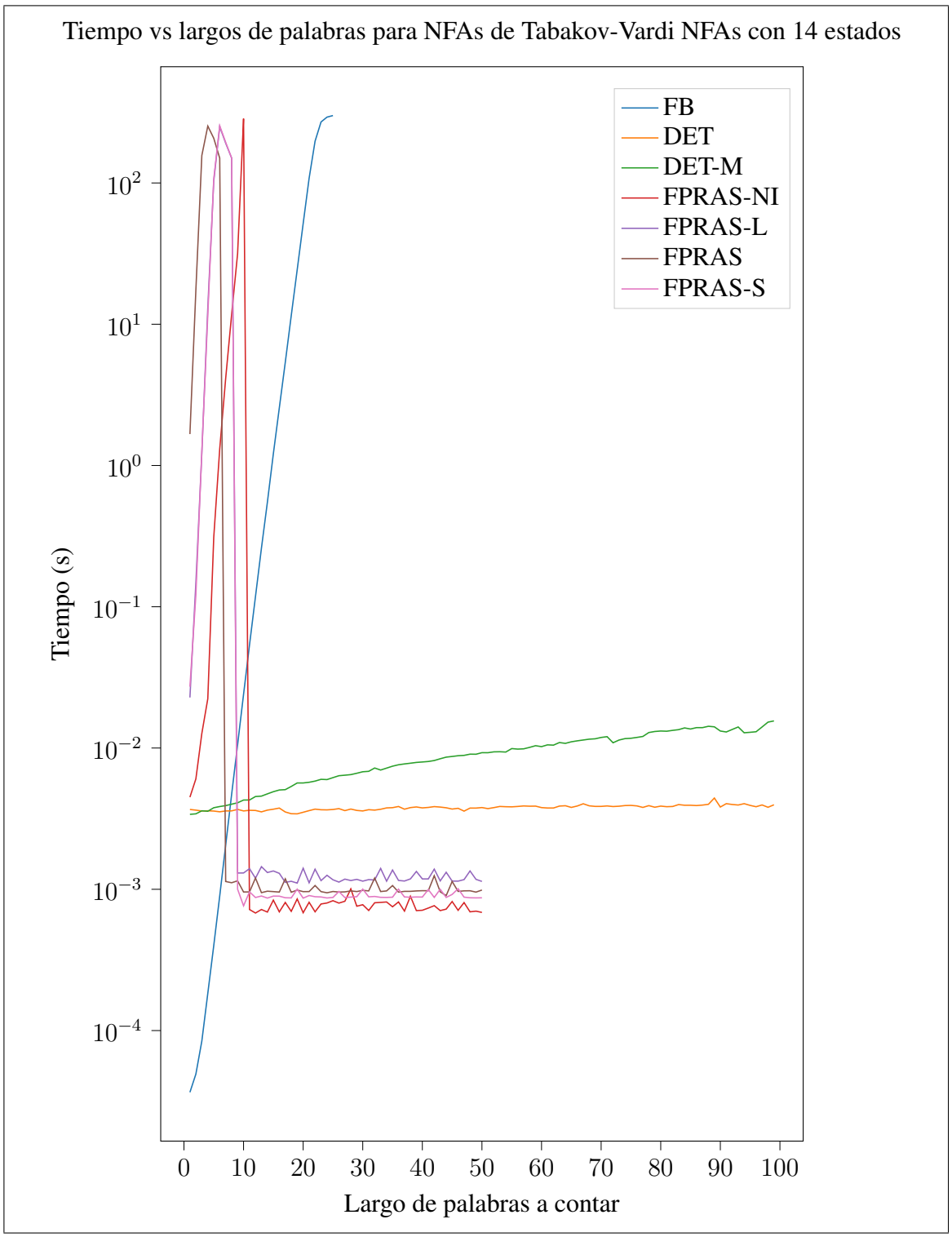


Figura A.13. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 14 estados.

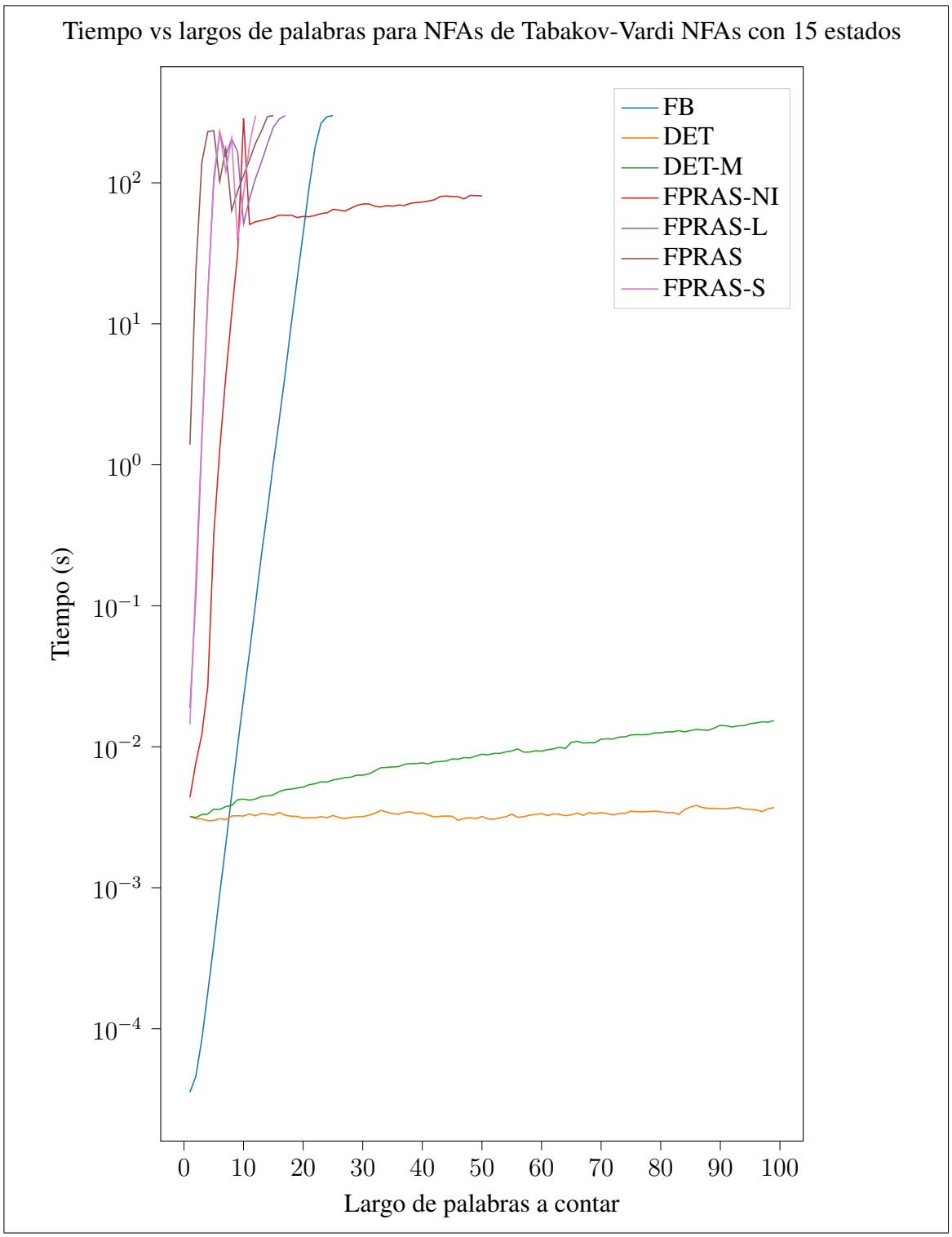


Figura A.14. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 15 estados.

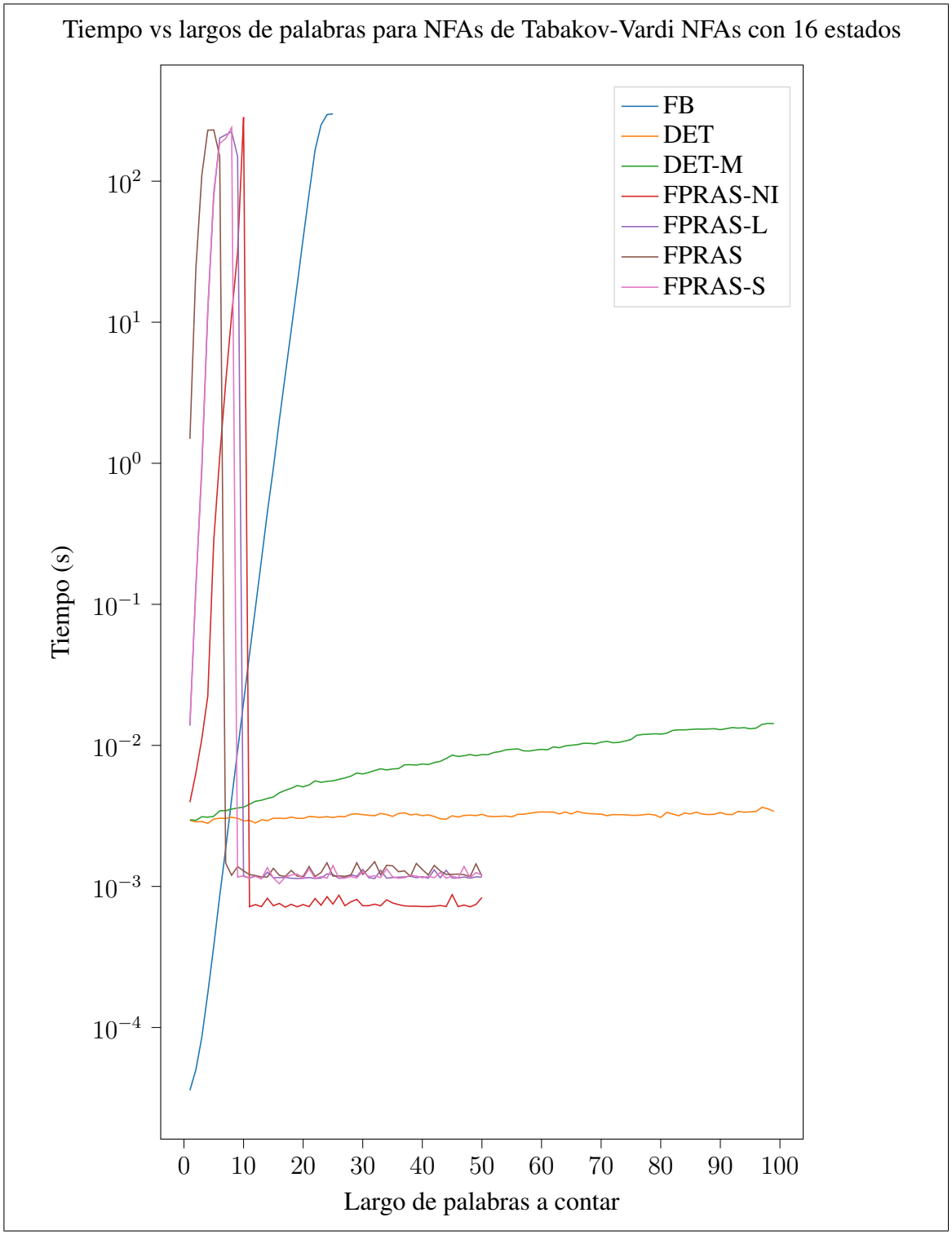


Figura A.15. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 16 estados.

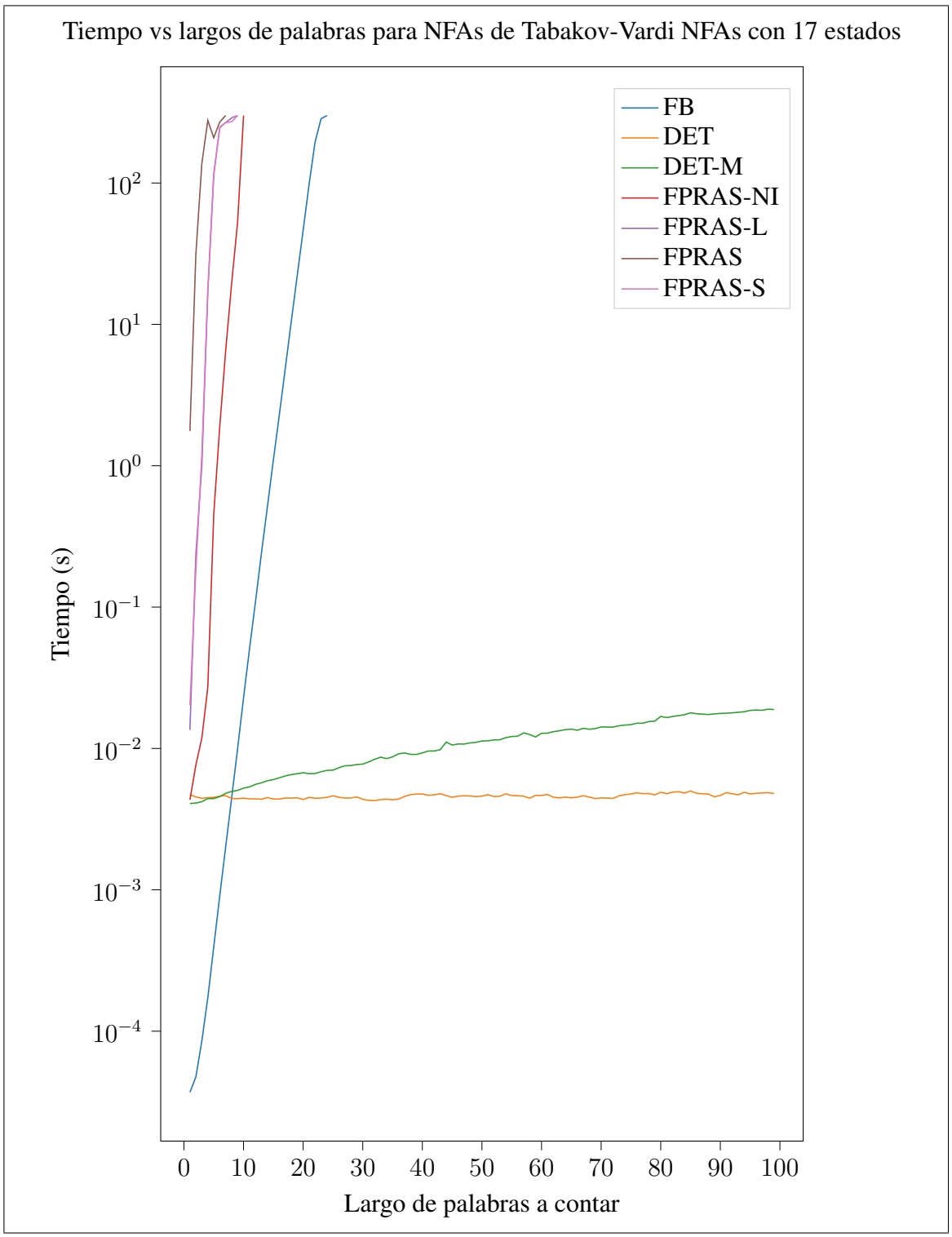


Figura A.16. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 17 estados.

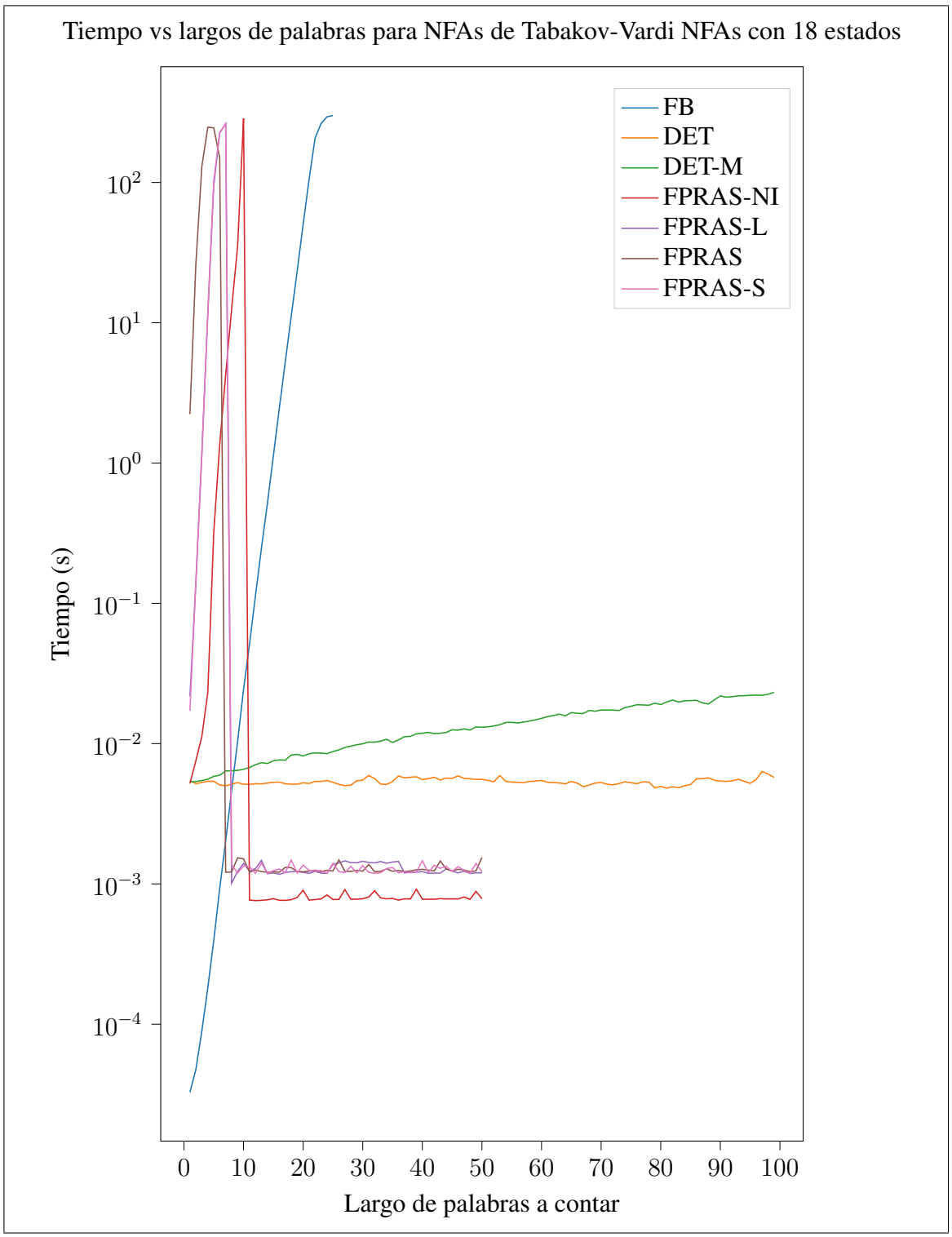


Figura A.17. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 18 estados.

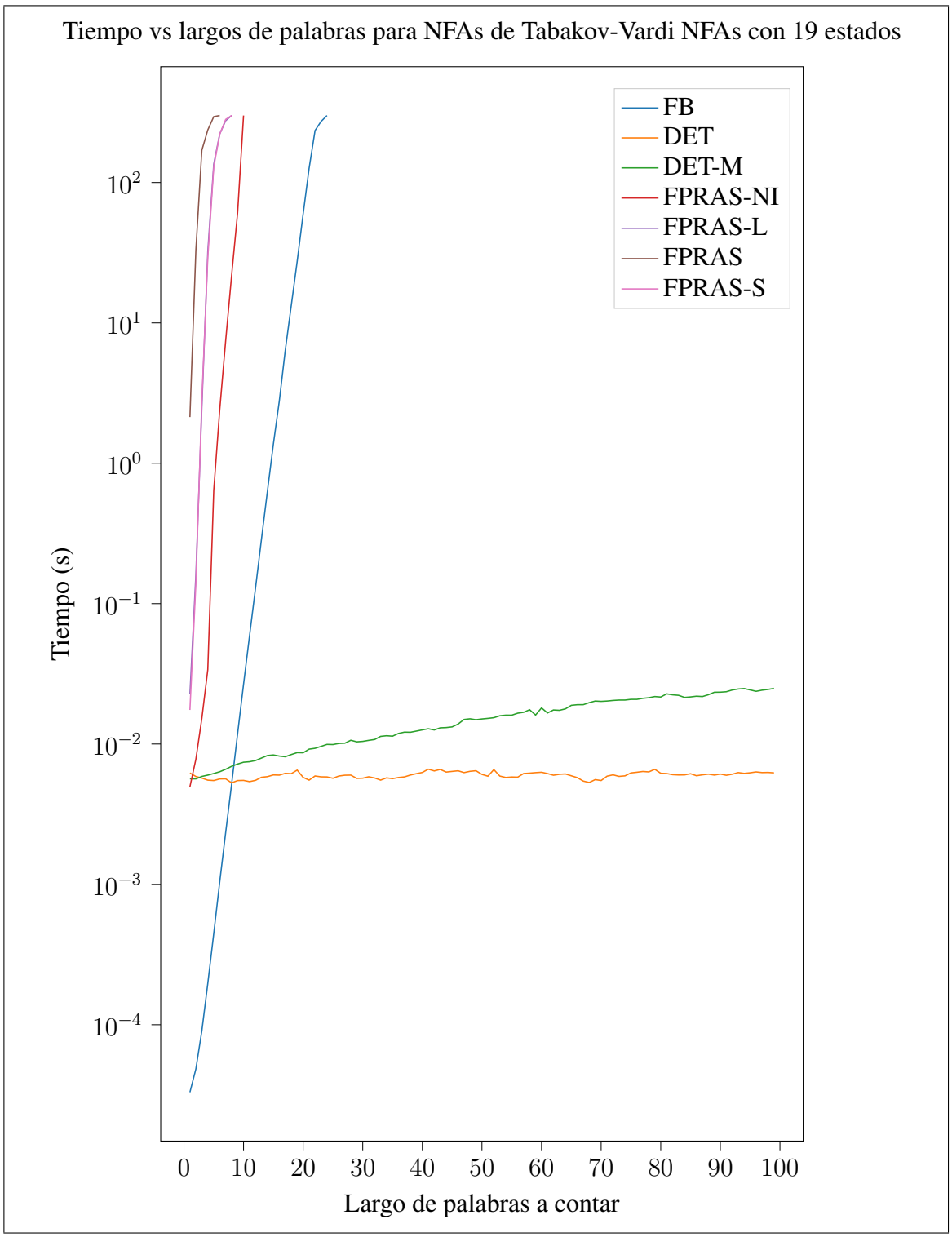


Figura A.18. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 19 estados.

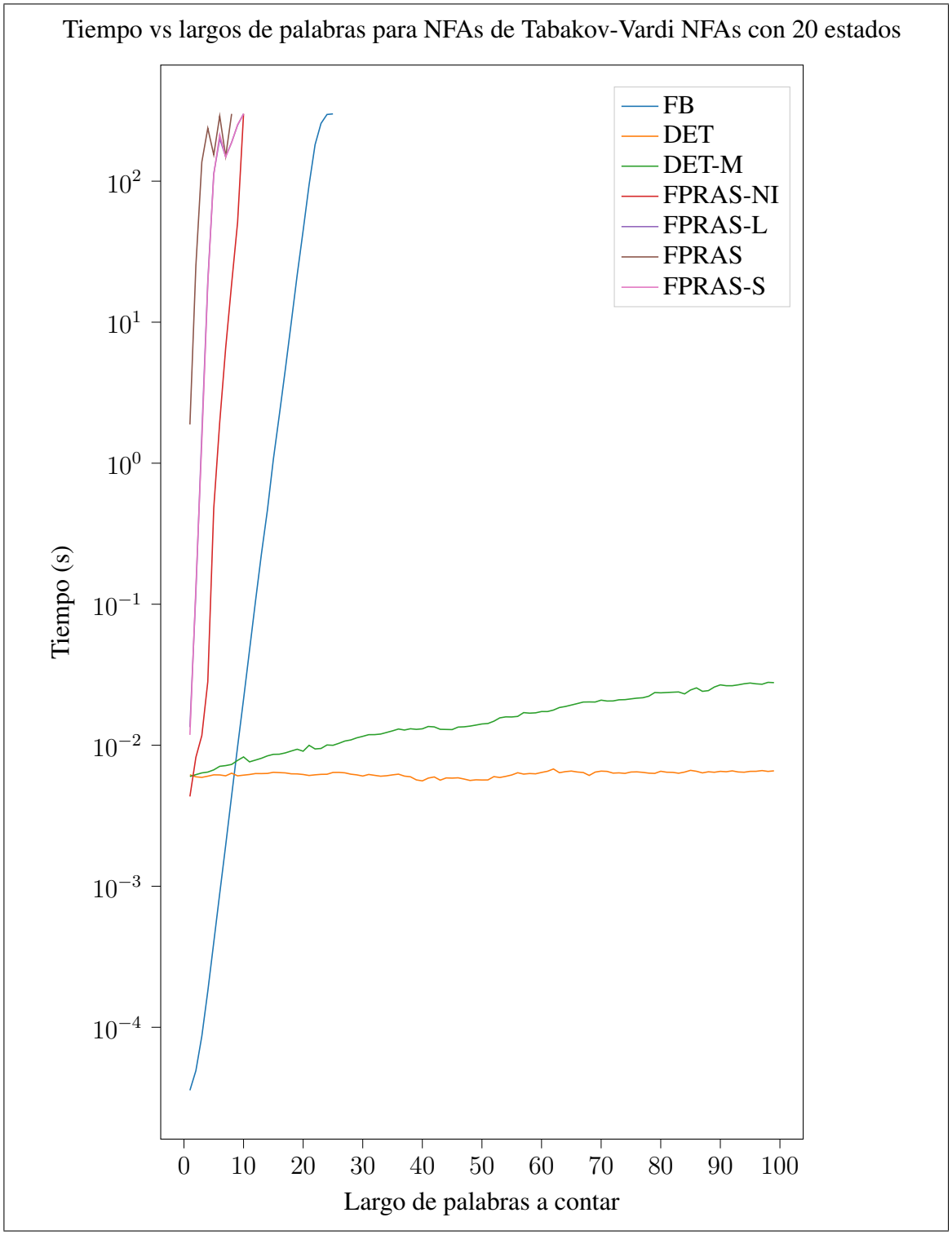


Figura A.19. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 20 estados.

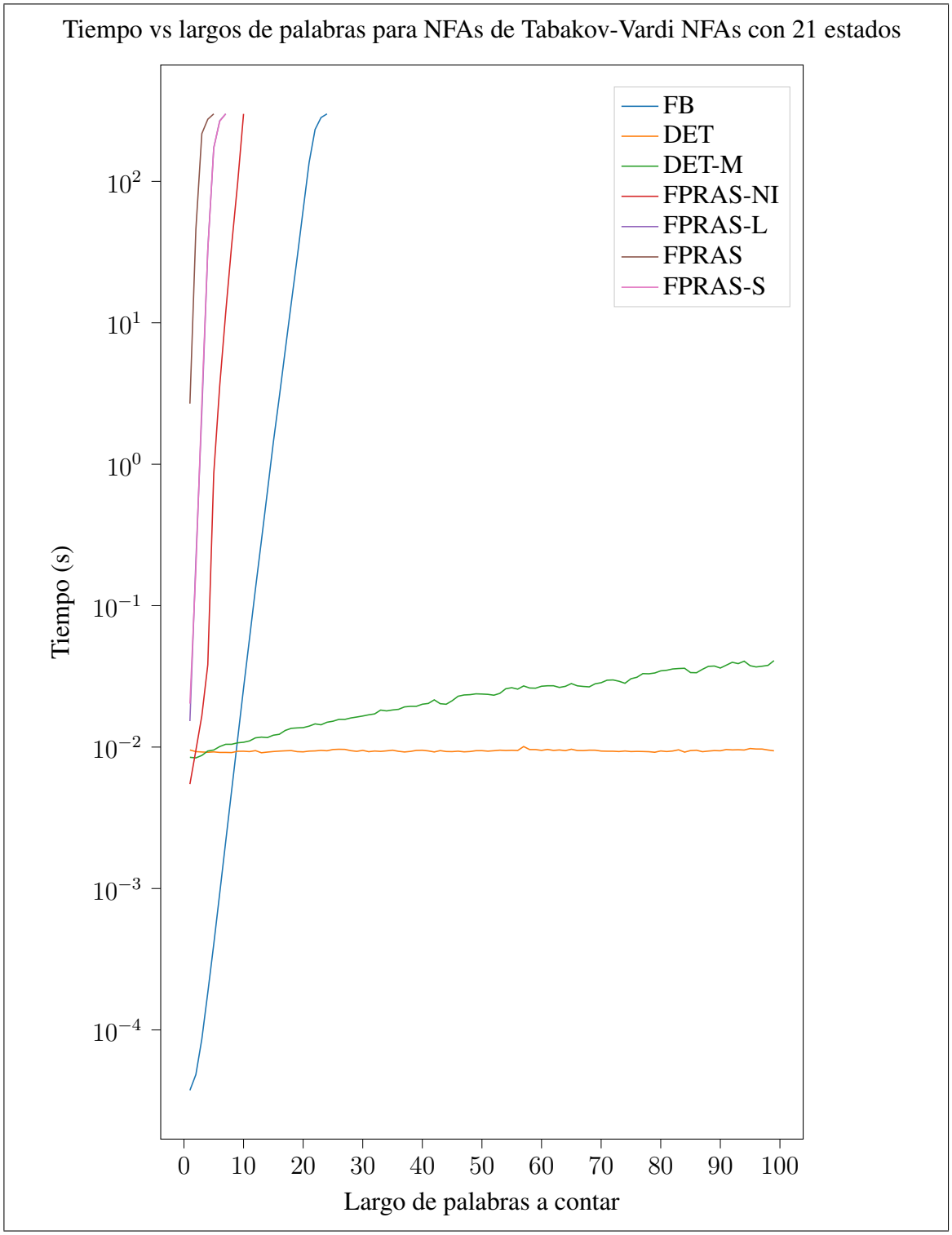


Figura A.20. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 21 estados.

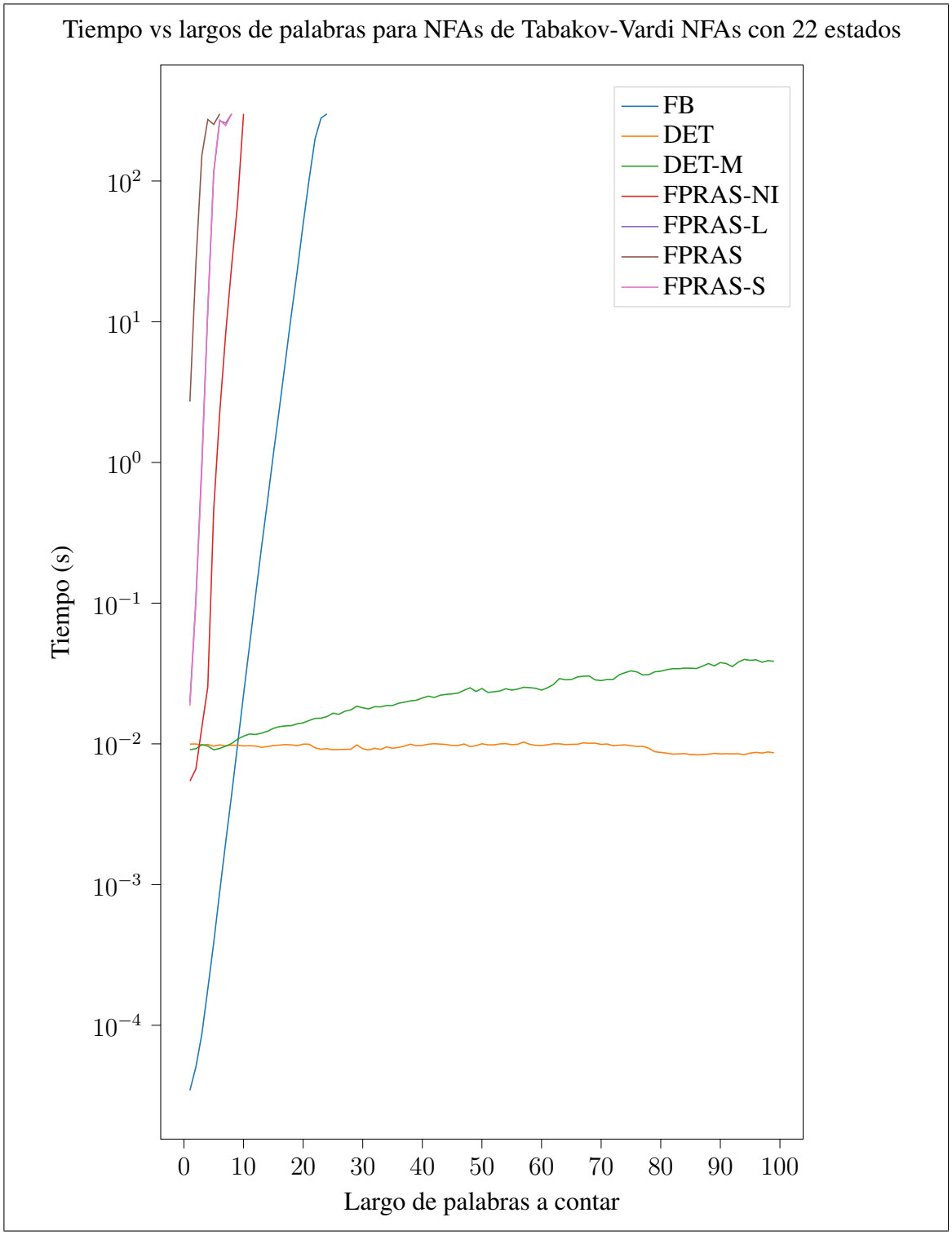


Figura A.21. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 22 estados.

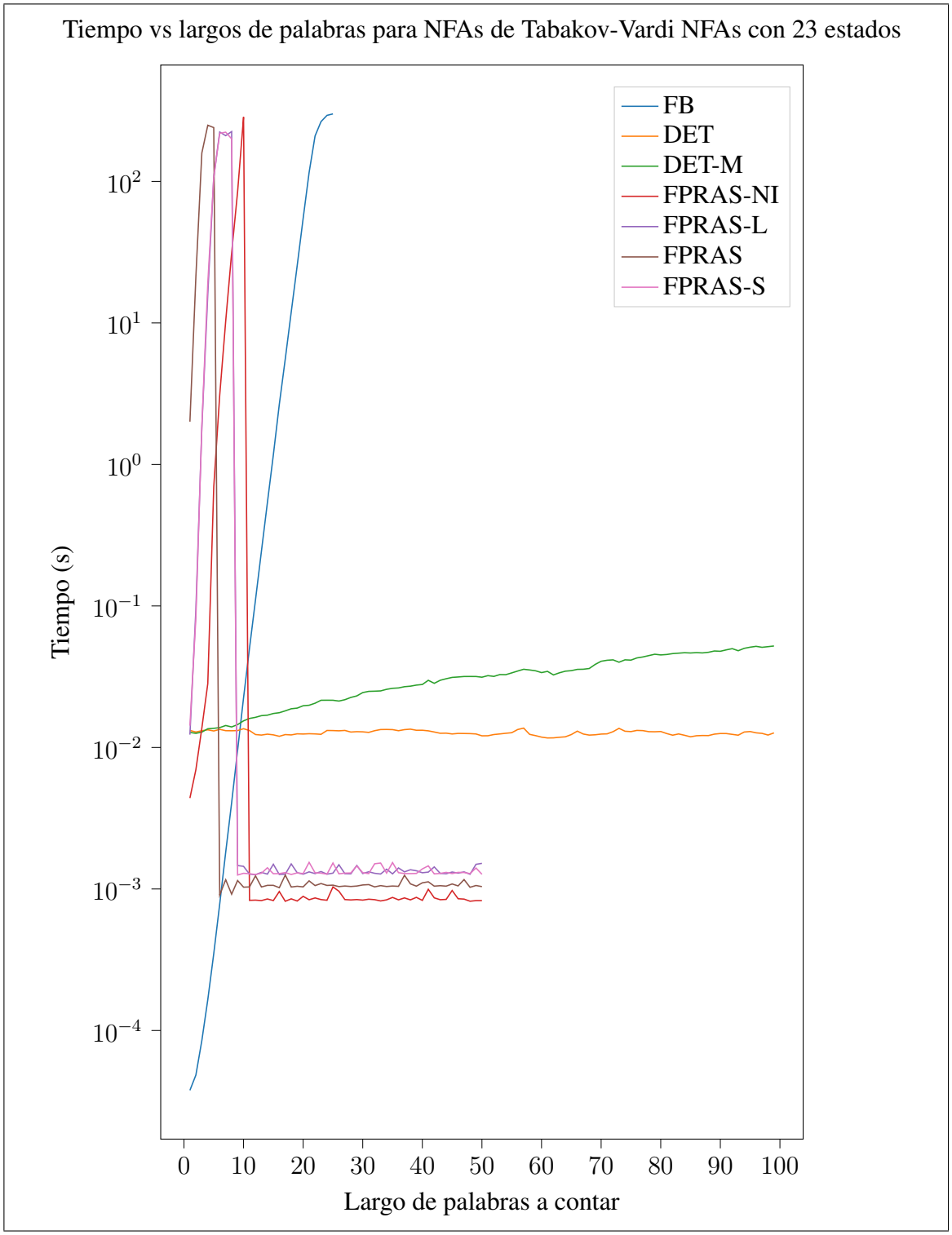


Figura A.22. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 23 estados.

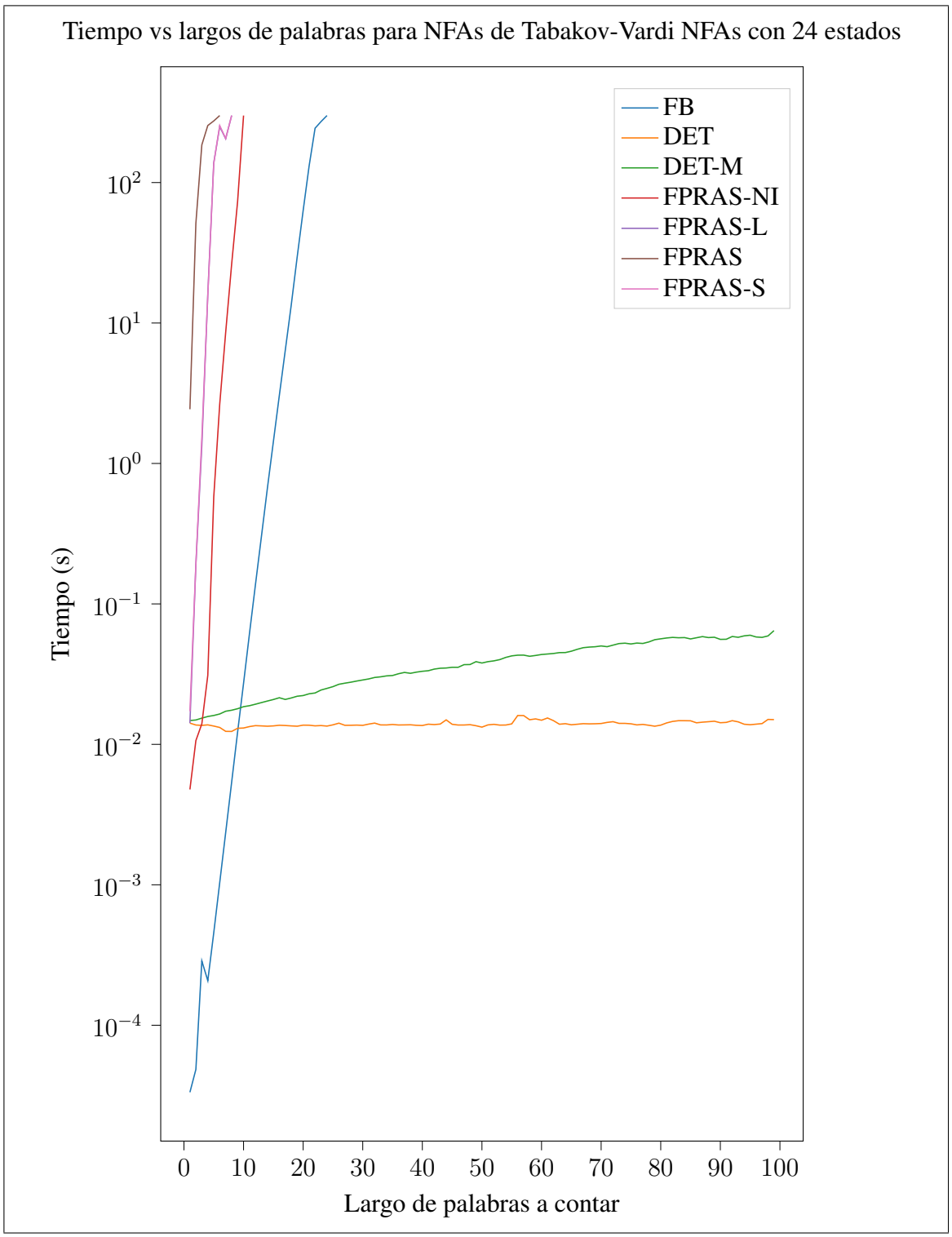


Figura A.23. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 24 estados.

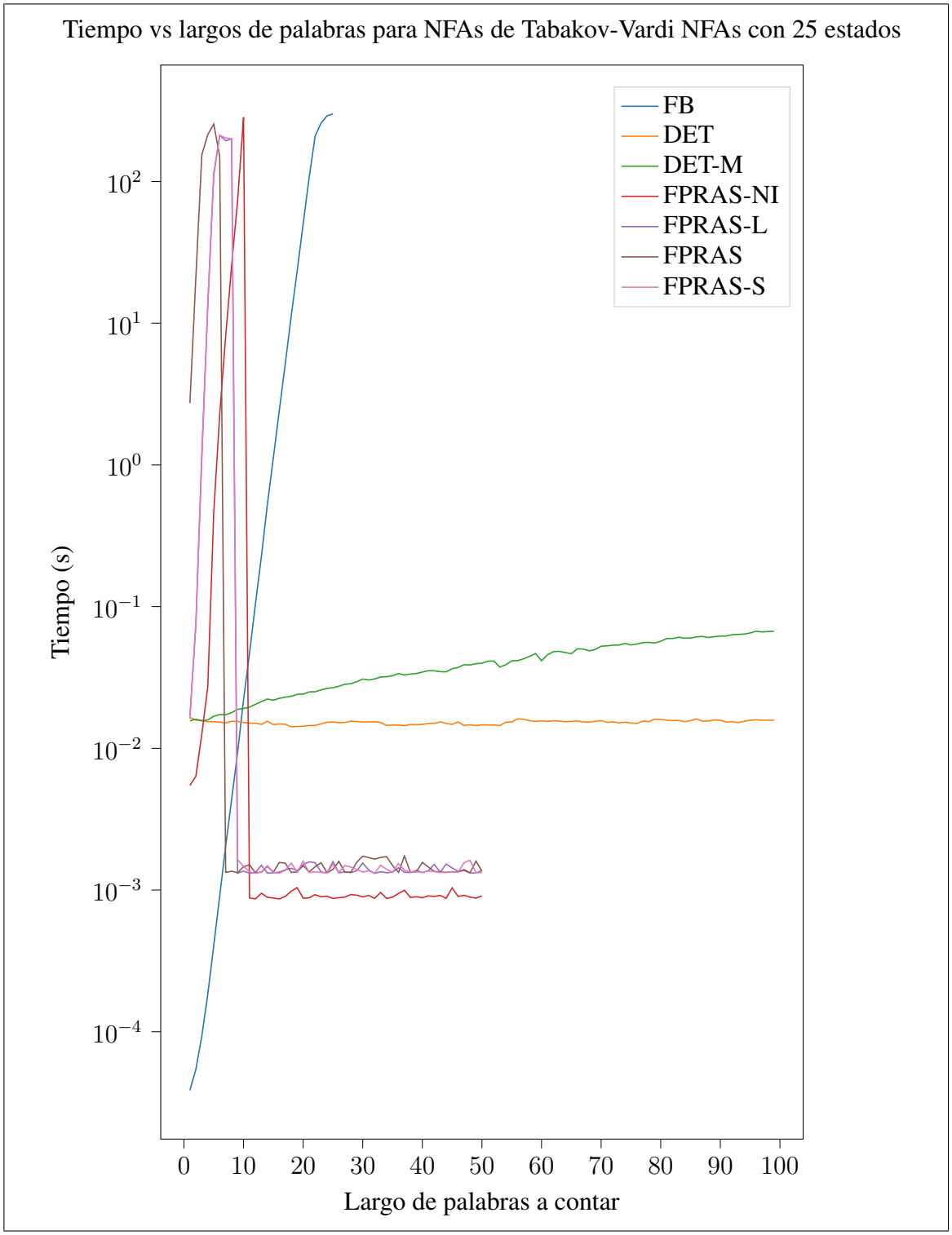


Figura A.24. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 25 estados.

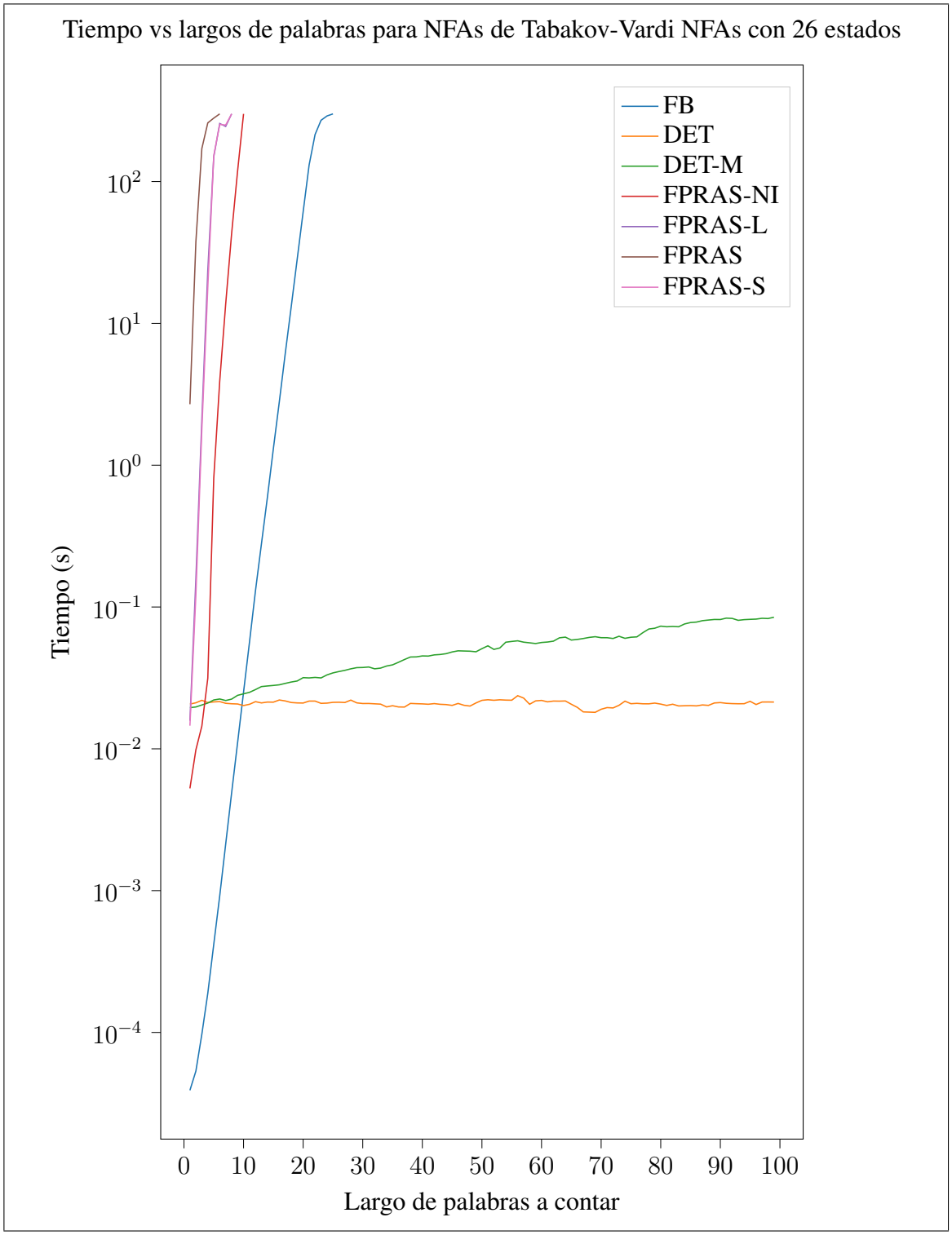


Figura A.25. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 26 estados.

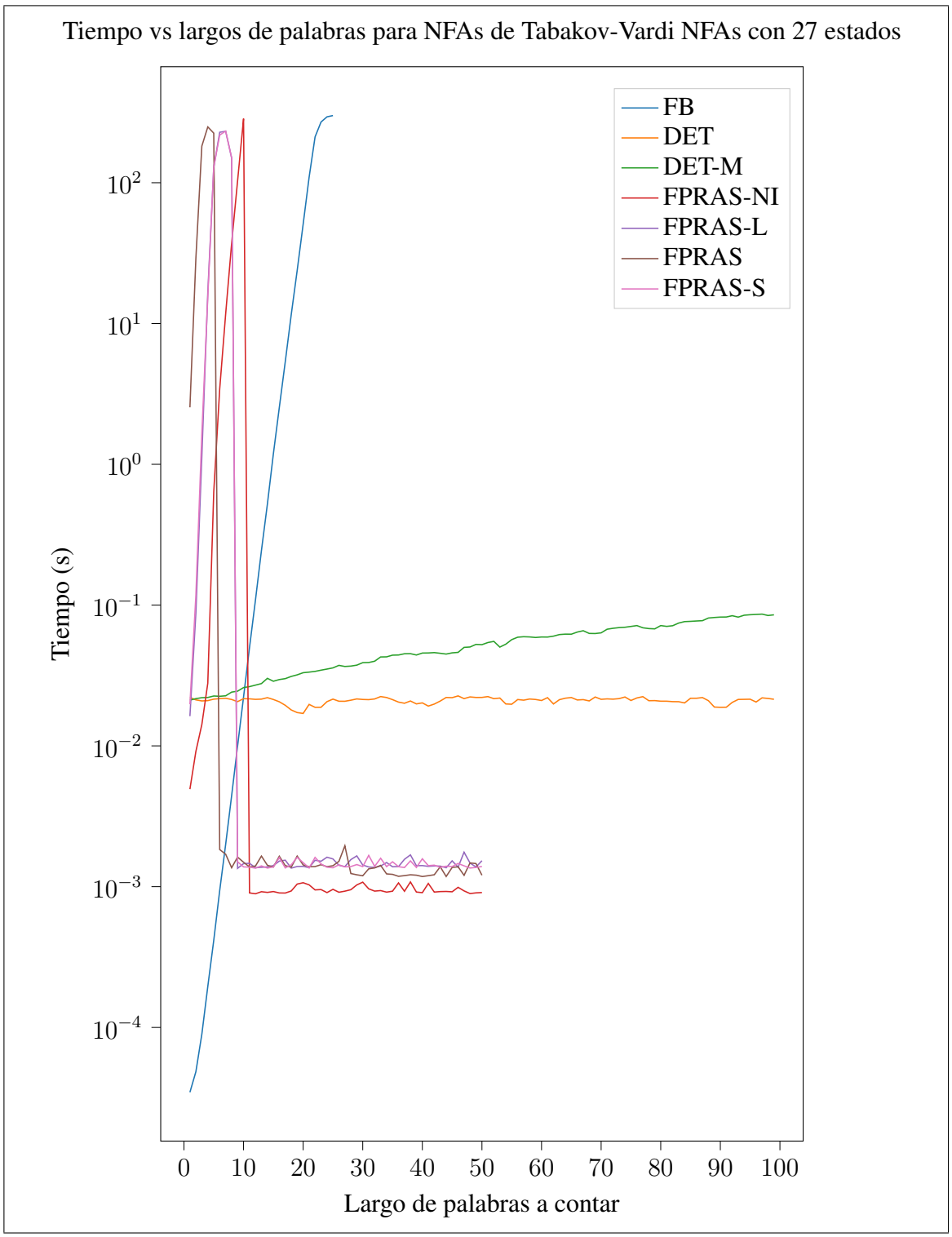


Figura A.26. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 27 estados.

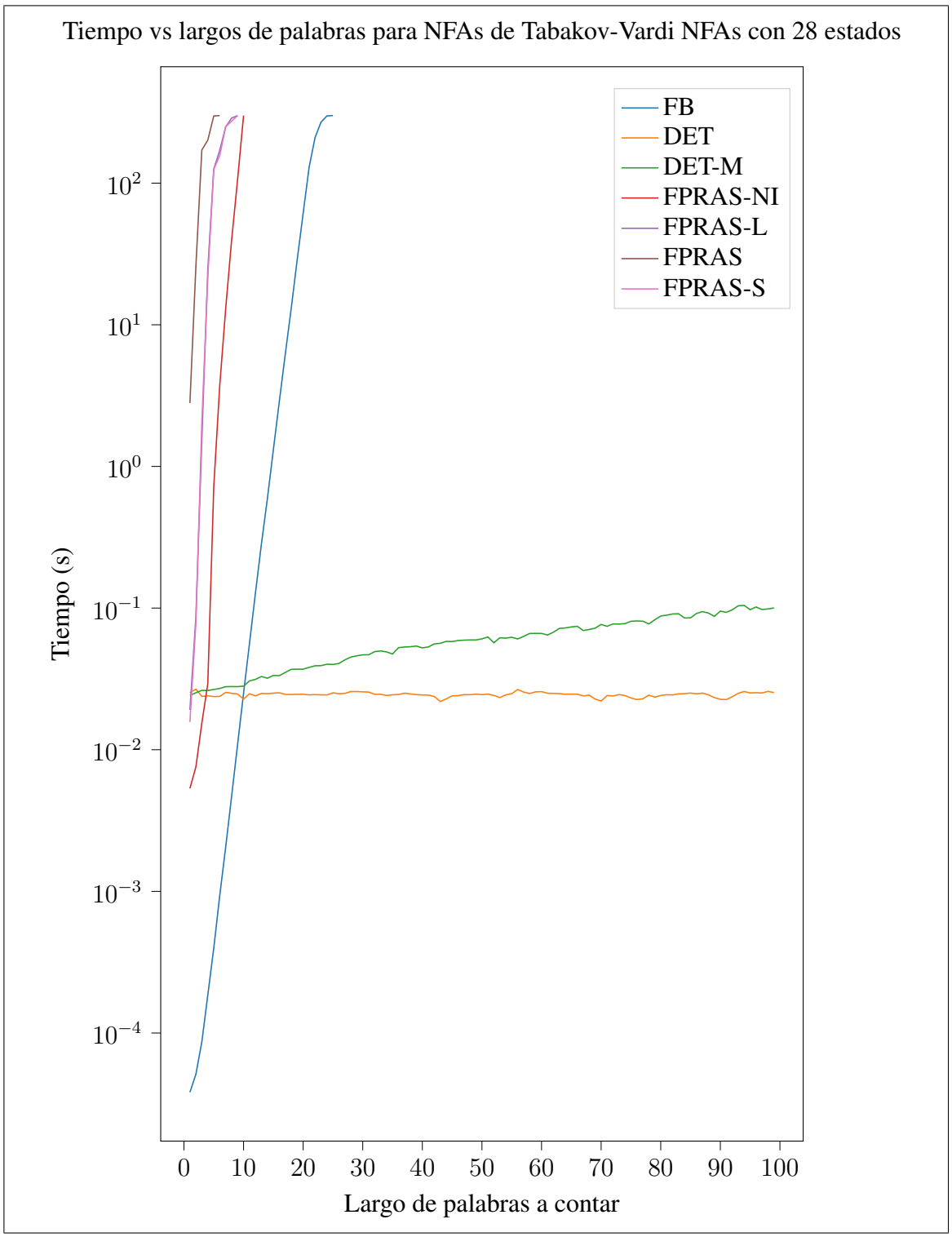


Figura A.27. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 28 estados.

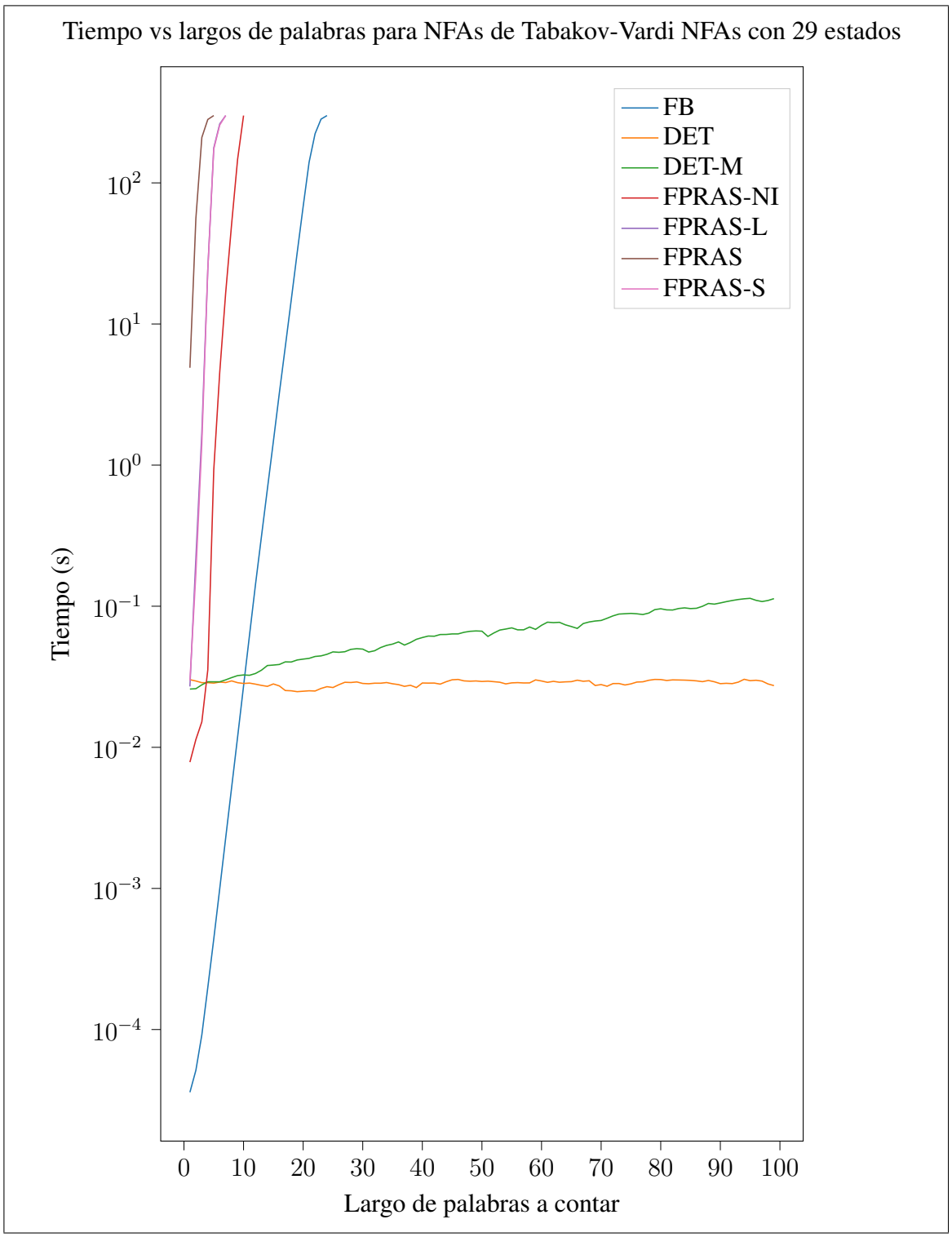


Figura A.28. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 29 estados.

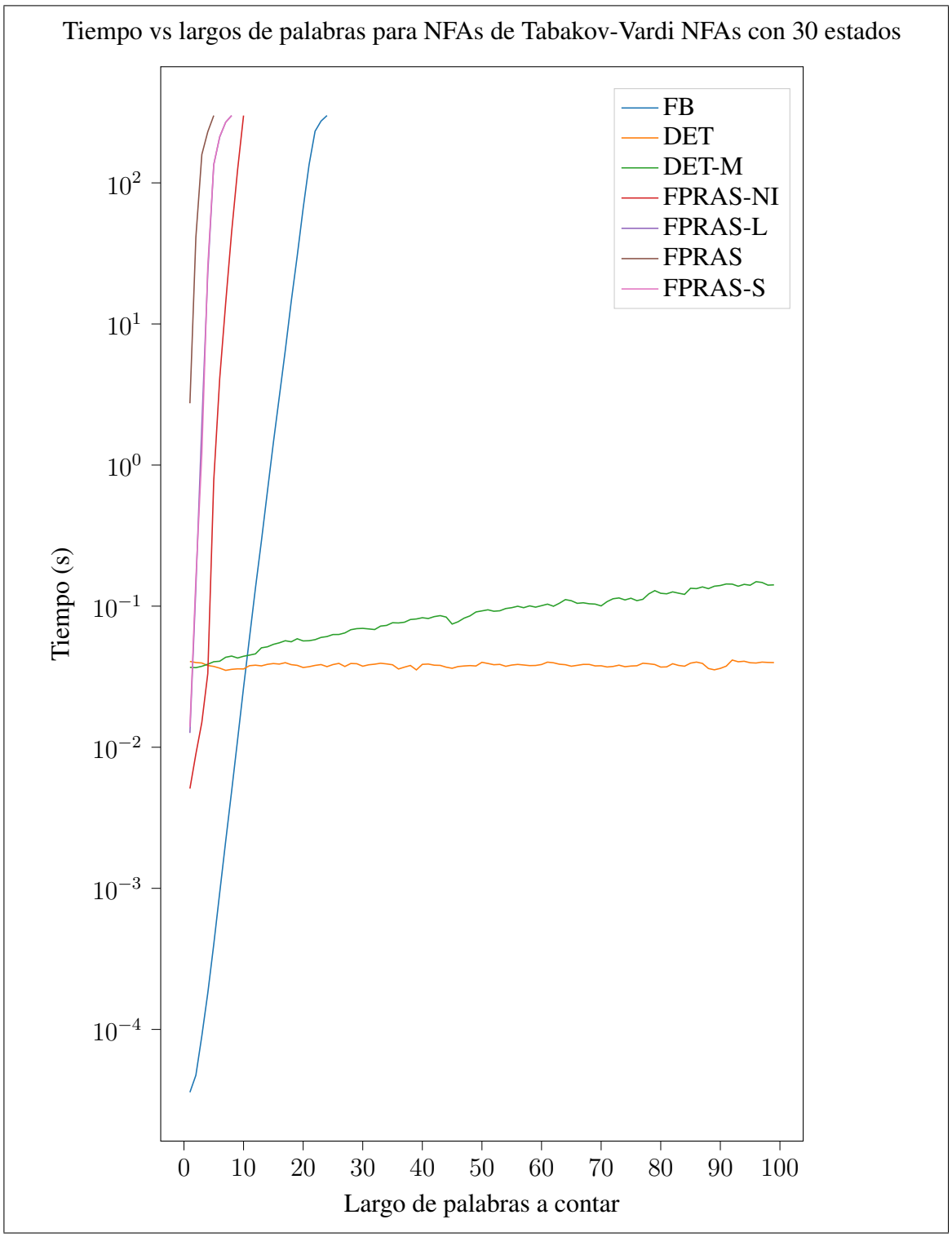


Figura A.29. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 30 estados.

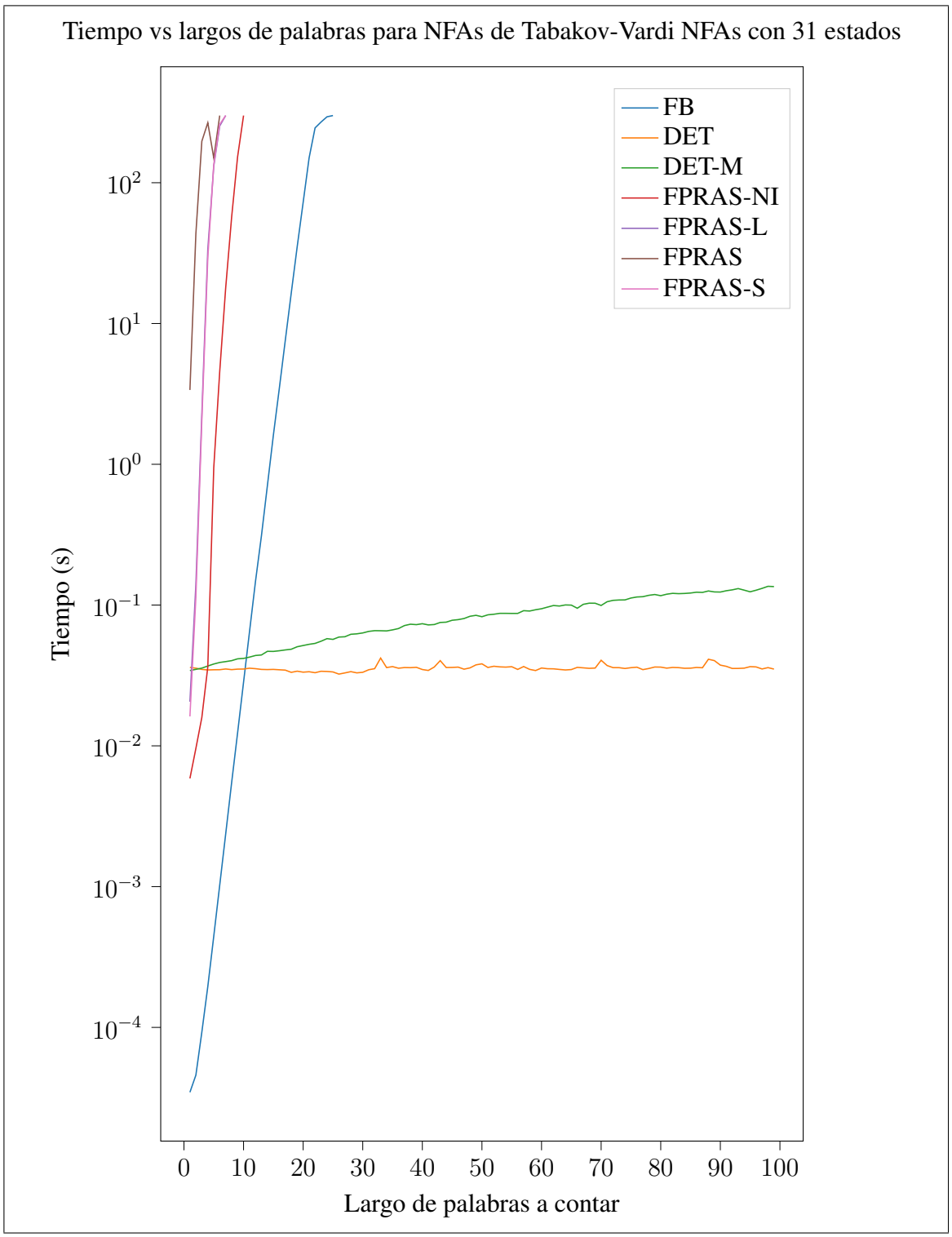


Figura A.30. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 31 estados.

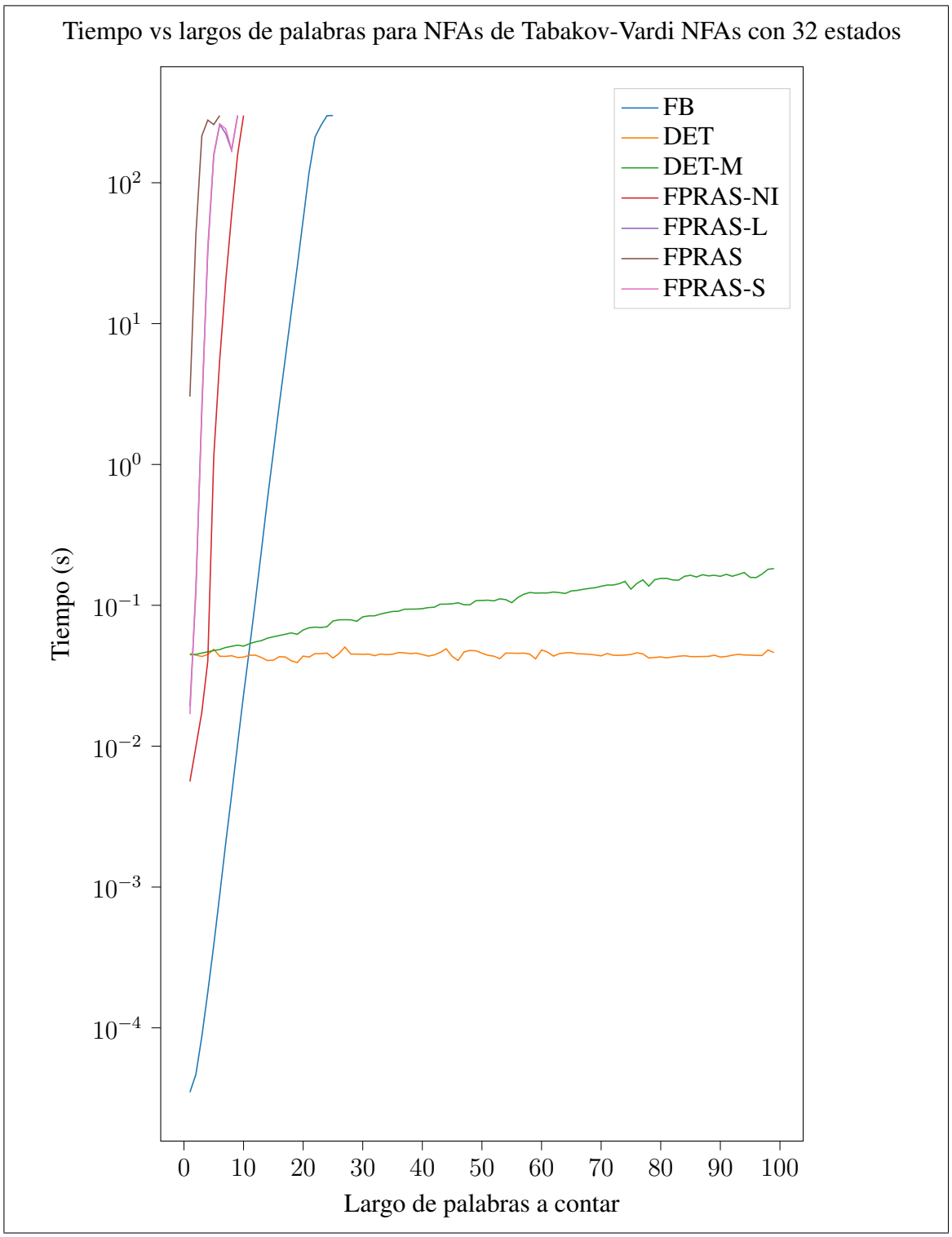


Figura A.31. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 32 estados.

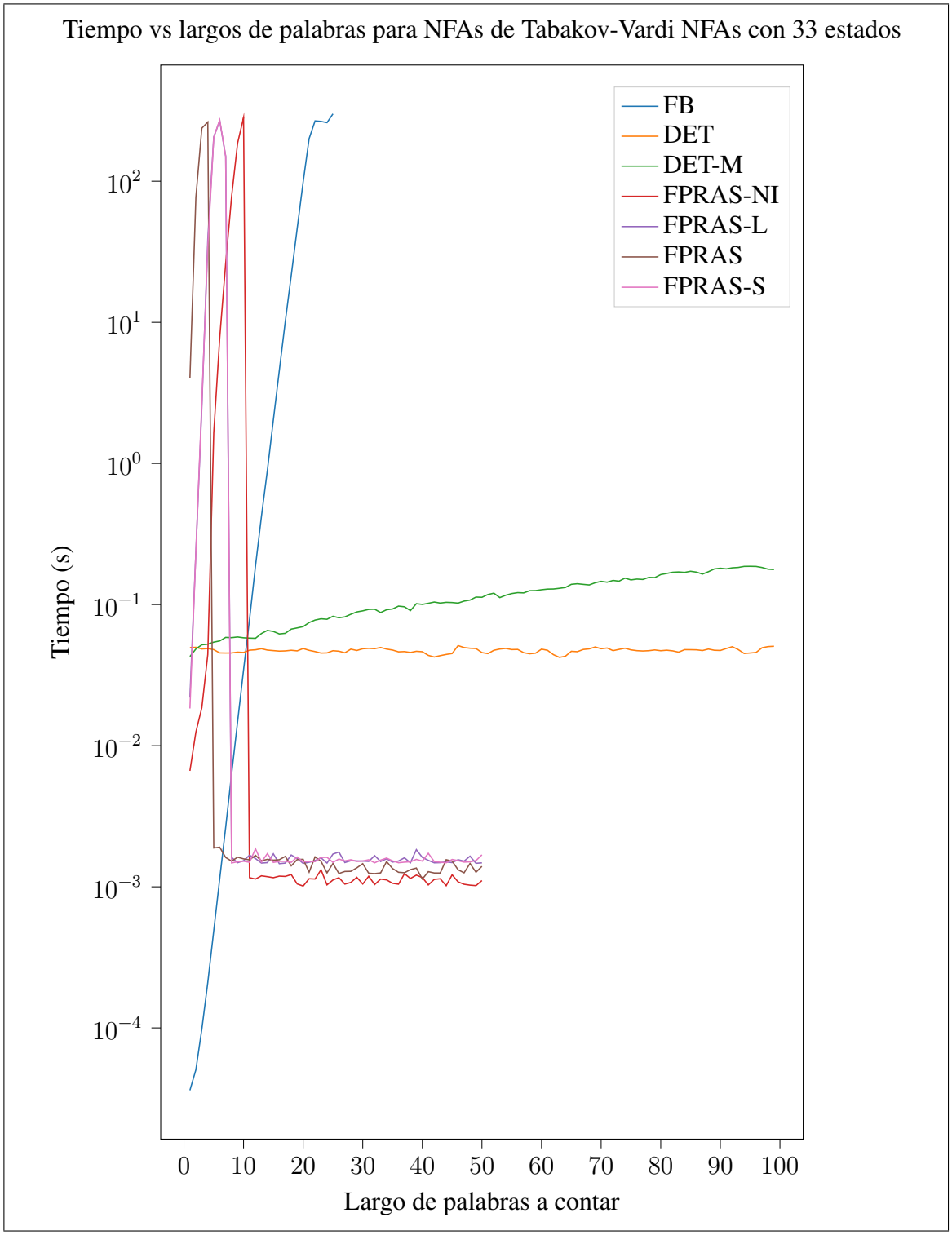


Figura A.32. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 33 estados.

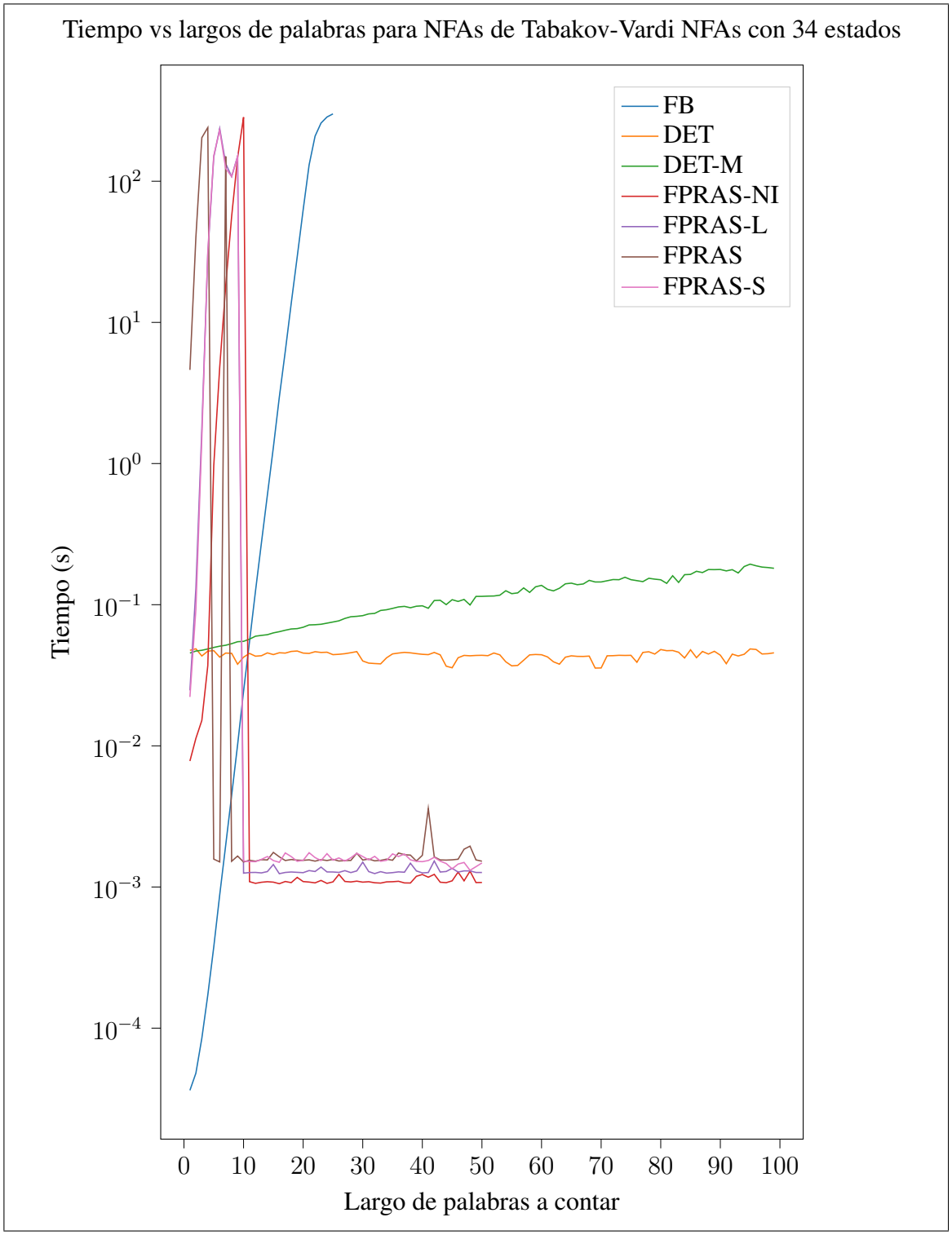


Figura A.33. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 34 estados.

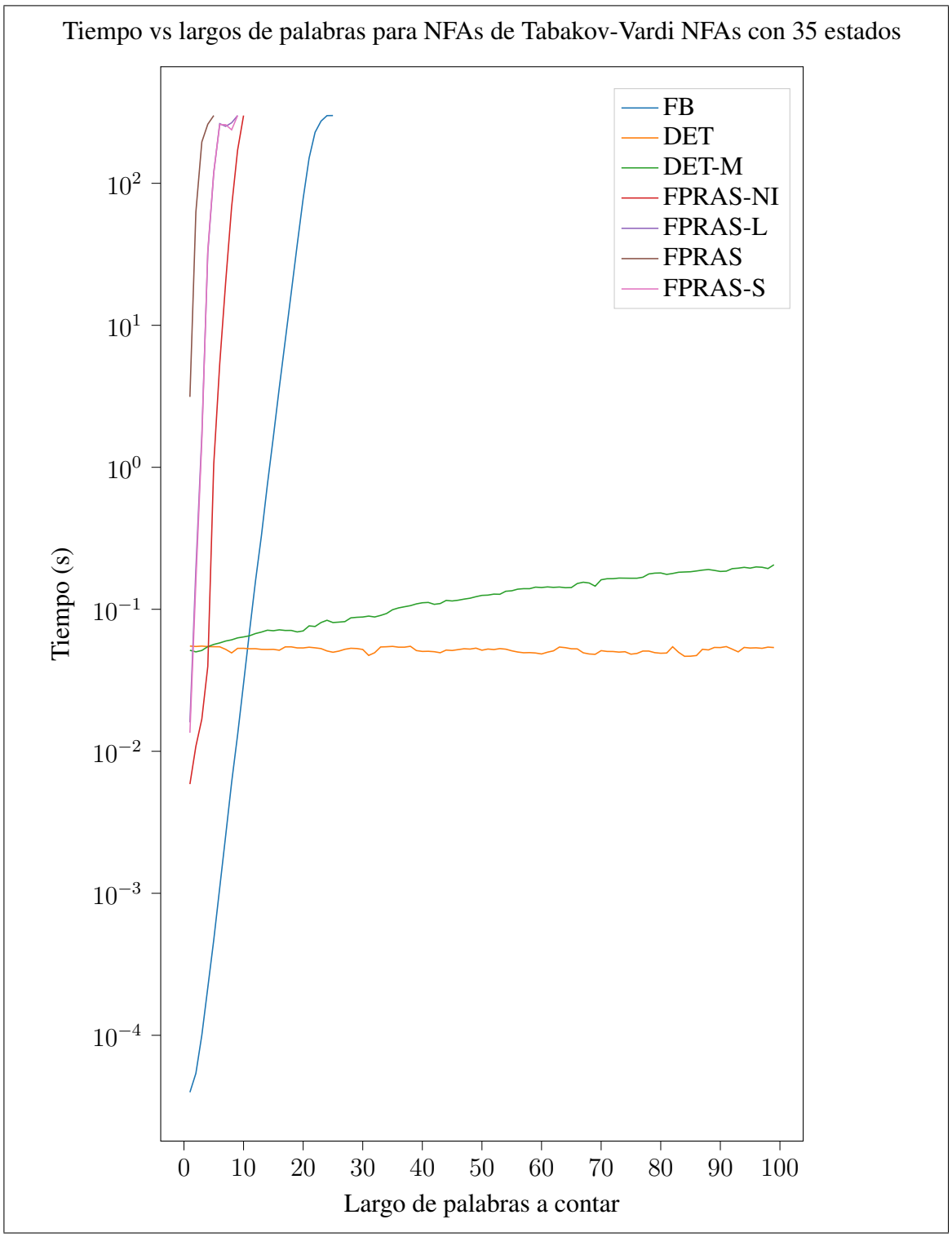


Figura A.34. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 35 estados.

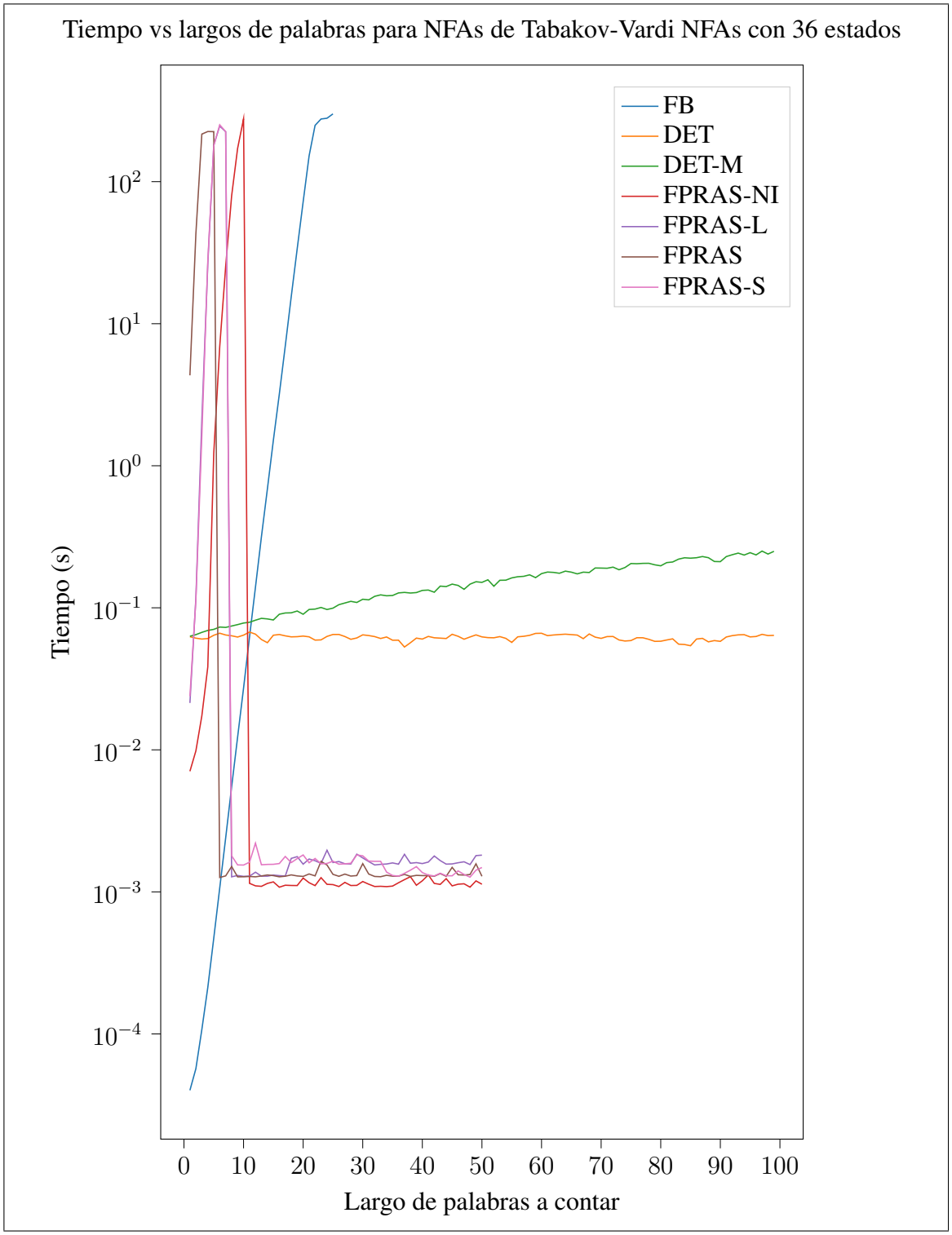


Figura A.35. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 36 estados.

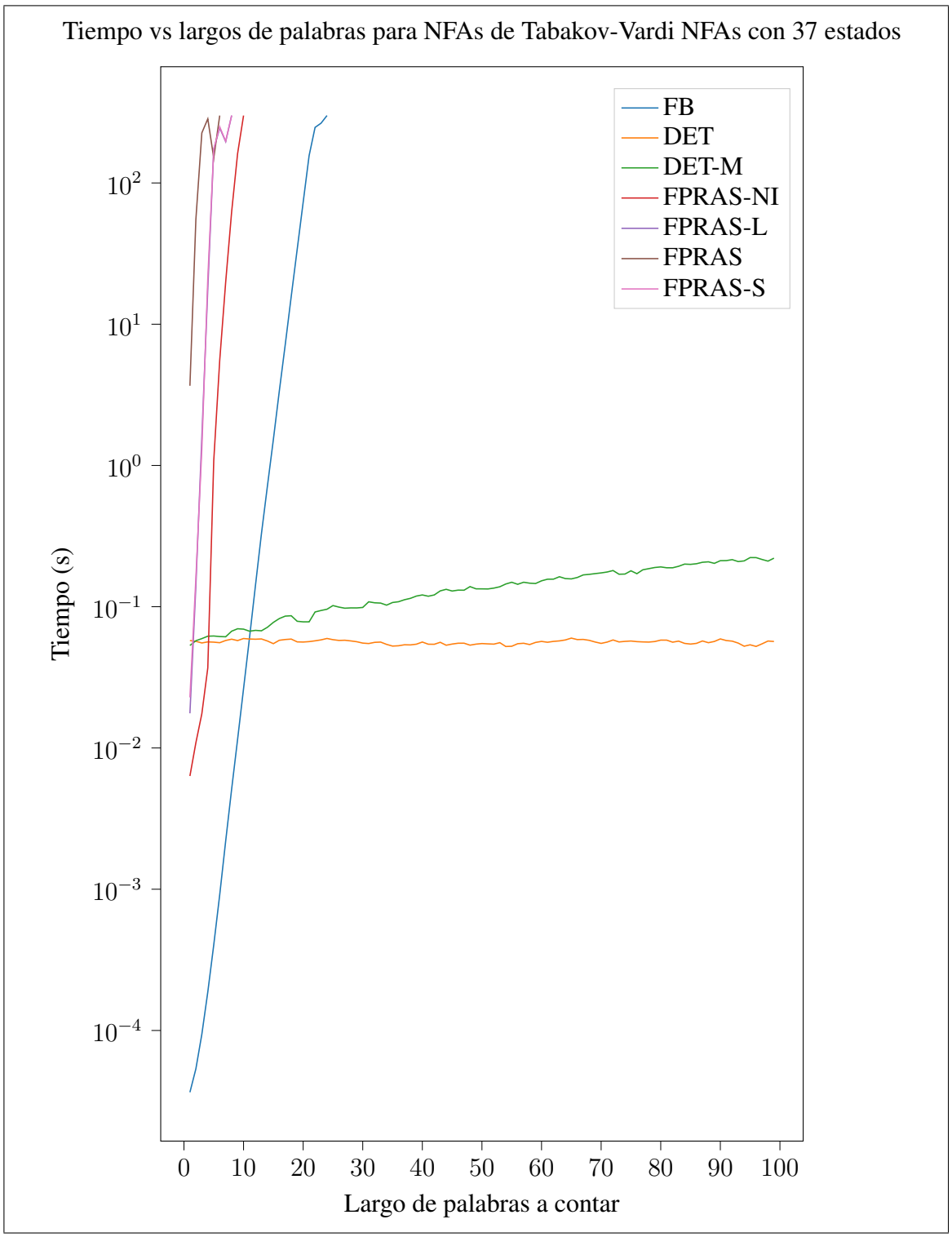


Figura A.36. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 37 estados.

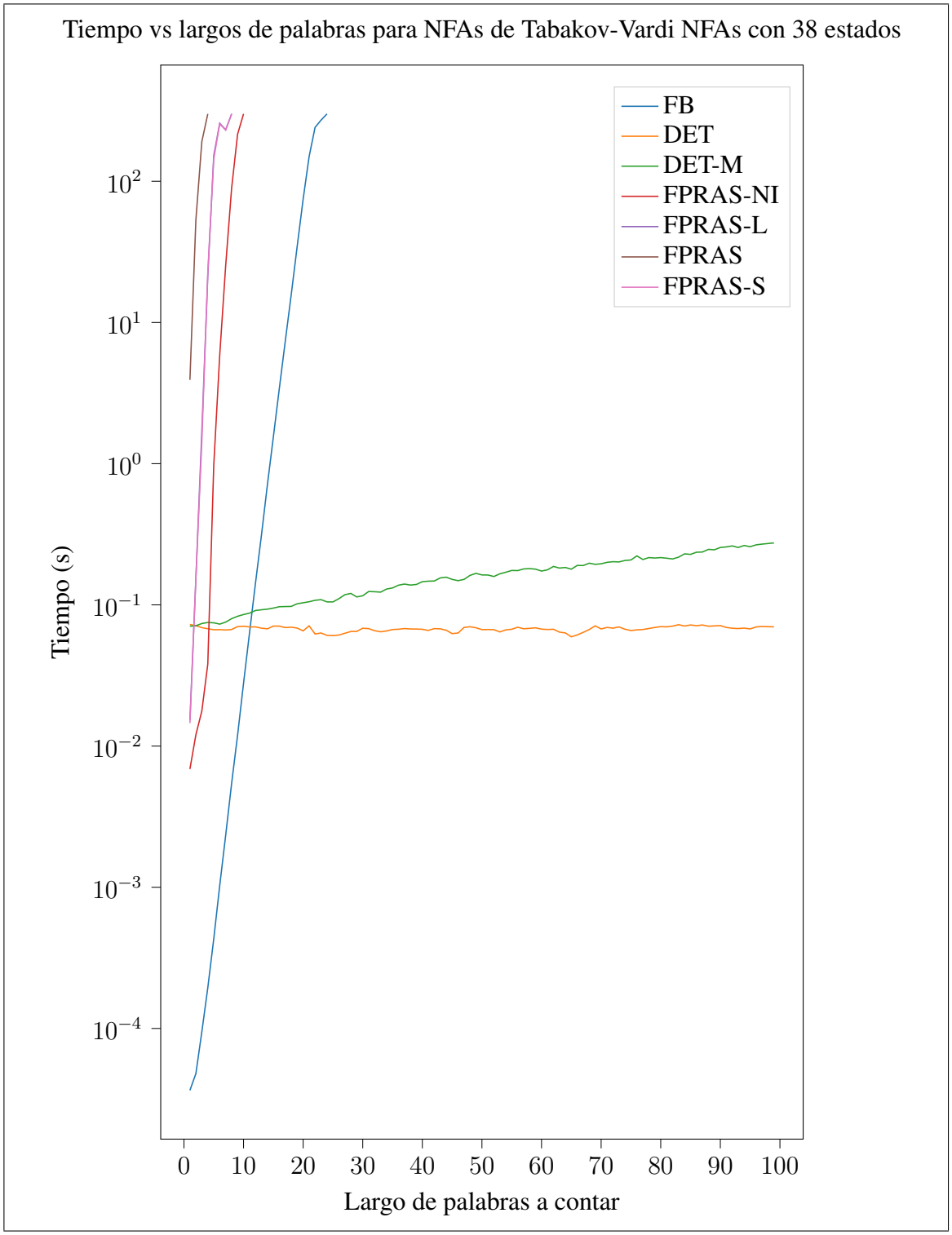


Figura A.37. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 38 estados.

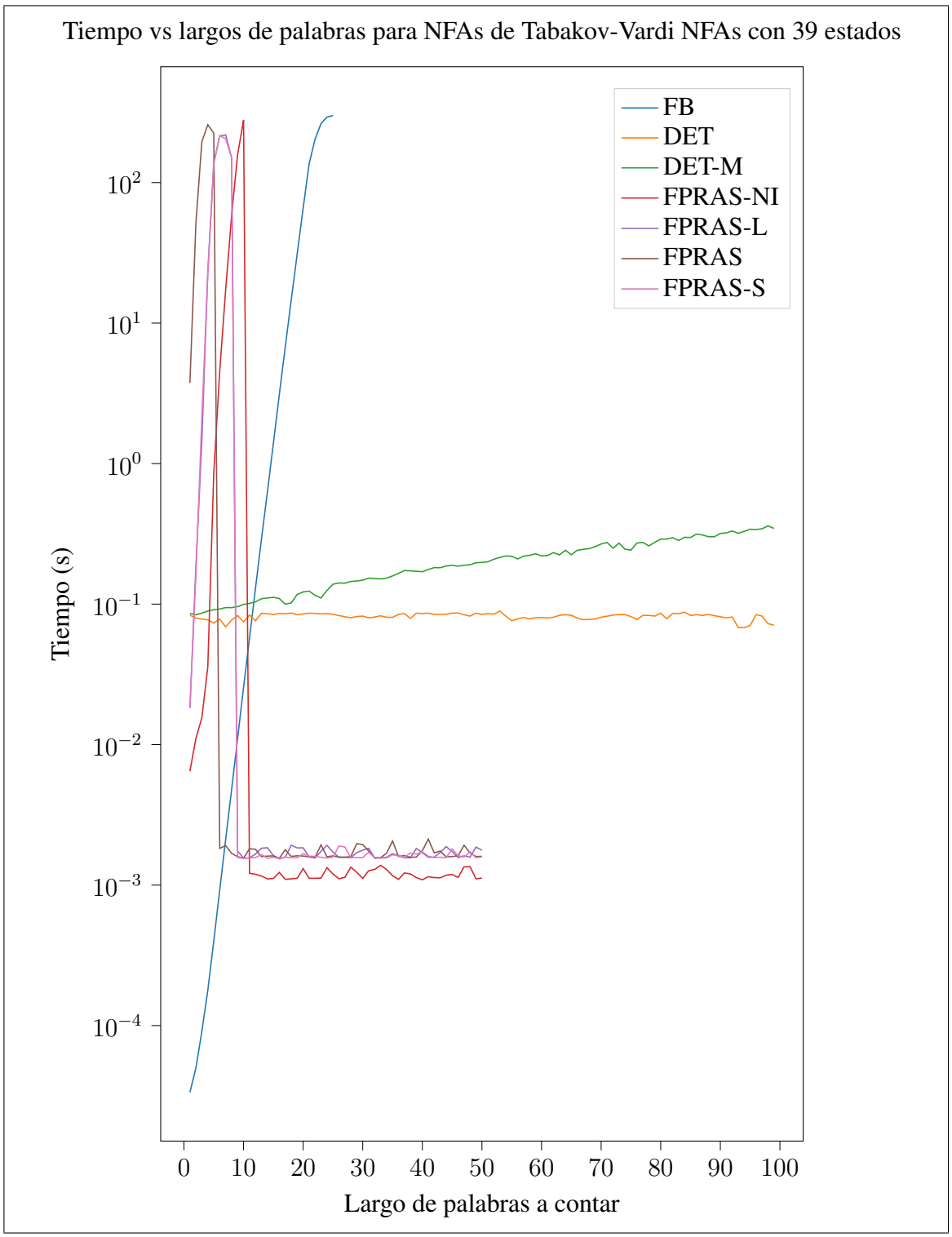


Figura A.38. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 39 estados.

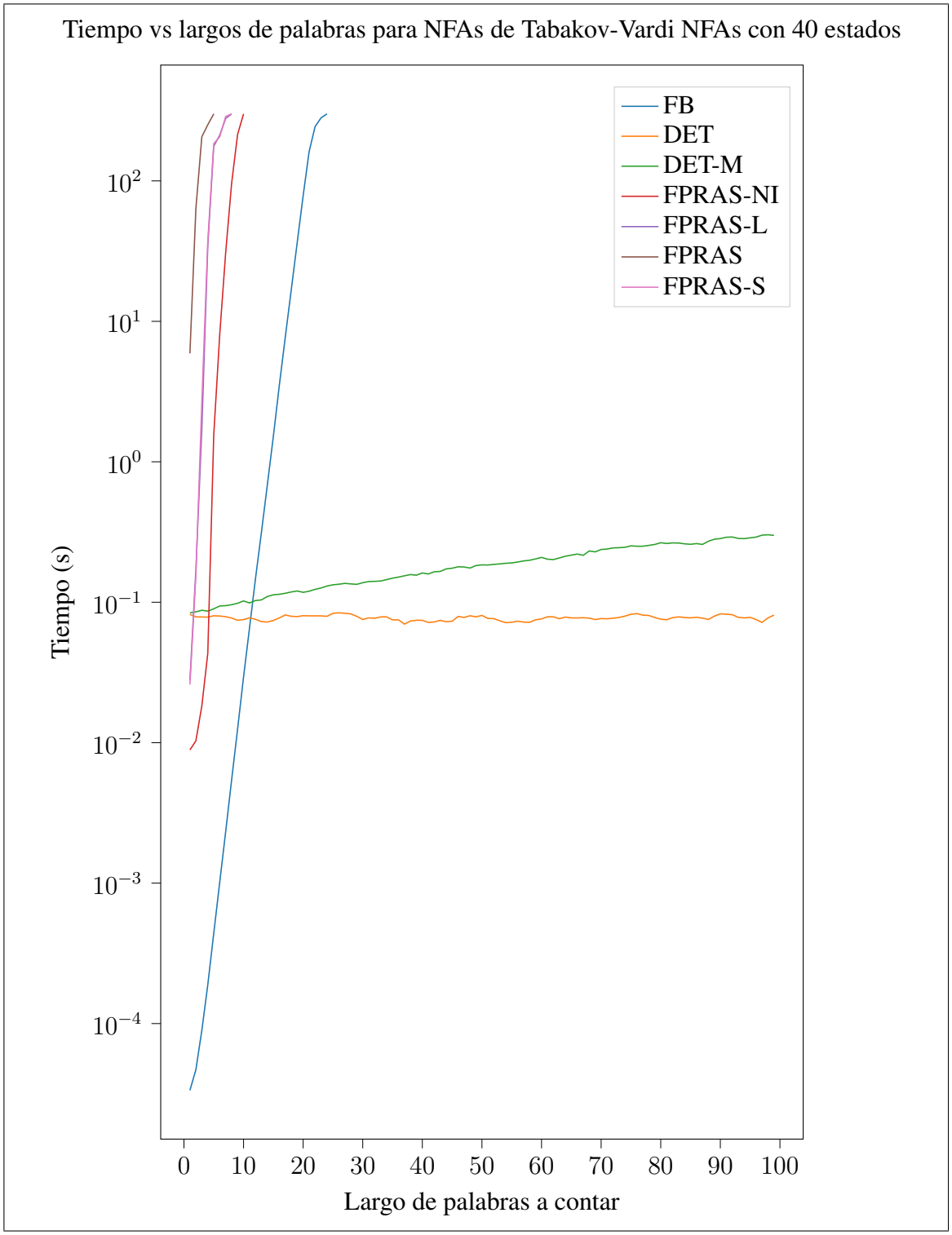


Figura A.39. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 40 estados.

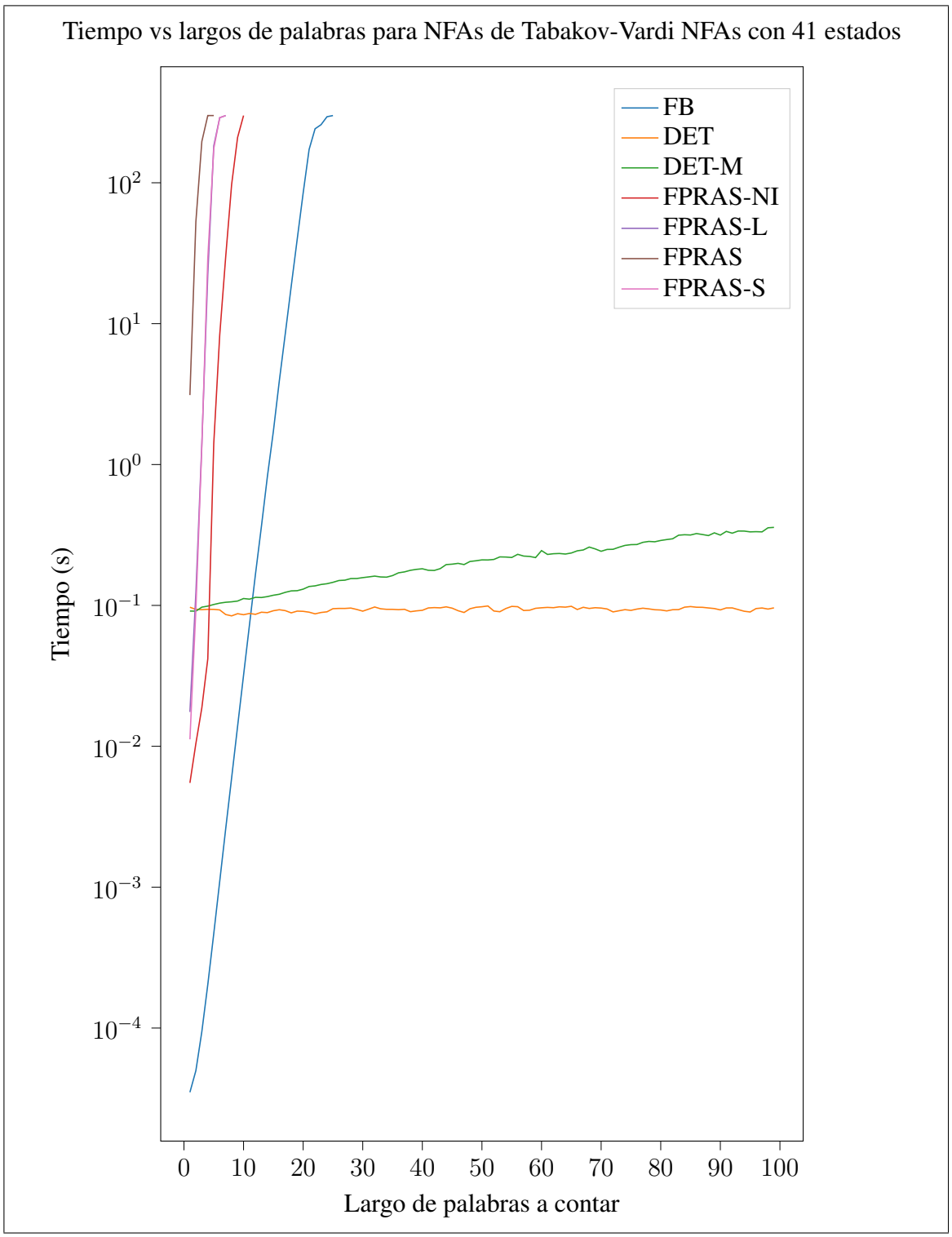


Figura A.40. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 41 estados.

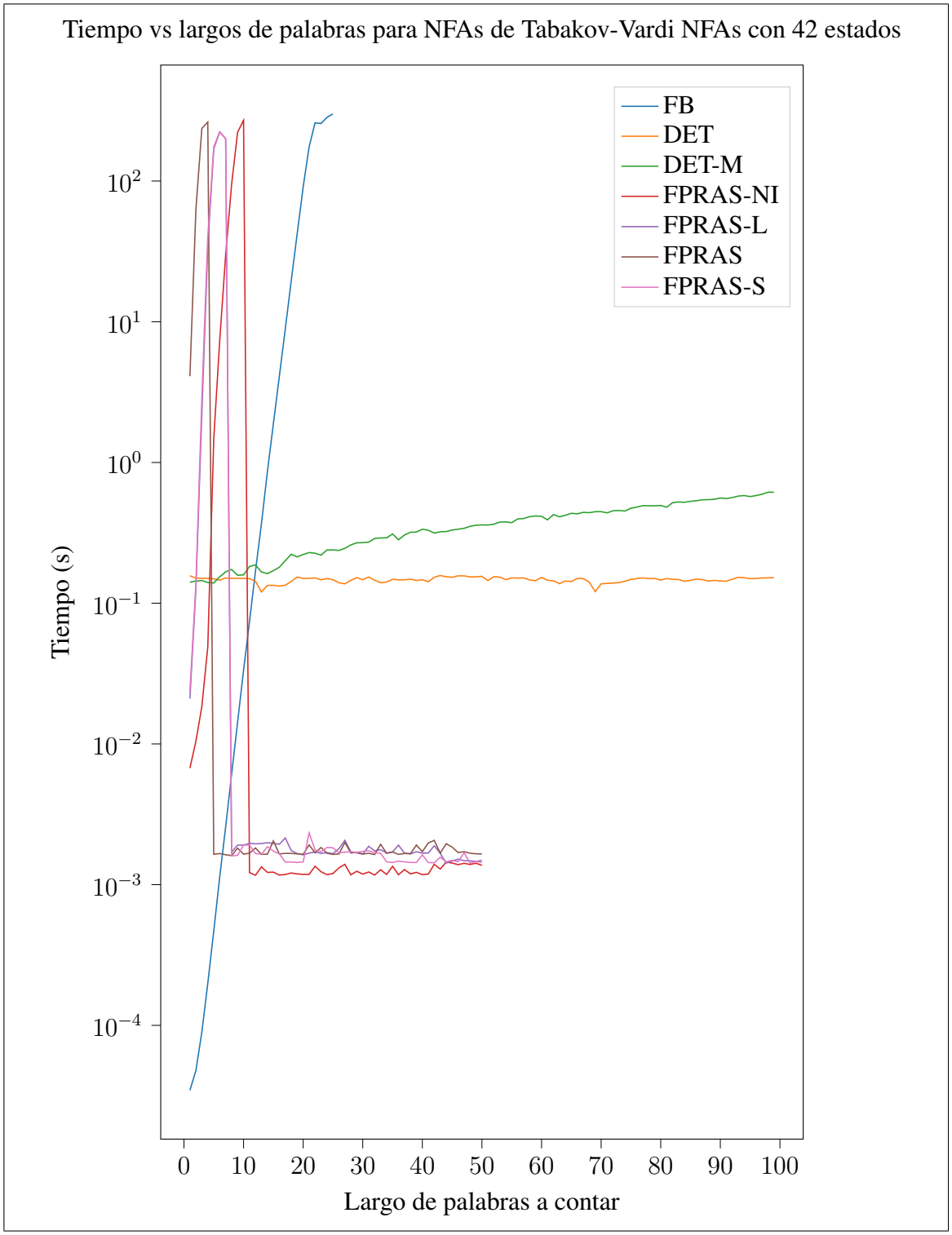


Figura A.41. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 42 estados.

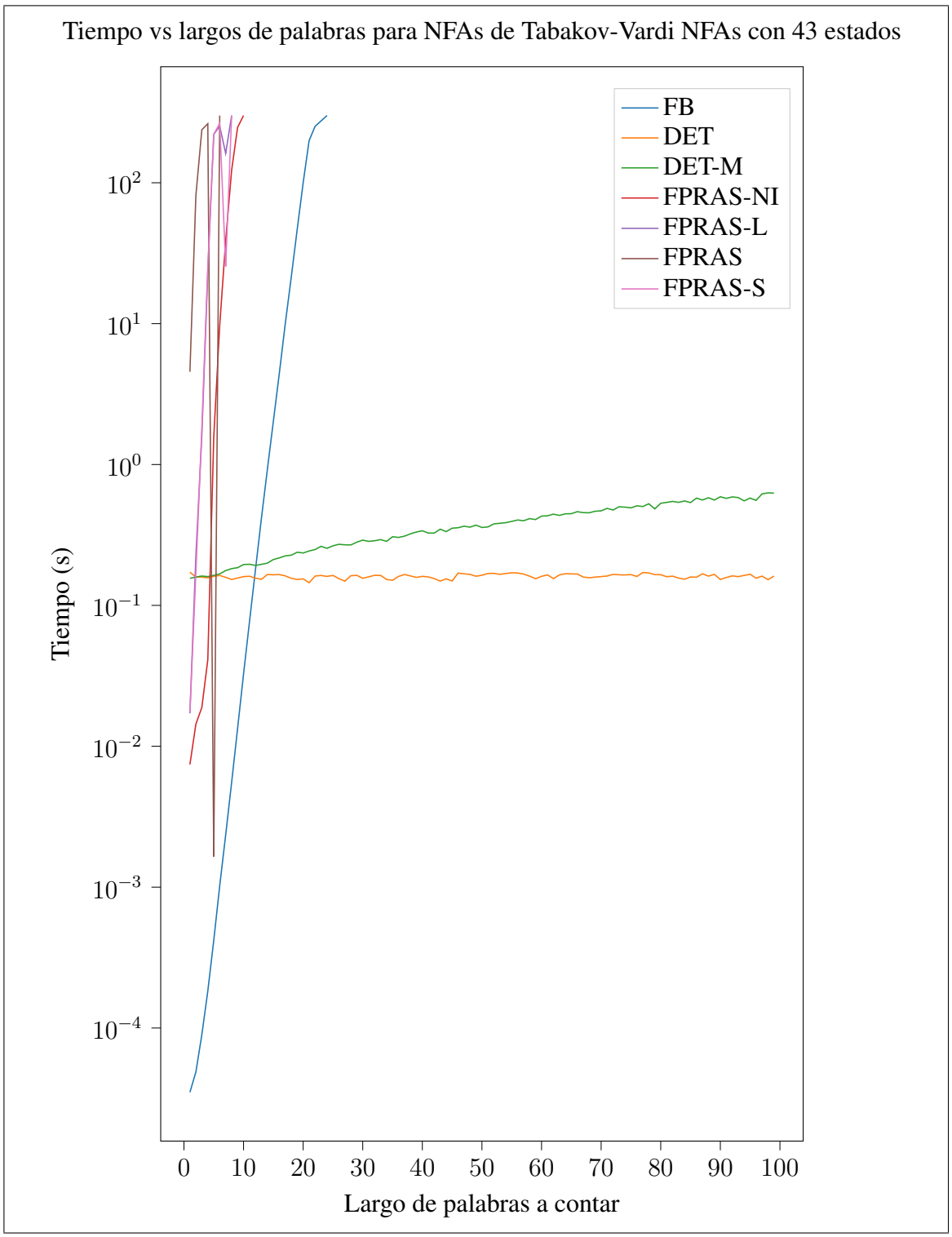


Figura A.42. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 43 estados.

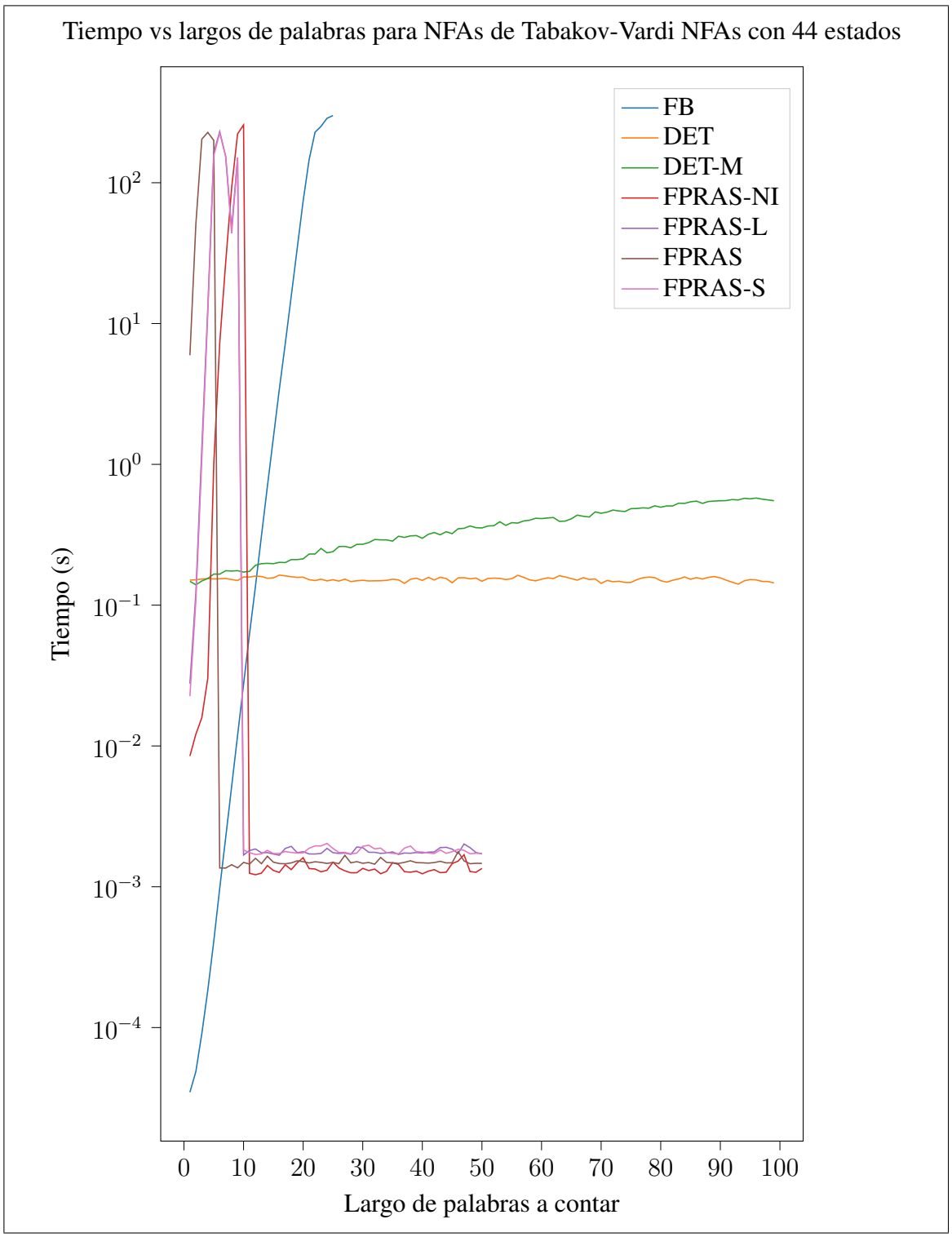


Figura A.43. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 44 estados.

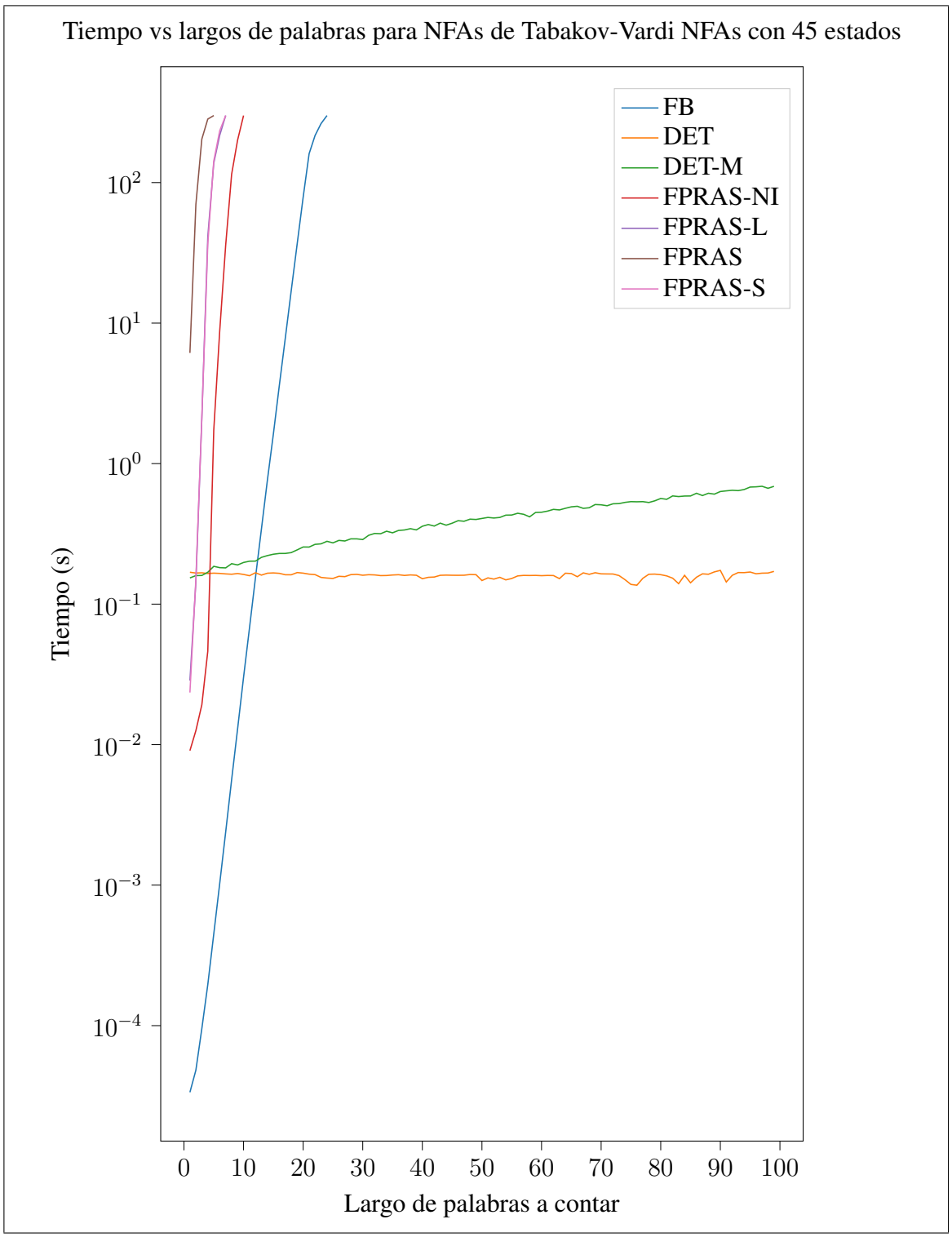


Figura A.44. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 45 estados.

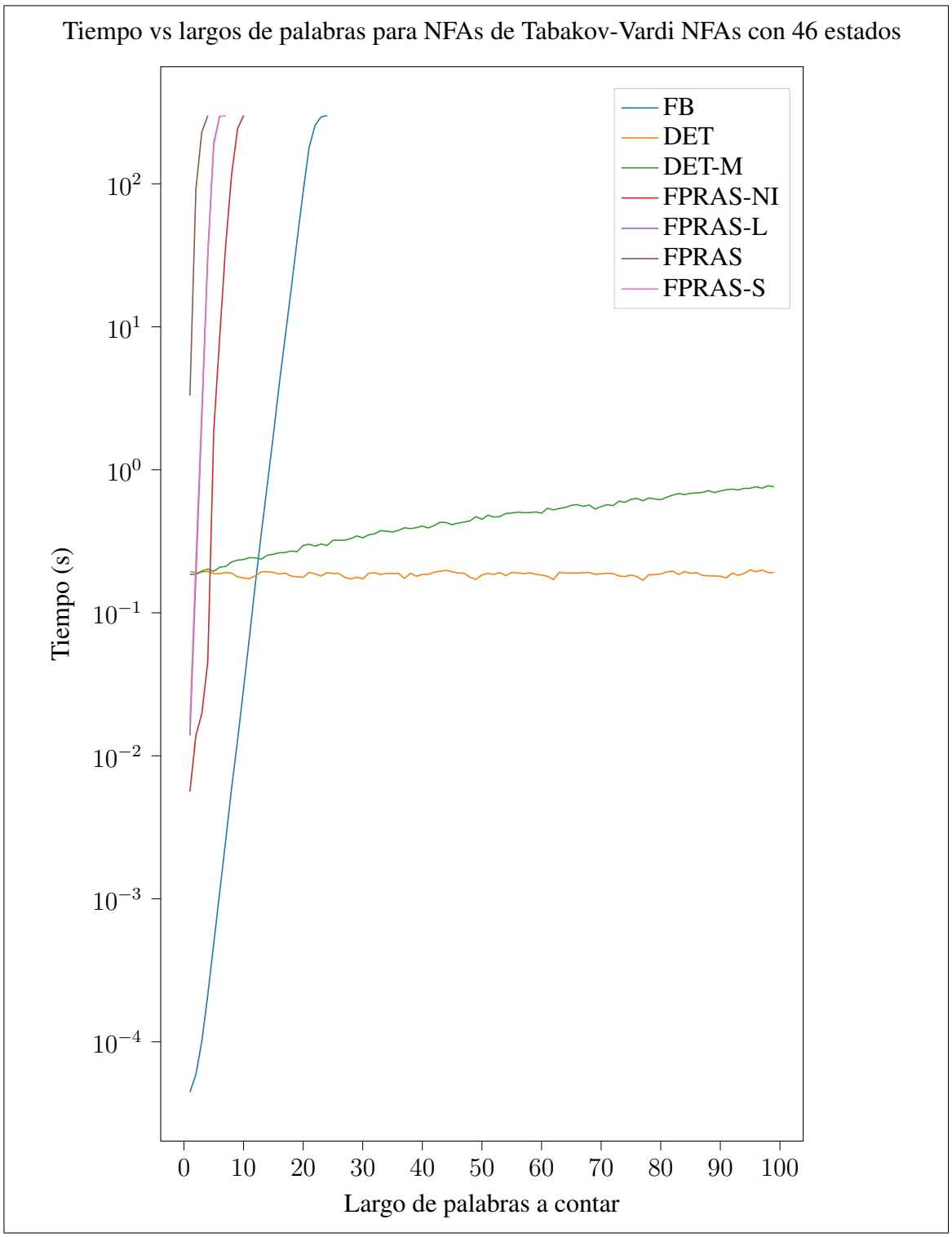


Figura A.45. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 46 estados.

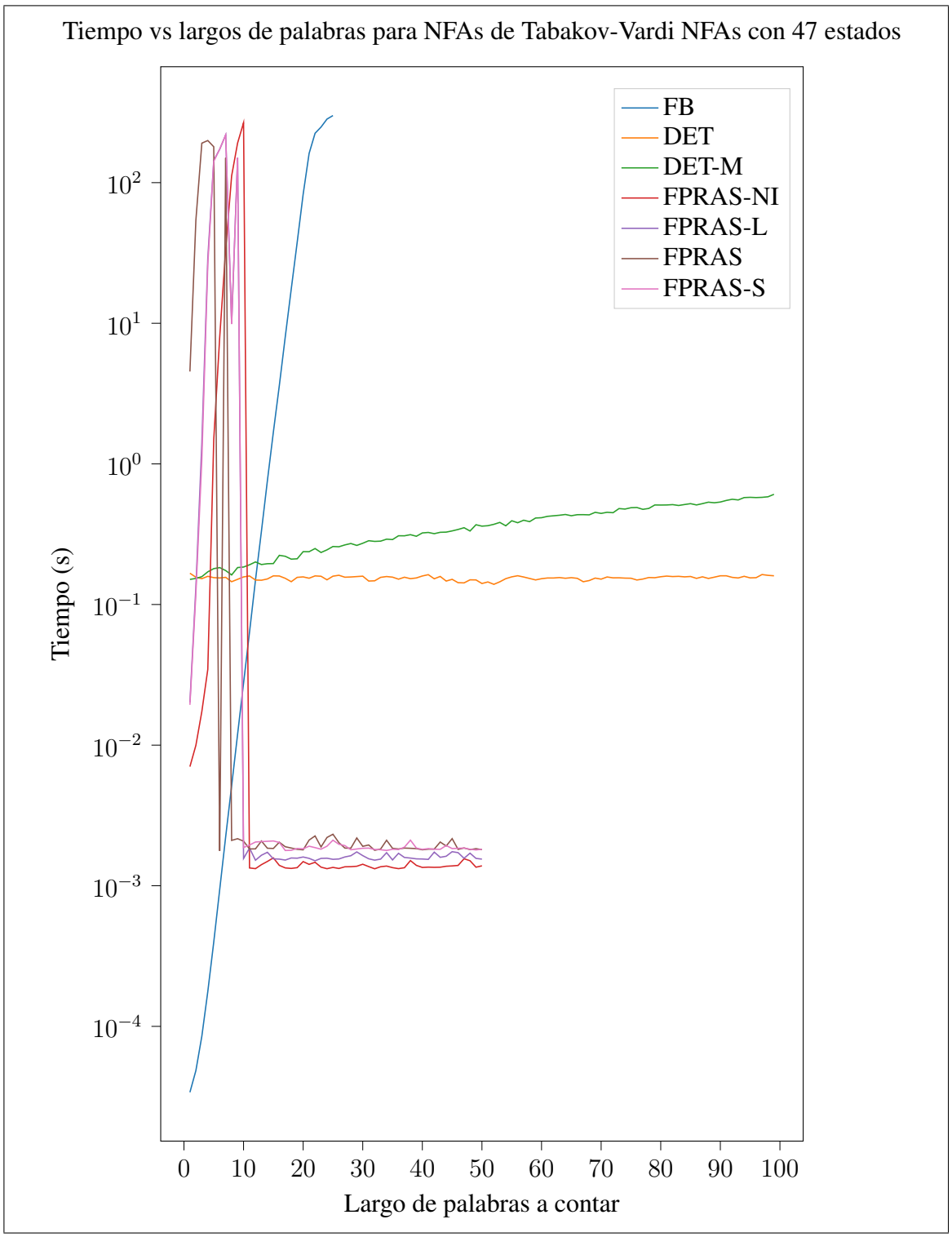


Figura A.46. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 47 estados.

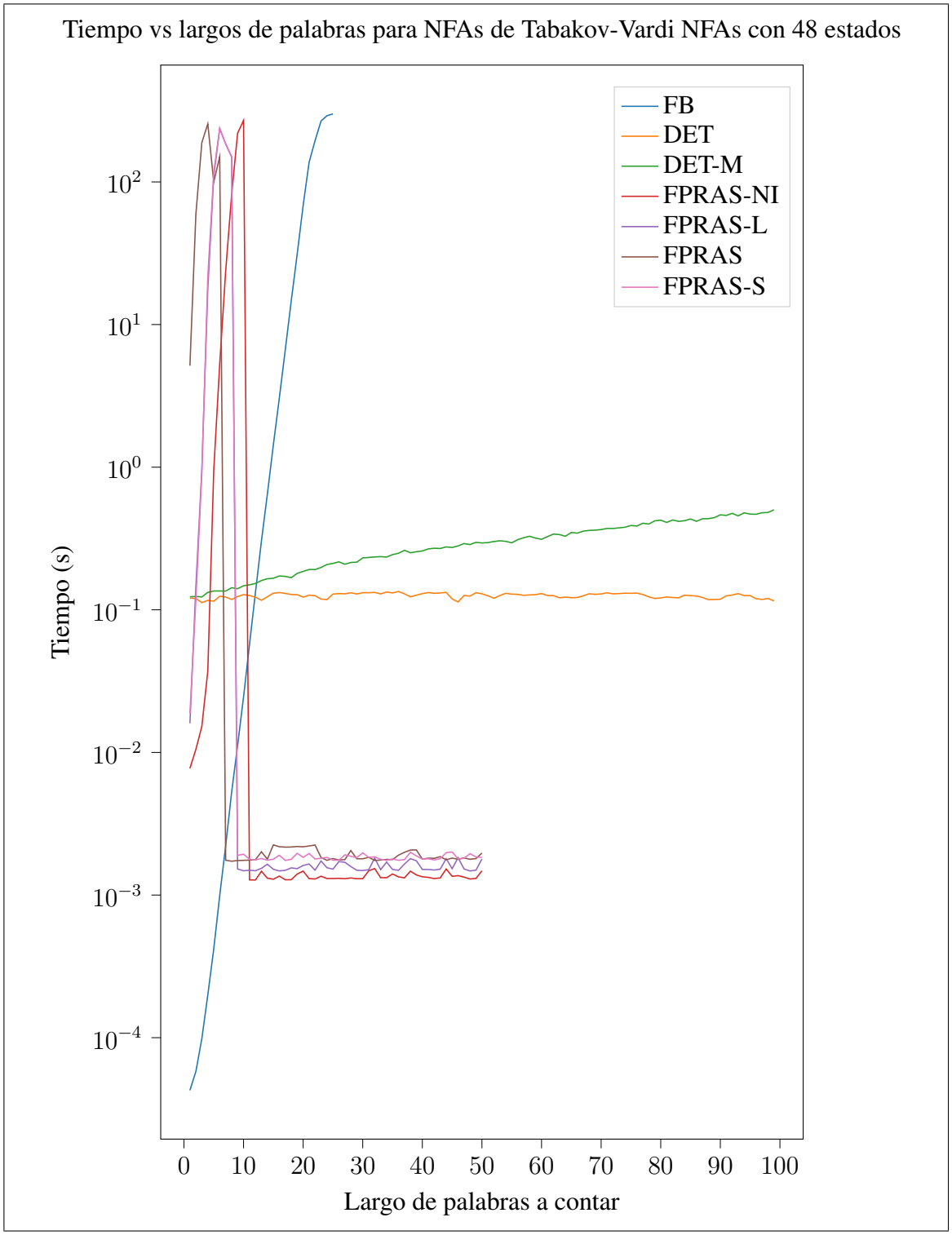


Figura A.47. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 48 estados.

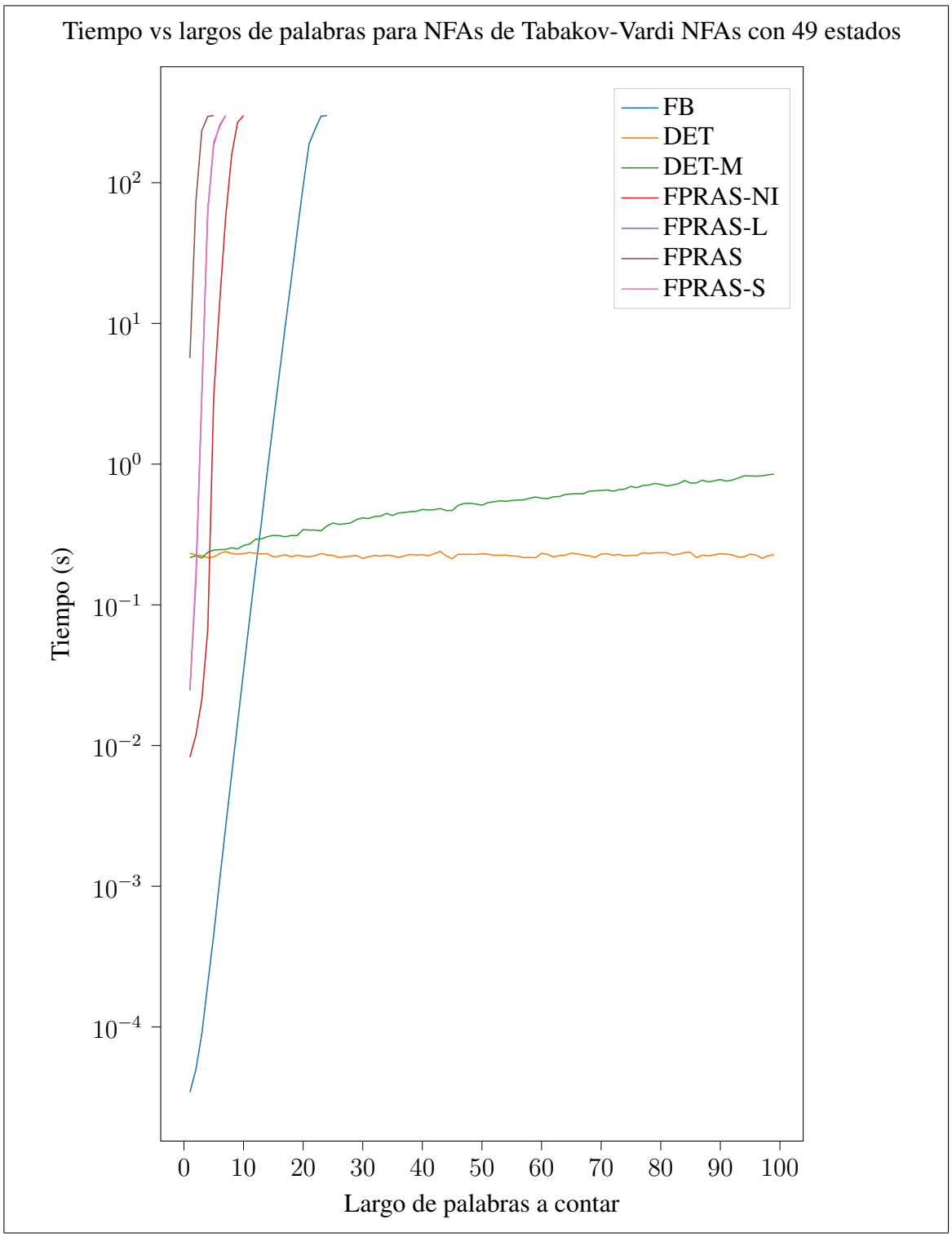


Figura A.48. Tiempos de ejecución promedio para NFAs de Tabakov-Vardi de 49 estados.