DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 11 No. Especial 2 (2023) 7–15

Metodología para la construcción de generadores de números pseudo-aleatorios utilizando programación genética
Methodology for the construction of pseudo random number generators using genetic programming

Ismael Rojas-Montes [a], Ariel Cavazos-Amador [a], Abraham Flores-Vergara [b,*], Eddie Clemente [b]

[a] *Maestría en Ciencias en Ingeniería Mecatrónica, TecNM/IT Ensenada, Blvd. Tecnológico No. 150, Ex-ejido Chapultepec, 22780, Ensenada, B. C., México.*
[b] *Departamento de Sistemas Computacionales, TecNM/IT Ensenada, Blvd. Tecnológico No. 150, Ex-ejido Chapultepec, 22780, Ensenada, B. C., México.*

Resumen

Los generadores de números pseudoaleatorios (PRNG, por sus siglas en inglés), se utilizan comúnmente en la informática para simular eventos aleatorios en aplicaciones como juegos, simulaciones, análisis estadístico y criptografía. En este trabajo se presenta una metodología para proponer PRNGs de forma indirecta, utilizando la encriptación de una imagen. Los PRNGs propuestos se construyen de forma automática utilizando programación genética. El rendimiento de los PRNGs propuestos, superan a PRNGs compuestos por sistemas caóticos y se validan utilizando el conjunto de pruebas estadísticas NIST 800-22 Rev. 1a., alcanzando rendimientos arriba del 99.46 %.

Palabras Clave: Generador de PRNG, Programación Genética, Criptografía de imágenes.

Abstract

Pseudo-Random Number Generators (PRNGs) are commonly used in computing to simulate random events in applications such as games, simulations, statistical analysis, and cryptography. This paper presents a methodology to propose PRNGs indirectly, using image coding. The proposed PRNGs are built automatically using genetic programming. The performance of the proposed PRNGs outperforms PRNGs composed of chaotic systems and is validated using the NIST 800-22 Rev. 1a statistical test set, reaching performances above 99.46 %.

Keywords: PRNG Generator, Genetic Programming, Image encryption.

1. Introducción

Un generador de números pseudoaleatorios (PRNG por sus siglas en ingles), es un algoritmo utilizado para generar una secuencia de números con un comportamiento parecido al aleatorio. Debido a que el PRNG está definido por un algoritmo determinístico, se dice que los números generados son pseudoaleatorios. Los algoritmos matemáticos para construir un PRNG, generalmente toman como base una condición inicial como entrada para generar la secuencia numérica Los PRNG se utilizan en aplicaciones tales como: sistemas de pruebas de software, sistemas de juegos de azar, para simulación de eventos en videojuegos, entre otros Kietzmann *et al.* (2021); James (1990). Actualmente, debido a la proliferación y crecimiento en el uso de dispositivos de telecomunicaciones, así como dispositivos conectados a internet, los PRNG han tenido su mayor aplicación en sistemas criptográficos modernos Sathya *et al.* (2021). Lo anterior para garantizar la segura transmisión y almacenamiento de información de carácter confidencial por los canales de comunicación.

En la literatura se encuentran reportados diversos métodos para la construcción de PRNG incluyendo: los métodos con algoritmos matemáticos basados en congruencias lineales Moreno (2015); Bhowmik *et al.* (2021), métodos basados en sistemas de ecuaciones caóticas Yuan *et al.* (2011); Zgliczynski (1997); Si *et al.* (2022), métodos no determinísticos basados en programación cuántica Akhshani *et al.* (2014); Gnatyuk *et al.* (2020), métodos basados en programación en ADN Tornea (2013) y métodos basados en programación genética Hernandez *et al.* (2004); Kösemen *et al.* (2018); Lamenca-Martinez

*Autor para correspondencia: aflores@ite.edu.mx
Correo electrónico: alm17280668@ite.edu.mx (Ismael Rojas-Montes), al16760481@ite.edu.mx (Ariel Cavazos-Amador), aflores@ite.edu.mx (Abraham Flores-Vergara), eclemente@ite.edu.mx (Eddie Clemente).

Historial del manuscrito: recibido el 28/04/2023, última versión-revisada recibida el 02/06/2023, aceptado el 06/06/2023 publicado el 11/09/2023. **DOI:** https://doi.org/10.29057/icbi.v11iEspecial2.10815

et al. (2006) entre otros Naik y Singh (2022); Senkerik *et al.* (2017); Sayed y Radwan (2020); cada uno con sus propias ventajas y limitaciones.

En cuanto a las aplicaciones de PRNG en la criptografía, los métodos que implementan algoritmos matemáticos comunes, tienen su principal desventaja en que son altamente predictibles, lo cual no garantiza fiabilidad para una comunicación segura. Los métodos que implementan sistemas de ecuaciones caóticas, han demostrado ser potencialmente viables en cuanto a seguridad criptográfica. Sin embargo, se requiere de un mayor recurso computacional para su implementación y el comportamiento caótico se mantiene en un espacio definido llamado atractor caótico, si se pierde el control del comportamiento caótico, se compromete la seguridad en el sistema criptográfico.

Los métodos basados en programación cuántica, utilizan principios como la superposición, el teletransporte y el entrelazamiento; tienen la ventaja de que son muy seguros en comparación con los generadores de PRNG clásicos, ya que su comportamiento está basado en principios físicos y no en algoritmos determinísticos. En la actualidad, la programación cuántica se encuentra aún en fase de desarrollo tanto en software como en hardware, aunado a ello, los algoritmos son intrínsecamente complejos y difíciles de diseñar.

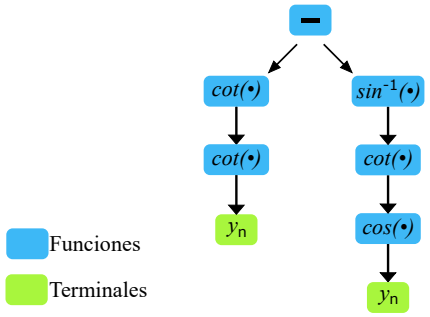
Los métodos que implementan programación en ADN utilizan los principios de la biología molecular para generar secuencias de números aleatorios. Los PRNG con programación ADN tienen la ventaja de que son muy compactos con un alto nivel de almacenamiento para grandes cantidades de información; sin embargo, las síntesis de moléculas de ADN actualmente es un proceso muy costoso y con un alto grado de complejidad, limitando su uso en aplicaciones prácticas.

Las técnicas de diseño de PRNG utilizando programación genética están basadas en la evolución biológica para generar la secuencia numérica incluyendo algoritmos genéticos, redes neuronales, sistemas caóticos o gramáticas formales, entre otros. Fundamentalmente, se crea una población inicial de cadenas de ADN que representa posibles soluciones; posteriormente, se aplican operaciones de selección, cruzamiento y mutación con el propósito de evolucionar para finalmente seleccionar la cadena de ADN que represente la mejor solución.

El presente trabajo de investigación, propone un método para construir múltiples PRNG con base en las técnicas de diseño de programación genética y utilizando algoritmos genéticos; lo anterior, considerando su aplicación en métodos de criptografía moderna, en específico, la criptografía de imágenes. Primeramente, se describe la metodología para la construcción de un generador de múltiples PRNG. En seguida, se presenta el análisis de viabilidad de los PRNG obtenidos, utilizando el conjunto de pruebas estadísticas NIST 800-22 Rev. 1a para la validación de las series numéricas pseudoaleatorias generadas por los PRNG; lo anterior, con el propósito de seleccionar los mejores PRNG obtenidos. Posteriormente, se presentan los resultados considerando los 10 mejores PRNG y su aplicación en el encriptado de imágenes en escala de grises, incluyendo la validación de los criptogramas obtenidos mediante su valor de entropía de la información. Finalmente se presentan las conclusiones y trabajo futuro.

2. Metodología

La Programación Genética (GP) es una técnica dentro del cómputo evolutivo que se utiliza para la optimización y aprendizaje de computadora. La GP sigue una analogía con la teoría NeoDarwinista de la evolución natural. En esta analogía un problema de búsqueda representa un medio ambiente, cada posible solución al problema es un individuo que interactúa en ese ambiente, así, un como un conjunto de individuos forman una población; un conjunto de posibles soluciones formarán también una población. Una característica de la GP es la representación de cada posible solución, esta se define por medio de un árbol sintáctico, ver figura 1, lo que permite codificar una expresión matemática como una estructura computacional.



$\mathbf{PRNG}_{gp}=cot(cot(y_n))\text{-}sin^{-1}(cot(cos(y_n)))$

Figura 1: Una función analítica puede ser representada computacionalmente como un árbol sintáctico, en este caso el GP, utiliza esta representación para generar los PRNG a partir de una combinación de funciones y terminales dadas.

Bajo estos aspectos, el ciclo evolutivo se describe como sigue:

1. Se crea una población inicial de posibles soluciones de forma aleatoria.
2. Se evalúa la aptitud de cada posible solución bajo un índice de rendimiento de acuerdo a la problemática definida.
3. Ponderando su rendimiento, se selecciona un conjunto de soluciones.
4. Al conjunto seleccionado, se le aplican operadores de recombinación y mutación para generar un nuevo conjunto (descendencia) de posibles soluciones.
5. Dado el conjunto original de posibles soluciones y el conjunto descendencia, se seleccionan las mejores soluciones, para crear una nueva población o generación.
6. Si no se ha alcanzado satisfacer alguna condición de paro, como un número máximo de generaciones, la solución del problema, o un error de tolerancia para la solución, regresar al paso 2.

El espacio de búsqueda en un problema de optimización es el conjunto de todas las posibles soluciones que pueden ser evaluadas para encontrar la mejor solución. En este caso, es el espacio de todas las posibles combinaciones de funciones y variables, que pudieran construir un PRNG. En la GP el espacio de búsqueda se determina por el conjunto de funciones y el conjunto de terminales que pueden construir un árbol sintáctico. Nótese que la diferencia entre una función y una terminal solo

Tabla 1: Funciones utilizadas en el proceso evolutivo.

No	Expresión	Descripción	No	Expresión	Descripción
1	$(\cdot)^2$	Cuadrática	20	sec(·)	Secante
2	$(\cdot)^3$	Cúbica	21	cot(·)	Cotangente
3	$\sqrt{\cdot}$	Raíz cuadrada	22	asin(·)	Arcoseno
4	asinh(·)	Inv. seno hiperbólico	23	acos(·)	Arcocoseno
5	acosh(·)	Inv. coseno hiperbólico	24	atanr(·)	Parte real del Arcotangente
6	atanh(·)	Inv. tangente hiperbólico	25	$e^{(\cdot)}$	Exponencial
7	acsch(·)	Inv. cosecante hiperbólico	26	ln(·)	Logaritmo natural
8	asech(·)	Inv. secante hiperbólico	27	Re(·)	Parte real del argumento
9	acoth(·)	Inv. cotangente hiperbólico	28	$\|\cdot\|$	Norma Euclidiana
10	sinh(·)	Seno Hiperbólico	29	$\|\cdot\|$	Valor Absoluto
11	cosh(·)	Coseno hiperbólico	30	sign(·)	Signum
12	tanh(·)	Tangente hiperbólica	31	+	Adición
13	csch(·)	Cosecante hiperbólico	32	−	Substracción
14	sech(·)	Secante hiperbólico	33	/	División
15	coth(·)	Cotangente hiperbólico	34	∗	Producto
16	cos(·)	Coseno	35	$(\cdot)^{(\cdot)}$	Exponenciación
17	sin(·)	Seno	36	máx(·)	Máximo
18	tan(·)	Tangente	37	mín(·)	Mínimo
19	csc(·)	Cosecante			

depende de su aridad o número de argumentos. Así, una función es un operador de más de un argumento o aridad mayor o igual a dos, mientras que una terminal se considera como un operador de un solo argumento o aridad igual a 1. En este caso, las funciones seleccionadas son operadores aritméticos, de exponenciación, trigonométricos, hiperbólicos, logarítmicos, y funciones especiales como el *máximo* y el *mínimo* de dos argumentos, la función *signum* y la parte real de un argumento complejo, ver tabla 1.

Las terminales se eligieron bajo el dominio del problema, en este caso es la dependencia con el valor actual, valores anteriores de una función dada, ver tabla 2, más algunas constantes.

Tabla 2: Terminales utilizadas en el proceso evolutivo

No	Term	Descripción	No	Term	Descripción
1	y_n	Valor actual	2	y_{n-1}	Valor anterior
3	1	Constante	4	0.5	Constante
5	2	Constante			

2.1. *Función de aptitud*

La función de aptitud mide la bondad de una posible solución. Así, esta función asigna un índice que relaciona la posible solución con la calidad de la misma. En este caso, se ha propuesto una medida indirecta, donde la calidad del PRNG se mide sobre una tarea de encriptación de una imagen de $N \times N$. Este proceso sigue los siguientes pasos:

- Dada una posible función PRNG y_{n+1}, se genera una serie de números de tamaño $N \times N$ y se considera solo la parte real de cada elemento.
- Cada elemento se multiplica por 1×10^9, con el fin de tomar en cuenta una precisión de 9 dígitos después del punto decimal.
- La serie cifrante se obtiene considerando el residuo de la división del resultado anterior y 255.
- Dada la serie cifrante, se aplica una operación XOR con la imagen a codificar, y el resultado de esta operación es un criptograma de la imagen, ver figura 2.
- Un índice de la aleatoriedad de los datos del criptograma es la entropía, esta se calcula como:

$$S_I = -\sum_{i=0}^{i=255} p_i \log_2(p_i)$$

  donde, p_i es la cuenta de cada uno de los elementos del histograma normalizado de la imagen.
- Finalmente la función de aptitud está definida como:

$$Apt_i = e^{S_I} \tag{1}$$

La función exponencial en la función de aptitud, se propone para tener una mejor dispersión de los datos y que el proceso de selección dentro del ciclo evolutivo sea eficiente.

(a) Imagen tomada para el aprendizaje de PRNGs. (b) Ejemplo de un criptograma resultante.

Figura 2: Ejemplo de un criptograma obtenido para medir su entropía y aptitud.

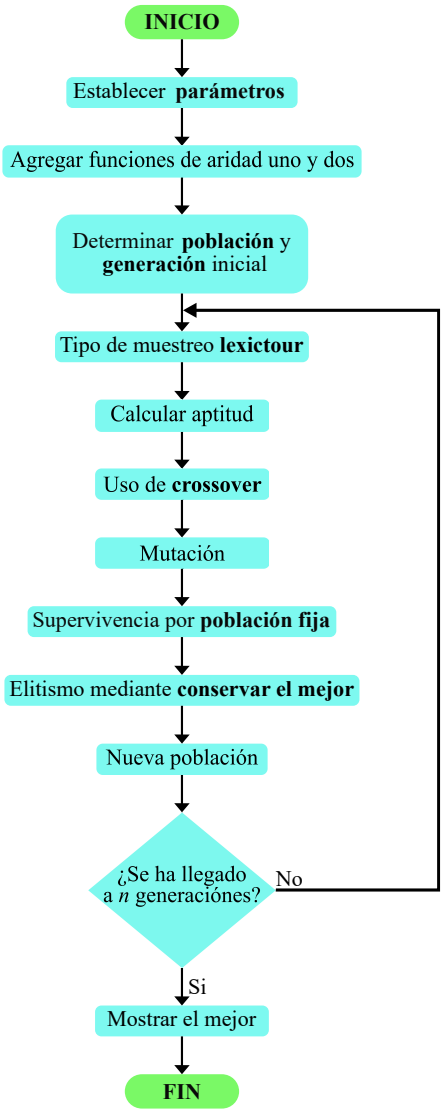


Figura 3: Diagrama de flujo mostrando el proceso del GP para la generación de las funciones.

Una vez definido el espacio de búsqueda y la función aptitud, se utilizó la librería GPLAB en Matlab para la implementación del algoritmo de programación genética. Los parámetros que se tomaron fueron 100 generaciones y una población fija de 200 individuos por generación. El método de selección elegido fue por torneo lexictour, es decir, dados dos individuos con la misma aptitud se prefiere aquel que tenga el mínimo número de nodos. Se utilizó elitismo, con esta técnica se mantiene el mejor individuo de cada generación. La probabilidad de cruzamiento fue propuesta del 80 % y probabilidad de mutación se definió en 20 %. Se realizaron 30 corridas para verificar la metodología propuesta. La figura 3 muestra un diagrama de flujo del procesos evolutivo implementado. En este caso, el objetivo de la optimización es encriptar una imagen a manera de mensaje, maximizando indirectamente la entropía del criptograma resultante. Para este ejercicio, se considera como mensaje inicial de prueba, una imagen en escala de grises con una resolución de 250x250 píxeles llamada Lena.png. Posteriormente, se implementa el método en mensajes de mayor tamaño, con imágenes a color de mayor resolución. Es importante mencionar que el tipo de imagen no es relevante, ya que finalmente, la imagen se considera como una secuencia o muestra de números que se desea encriptar; en este sentido, la hipótesis es que existe una función tal que, al aplicarse a dicha secuencia de números dados (la imagen), esta puede producir un criptograma con el mayor desorden posible. Lo anterior incluso incrementando el tamaño de la secuencia numérica y en donde esta función sea un PRNG.

La figura 4 muestra el comportamiento de la mejor aptitud durante 30 generaciones. También se puede apreciar el promedio y la desviación estándar de la mejor aptitud a lo largo de cada generación. Note su convergencia promedio a una aptitud de 2977.1, lo que indica que la entropía del criptograma obtenido es de 7.9987 unidades, donde la máxima entropía en una imagen de 8 bits por píxel es de 8. De esta forma el mejor resultado tiene una aptitud de 2977.75 unidades, lo que equivale a una entropía en su criptograma de 7.9989 unidades.

En cada una de las corridas que se realizaron se obtuvieron valores cercanos a la entropía máxima de la imagen. Con lo cual, se realizó el siguiente análisis, se seleccionaron las soluciones con una aptitud mayor a 2977 unidades o 7.99 unidades de entropía, a lo largo de las 30 corridas, obteniendo un total de 136 funciones. Sobre este conjunto de soluciones se contabilizaron las funciones y terminales que las componen, ver figura 5; en este caso, se observó que las funciones de mayor frecuencia, quitando las funciones aritméticas, fueron el *coseno*, el *arcotangente* con argumento real, la *parte real* de un argumento y la función *máximo* de dos argumentos. De igual manera, la funciones menos utilizadas, debajo del 2 % de las soluciones, fueron las funciones: *raíz cuadrada*, *arcosecante hiperbólica*, *coseno hiperbólico*, *exponencial*, *signum*, y *potenciación*. Mientras las terminales más utilizadas fueron: y_n, y_{n-1} y la constante 2.

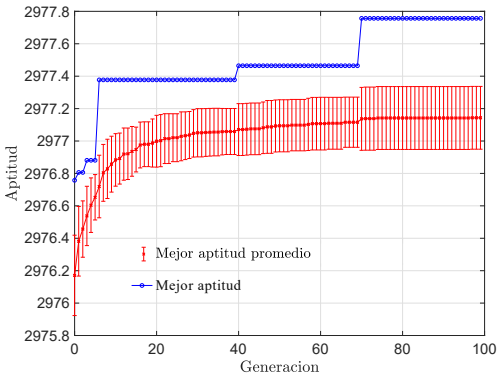
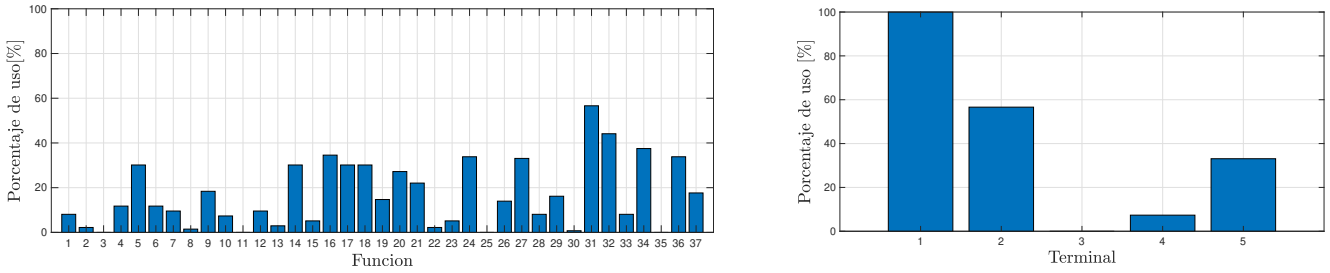


Figura 4: Mejor aptitud promedio durante 30 corridas.

3. Análisis de resultados

Para validar que las series numéricas generadas por los PRNG obtenidos con el método propuesto y que sean consideradas como series pseudoaleatorias, se utilizó la prueba estándar NIST 800-22 Rev. 1a. La prueba NIST para generadores de números pseudoaleatorios es un conjunto de 15 pruebas estadísticas desarrolladas por el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés) de los Estados Unidos. El propósito de estas pruebas es evaluar la calidad de los generadores de números pseudoaleatorios (PRNGs, por



(a) Porcentaje de uso de funciones

(b) Porcentaje de uso de terminales

Figura 5: Contabilización de las funciones y terminales de mayor uso en las soluciones con un desempeño mayor o igual a 7.99 unidades de entropía. El número de la función o terminal, corresponde la descrita en las tablas 1 y 2. Nóte que el 100 % de las soluciones analizadas están en función del valor actual, y_n, y solo alrededor del 58 % emplean el valor anterior, y_{n-1} de la función.

sus siglas en inglés) utilizados en aplicaciones criptográficas. La prueba NIST toman secuencias binarias generadas por un PRNG, y buscan evaluar propiedades como la uniformidad, la independencia y la complejidad. A continuación se describen brevemente cada una de las 15 pruebas NIST:

- Prueba de Frecuencia (Frequency Test, **F**): evalúa la proporción de unos y ceros en la secuencia.
- Prueba de Frecuencia de Bloques (Block Frequency Test, **BF**): evalúa la proporción de unos y ceros en bloques de tamaño fijo dentro de la secuencia.
- Prueba de Ejecución (Runs Test, **Rs**): busca patrones repetitivos en la secuencia.
- Prueba de Ejecución de Frecuencia de Bloques (Runs Distribution Test, **S**): busca patrones repetitivos en bloques de tamaño fijo dentro de la secuencia.
- Prueba del Test de la Serie (Test for the Longest Run of Ones in a Block, **LR**): busca la serie más larga de unos o ceros en bloques de tamaño fijo dentro de la secuencia.
- Prueba de la Aproximación Pi (Binary Matrix Rank Test, **Rk**): busca patrones en una matriz generada a partir de la secuencia.
- Prueba de la Varianza (Discrete Fourier Transform Test, **SD**): aplica una transformación de Fourier a la secuencia y evalúa la varianza de los coeficientes resultantes.
- Prueba de Frecuencia de Entropía (Non-overlapping Template Matching Test, **NT**): busca patrones específicos en la secuencia.
- Prueba de la Entropía de la Cadena (Overlapping Template Matching Test, **OT**): busca patrones específicos en bloques superpuestos dentro de la secuencia.
- Prueba del Test de Frecuencia Universal (Universal Statistical Test, **U**): evalúa la proporción de unos y ceros en la secuencia y los compara con una distribución teórica conocida.
- Prueba de la Compresibilidad (Approximate Entropy Test, **AE**): evalúa la complejidad de la secuencia mediante la comparación de la entropía aproximada de subsecuencias.
- Prueba de la Compresibilidad de Frecuencia de Bloques (Cumulative Sums Test, **CS**): evalúa la proporción de unos y ceros en bloques de tamaño variable dentro de la secuencia.
- Prueba del Test de Aleatoriedad Lineal (Random Excursions Test, **RE**): busca patrones en una caminata aleatoria generada a partir de la secuencia.
- Prueba del Test de Aleatoriedad Lineal de Frecuencia de Bloques (Random Excursions Variant Test, **RV**): busca patrones en una caminata aleatoria generada a partir de bloques de tamaño fijo dentro de la secuencia.
- Prueba del Test de Aleatoriedad Lineal Extendido (Linear Complexity Test, **LC**): evalúa la complejidad de la secuencia mediante la búsqueda de patrones en subsecuencias de longitud variable.

En estándar NIST 800-22 Rev.1a, establece que una secuencia numérica puede ser considerada como una secuencia pseudoaleatoria, si el valor de la proporción obtenida al realizar cada prueba es mayor a 0.9805. Las series numéricas obtenidas por los PRNG propuestos, deben superar cada prueba con un mínimo de 98.05 % de rendimiento para que puedan ser consideradas como series pseudoaleatorias. Al conjunto de soluciones que obtuvieron una aptitud mayor a 2977 unidades o 7.99 unidades de entropía, se le aplicaron el conjunto de pruebas NIST 800-22 Rev.1a, las 10 funciones que obtuvieron los mejores resultados se listan en la tabla 3.

Por otra partre la tabla 4 muestra que los PRNG propuestos por GP pasan las pruebas del estándar NIST 800-22 Rev.1a, superando el 99.46 % de rendimiento. Adicionalmente, se observa que los porcentajes de rendimiento de los PRNG obtenidos, son equiparables con algunos PRNG basados en sistemas caóticos documentados en la literatura y ampliamente utilizados en sistemas con criptografía caótica, tales como: el mapeo de Tinkerbell Yuan *et al.* (2011), el mapeo de Chen Zhao *et al.* (2023), el mapeo de Hénon Chen *et al.* (2021) y el mapeo de Rössler Yildirim y Tanyildizi (2023), entre otros. Se observa que incluso supera el rendimiento en la mayoría de los casos comparados.

La figura 6 muestra la gráfica de las series pseudoaleatorias obtenidas por los 10 mejores PRNG propuestos. Se observa que las trayectorias de las series numéricas obtenidas presentan comportamiento pseudoaleatorio. Adicionalmente, en la figura 7 se muestra la evidencia de sensibilidad a valores iniciales

Tabla 3: Diez mejores PRNG encontrados con GP que cumplen con la encriptación de una imagen de escala de grises y pasan el estándar de las 15 pruebas NIST 800-22 Rev.1a.

PRNG	Expresión	Entropía	Aptitud
GP1	$\mathrm{Re}((\sec(y_n)) - (\mathrm{csch}(((\sec(\sec(\sec(y_n))))+ (y_n)) - \arctan(\mathrm{R}(\sec(\sin(y_n)))))))$	7.99618	2969.6
GP2	$\mathrm{Re}(\sin(y_{n-1}) - (\log(\sin(\cot(\sec(y_{n-1}))))- (\cot(y_{n-1}))) - \csc(\|\|\cot(y_n)\|\| - y_{n-1}))$	7.99640	2970.2
GP3	$\mathrm{Re}(\sec(\sec(y_n)))$	7.99675	2971.3
GP4	$\mathrm{Re}(\tan(\csc(\csc(y_n))) + (\cot(\tan(y_{n-1})) - \cot(y_n)))$	7.99504	2966.2
GP5	$\mathrm{Re}(\cot(\cot(y_n)) - (\arcsin(\cot(\cos(y_n)))))$	7.99646	2970.4
GP6	$\mathrm{Re}(\cot(\cot(y_n)) - (\mathrm{asech}((y_n)+(\csc((\cot(y_n))- (\csc(\csc(y_n))))))))$	7.99695	2971.8
GP7	$\mathrm{Re}(\sin((y_n)^2 \times (\tan(\|\mathrm{acoth}(y_n)\|) - \mathrm{acsch}(\min(y_n, 2)))))$	7.99680	2971.4
GP8	$\mathrm{Re}(\sin((y_n)^2\times(\tan(\|\mathrm{acoth}(y_n)\|)-\mathrm{acsch}(y_n))))$	7.99680	2971.4
GP9	$\mathrm{Re}(\sin((y_n)^2\times(\tan(\|\mathrm{acoth}(y_n)\|)-\mathrm{acsch}(\min(y_n, y_n)))))$	7.99680	2971.4
GP10	$\mathrm{Re}(\sin((y_n - 2)^2\times(\tan(\|\mathrm{acoth}(y_n)\|)-(\mathrm{acsch}(\min(y_n, 2))))))$	7.99680	2971.4

Tabla 4: Rendimiento de los resultados aplicando la prueba NIST. A manera de comparación, las últimas 4 filas muestran los resultados con series generadas por sistemas caóticos de 2 o más ecuaciones, donde Tink, se refiere a la función caótica de tinkerbell; Ch a la función caótica de Chen; Hén al mapa caótico de Hénon y Röss al mapa caótico de Rössler.

PRNG	Pruebas NIST															
	F	BF	CS	Rs	LR	Rk	SD	NT	OT	U	AE	RE	RV	S	LC	media
GP1	100	99	100	100	100	98	100	100	100	100	98	100	100	100	100	99.66
GP2	100	99	100	99	100	99	99	100	99	100	100	100	100	98	100	99.53
GP3	99	100	99	98	100	98	99	100	100	100	99	100	100	100	100	99.46
GP4	100	99	100	99	99	100	99	100	99	99	99	100	100	100	99	99.46
GP5	100	100	100	98	100	100	99	100	100	99	98	100	100	98	100	99.46
GP6	98	100	99	100	99	100	100	100	98	99	100	100	100	100	99	99.46
GP7	99	99	99	98	100	100	99	100	100	99	99	100	100	100	100	99.46
GP8	99	99	99	98	100	100	99	100	100	99	99	100	100	100	100	99.46
GP9	99	99	99	98	100	100	99	100	100	99	99	100	100	100	100	99.46
GP10	99	99	99	98	100	100	99	100	100	99	99	100	100	100	100	99.46
Tink																
x_{n+1}	99	99	99	97	100	99	99	100	100	98	99	100	100	99	100	99.18
y_{n+1}	98	100	99	98	100	100	99	100	100	98	98	100	100	99	100	99.18
Ch																
x_{n+1}	98	100	99	98	89	96	97	99	100	99	99	100	100	100	100	98.28
y_{n+1}	96	99	100	99	98	99	99	100	100	98	98	100	100	100	99	99.00
Hén																
x_{n+1}	96	100	97	100	98	98	100	100	100	98	99	100	100	99	100	98.87
y_{n+1}	100	99	100	97	99	98	97	100	98	100	99	100	100	100	98	99.06
Röss																
x_{n+1}	98	99	99	97	100	99	97	100	96	98	99	100	100	100	100	98.87
y_{n+1}	100	100	100	86	100	99	99	100	96	98	99	100	100	98	100	98.37
z_{n+1}	96	100	96	100	99	99	100	100	97	98	99	100	100	97	100	98.56

que presentan los PRNG propuestos. La característica de ser sensibles a valores iniciales, produce que los PRNG propuestos sean potencialmente viables para ser utilizados en métodos criptográficos. Se observa que con un pequeño cambio en el valor inicial para generar una secuencia numérica, produce secuencias numéricas completamente distintas.

4. Conclusiones y trabajo futuro

Con el método propuesto para generar PRNG basado en programación genética, se obtuvieron 136 soluciones como posibles generadores de series numéricas pseudoaleatorias y que encriptaron la imagen muestra con una entropía mayor a H(s)=7.99. De las 136 soluciones, 58 de ellas pasaron las pruebas estadísticas NIST con un porcentaje de rendimiento mayor a 98.05 %, por lo que se dice, que las series numéricas generadas por los correspondientes PRNG, son pseudoaleatorias. Los 10 mejores PRNG seleccionados, superan el 99.46 % de rendimiento en las pruebas estadísticas mejorando los porcentajes de rendimiento obtenidos por PRNG basados en sistemas caóticos tomados como referencia para su comparación. Adicionalmente, se observó que los PRNG creados con el método propuesto, presentan la característica de ser sensibles a condiciones iniciales, es decir, con un mínimo cambio en el valor inicial, se obtienen series numéricas pseudoaleatorias completamente distintas. Por lo anterior, se concluye que las técnicas de diseño basadas en programación genética para generar PRNG presentan soluciones con un alto nivel de rendimiento y pueden ser considerados como potencialmente válidos para ser utilizados en sistemas criptográficos, sistemas de juegos de azar, sistemas de pruebas de Software, sistemas de simulación de eventos en videojuegos, entre otros. Finalmente, se propone implementar los PRNG obtenidos, en el diseño de sistemas criptográficos de imágenes a color, con el propósito de optimizar el espacio de claves del método criptográfico e implementar programación en paralelo para con ello, proponer métodos criptográficos más eficientes.

Referencias

Akhshani, A., Akhavan, A., Mobaraki, A., Lim, S.-C., y Hassan, Z. (2014). Pseudo random number generator based on quantum chaotic map. *Communications in Nonlinear Science and Numerical Simulation*, 19(1):101–111.

Bhowmik, A., Karforma, S., Dey, J., y Sarkar, A. (2021). Approximation algorithm and linear congruence: a state-of-art approach in information security issues towards internet of vehicles. *Internet of vehicles and its applications in autonomous driving*, pp. 149–172.

Chen, L., Yin, H., Yuan, L., Machado, J. T., Wu, R., y Alam, Z. (2021). Double color image encryption based on fractional order discrete improved henon map and rubik's cube transform. *Signal Processing: Image Communication*, 97:116363.

Gnatyuk, S., Okhrimenko, T., Azarenko, O., Fesenko, A., y Berdibayev, R. (2020). Experimental study of secure prng for q-trits quantum cryptography protocols. En *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 183–188. IEEE.

Hernandez, J. C., Seznec, A., e Isasi, P. (2004). On the design of state-of-the-art pseudorandom number generators by means of genetic programming. En *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volumen 2, pp. 1510–1516. IEEE.

James, F. (1990). A review of pseudorandom number generators. *Computer physics communications*, 60(3):329–344.

Kietzmann, P., Schmidt, T. C., y Wählisch, M. (2021). A guideline on pseudorandom number generation (prng) in the iot. *ACM Computing Surveys (CSUR)*, 54(6):1–38.

Kösemen, C., Dalkiliç, G., y Aydin, Ö. (2018). Genetic programming-based pseudorandom number generator for wireless identification and sensing platform. *Turkish Journal of Electrical Engineering and Computer Sciences*, 26(5):2500–2511.

Lamenca-Martinez, C., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., y Ribagorda, A. (2006). Lamar: A new pseudorandom number generator evolved by means of genetic programming. En *Parallel Problem Solving from Nature-PPSN IX: 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Proceedings*, pp. 850–859. Springer.

Moreno, M. (2015). Linear congruence generators.

Naik, R. B. y Singh, U. (2022). A review on applications of chaotic maps in pseudo-random number generators and encryption. *Annals of Data Science*, pp. 1–26.

Sathya, K., Premalatha, J., y Rajasekar, V. (2021). Investigation of strength and security of pseudo random number generators. En *IOP Conference Series: materials Science and Engineering*, volumen 1055, p. 012076. IOP Publishing.

Sayed, W. S. y Radwan, A. G. (2020). Generalized switched synchronization and dependent image encryption using dynamically rotating fractional-order chaotic systems. *AEU-International Journal of Electronics and Communications*, 123:153268.

Senkerik, R., Zelinka, I., y Pluhacek, M. (2017). Chaos-based optimization-a review. *J. Adv. Eng. Comput.*, 1(1):68–79.

Si, Y., Liu, H., y Chen, Y. (2022). Constructing a 3d exponential hyperchaotic map with application to prng. *International Journal of Bifurcation and Chaos*, 32(07):2250095.

Tornea, O. (2013). *Contributions to DNA cryptography: applications to text and image secure transmission*. Tesis doctoral, Université Nice Sophia Antipolis; Universitatea tehnică (Cluj-Napoca, Roumanie).

Yildirim, G. y Tanyildizi, E. (2023). An innovative approach based on optimization for the determination of initial conditions of continuous-time chaotic system as a random number generator. *Chaos, Solitons & Fractals*, 172:113548.

Yuan, S., Jiang, T., y Jing, Z. (2011). Bifurcation and chaos in the tinkerbell map. *International Journal of bifurcation and chaos*, 21(11):3137–3156.

Zgliczynski, P. (1997). Computer assisted proof of chaos in the rössler equations and in the hénon map. *Nonlinearity*, 10(1):243.

Zhao, C., Wang, T., Wang, H., Du, Q., y Yin, C. (2023). A novel image encryption algorithm by delay induced hyper-chaotic chen system. *The Journal of Imaging Science and Technology*.

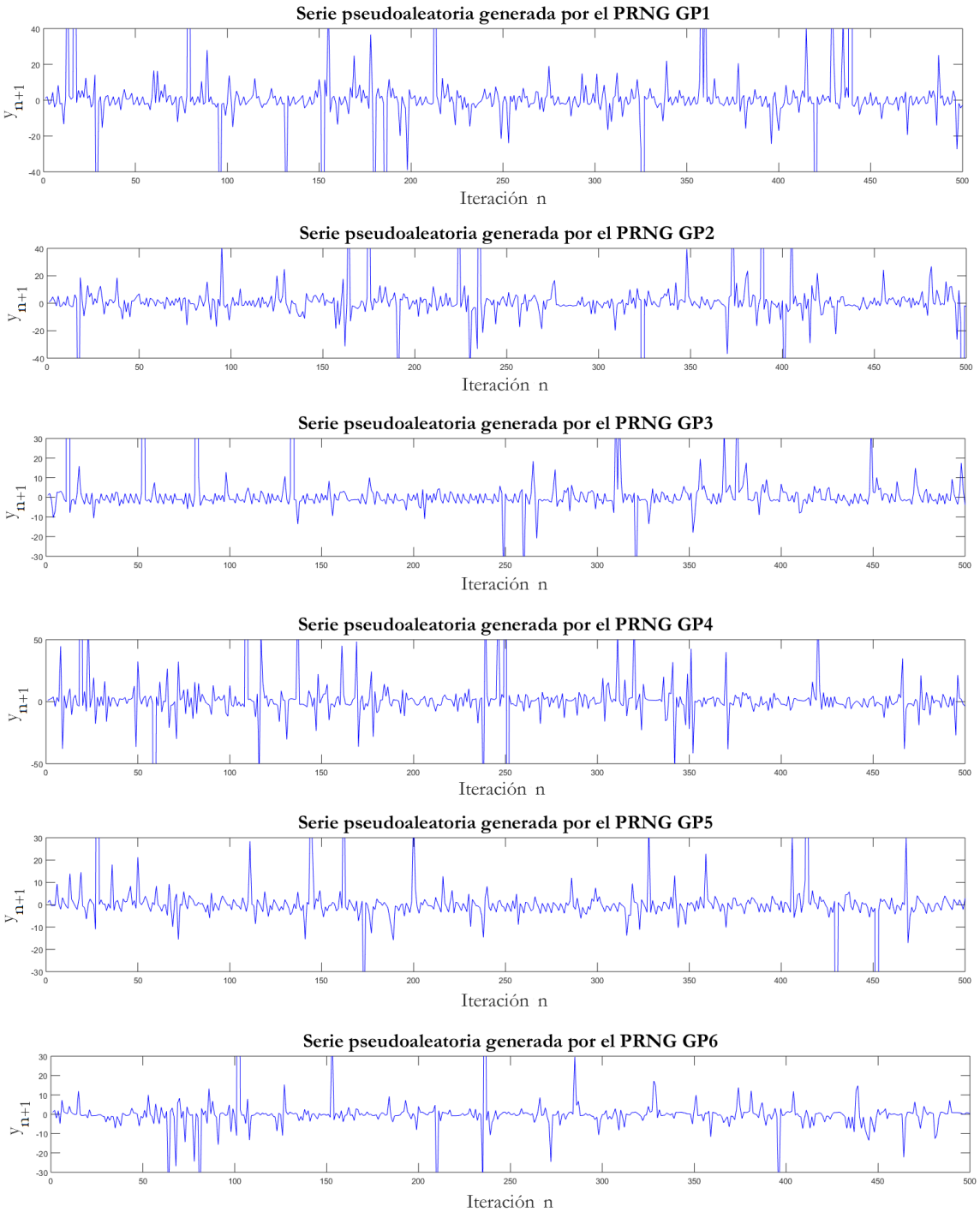


Figura 6: Series pseudoaleatorias de PRNG obtenidos.

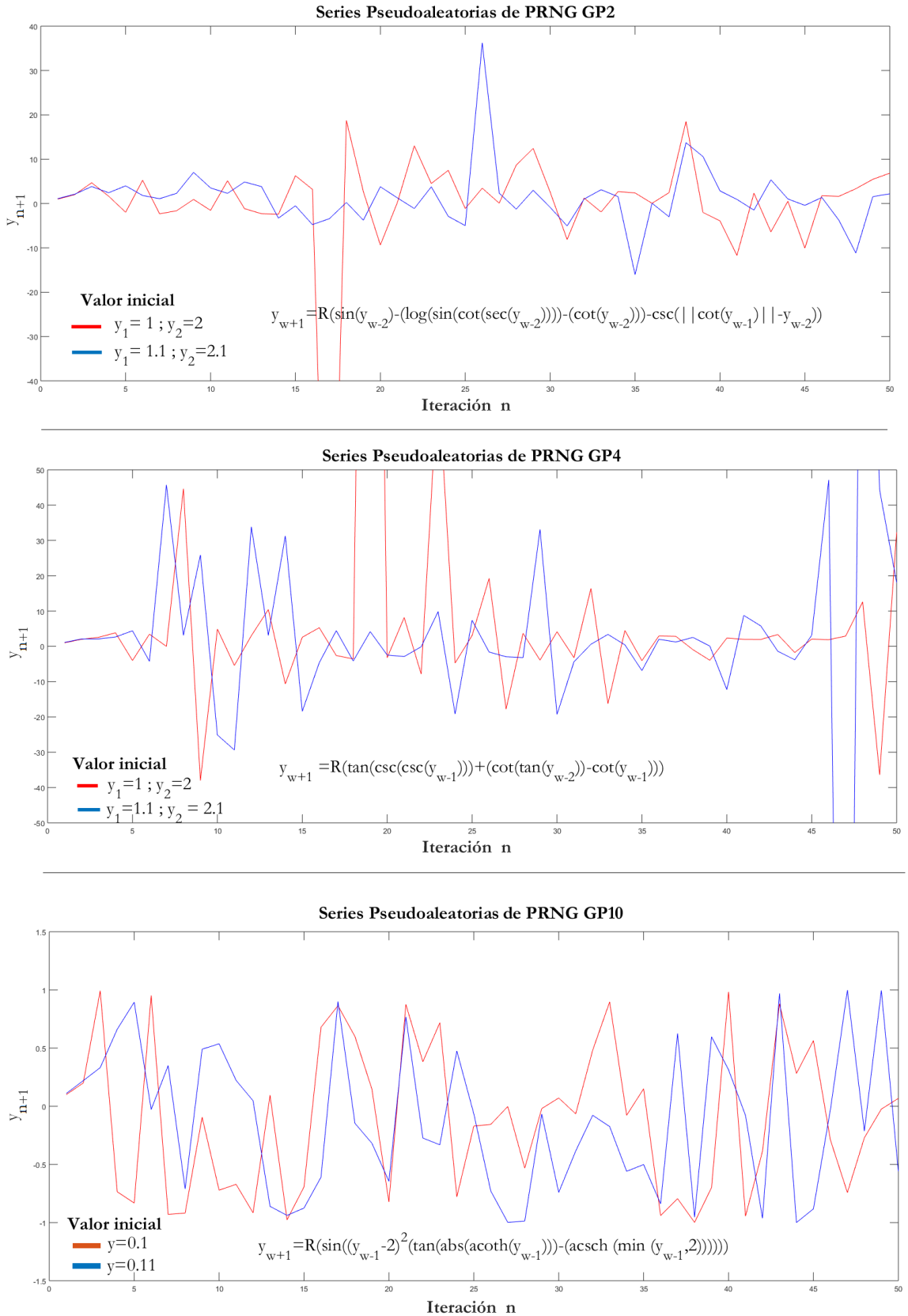


Figura 7: Sensibilidad a valores iniciales