



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Reporte Final.- Atractores y Expresiones Regulares para la
regla 22 y 54

Unidad de aprendizaje: Computing Selected Topics

Grupo: 3CM19

Alumno:
Sanchez Mendez Edmundo Josue

Profesor:
Juarez Martinez Genaro

Índice

1. Introducción	4
1.1. Atractores	4
1.2. Expresiones regulares	4
1.3. Entropía de Shannon	4
2. Expresiones Regulares	5
2.1. Regla 22	5
2.1.1. Probabilidad del 50 % de unos	5
2.1.2. Probabilidad del 95 % de unos	8
2.1.3. Generación de condición inicial completamente aleatoria	11
2.2. Regla 54	13
2.2.1. Probabilidad del 50 % de unos	14
2.2.2. Probabilidad del 95 % de unos	17
2.2.3. Generación de condición inicial completamente aleatoria	19
3. Atractores	22
3.1. Regla 22	23
3.1.1. Tamaño 2	23
3.1.2. Tamaño 3	24
3.1.3. Tamaño 4	25
3.1.4. Tamaño 5	26
3.1.5. Tamaño 6	27
3.1.6. Tamaño 7	29
3.1.7. Tamaño 8	30
3.1.8. Tamaño 9	31
3.1.9. Tamaño 10	33
3.1.10. Tamaño 11	34
3.1.11. Tamaño 12	35
3.1.12. Tamaño 13	36
3.1.13. Tamaño 14	38
3.1.14. Tamaño 15	40
3.1.15. Tamaño 16	42
3.1.16. Tamaño 17	44
3.1.17. Tamaño 18	46
3.1.18. Tamaño 19	47

3.1.19. Tamaño 20	48
3.1.20. Tamaño 21	49
3.1.21. Tamaño 22	50
3.1.22. Tamaño 23	51
3.1.23. Tamaño 24	52
3.1.24. Tamaño 25	53
3.2. Regla 54	54
3.2.1. Tamaño 2	54
3.2.2. Tamaño 3	56
3.2.3. Tamaño 4	57
3.2.4. Tamaño 5	58
3.2.5. Tamaño 6	60
3.2.6. Tamaño 7	62
3.2.7. Tamaño 8	63
3.2.8. Tamaño 9	64
3.2.9. Tamaño 10	65
3.2.10. Tamaño 11	66
3.2.11. Tamaño 12	67
3.2.12. Tamaño 13	68
3.2.13. Tamaño 14	69
3.2.14. Tamaño 15	71
3.2.15. Tamaño 16	73
3.2.16. Tamaño 17	74
3.2.17. Tamaño 18	76
3.2.18. Tamaño 19	77
3.2.19. Tamaño 20	78
3.2.20. Tamaño 21	79
3.2.21. Tamaño 22	80
3.2.22. Tamaño 23	81
3.2.23. Tamaño 24	82
4. Código	83
5. Conclusiones	102
Referencias	102

Resumen

Para este reporte abarcaremos la generación de atractores y generación de cadenas aleatorias con base en las expresiones regulares para los autómatas celulares elementales con regla 22 y 54. Nuestro objetivo es poder examinar el comportamiento de estas reglas con base en sus atractores y expresiones regulares, como en el siguiente ejemplo.

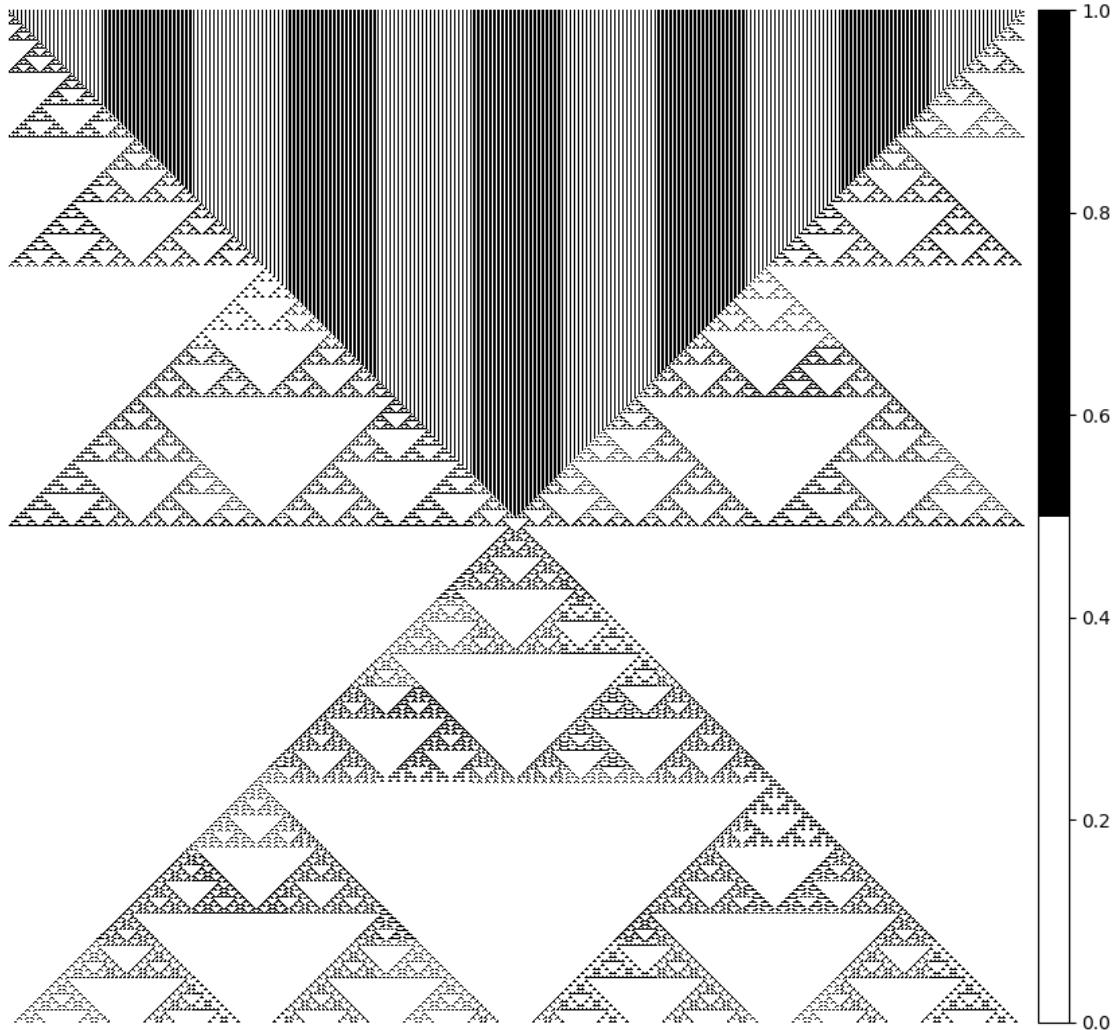


Figura 1: Autómata celular elemental de 1000 células evolucionado 1000 veces, regla 22 usando su expresión regular.

1. Introducción

1.1. Atractores

Los atractores son un conjunto de valores numéricos hacia los cuales un sistema tiende a evolucionar, dada por una variedad de condiciones iniciales en el sistema, existen tres tipos de atractores que son los siguientes:

- Atractor de punto fijo: El sistema que tenga un atractor de punto fijo tenderá a estabilizarse en un único punto.
- Atractor de ciclo límite o atractor periódico: Este tipo de atractor tiende a mantenerse en un periodo o en un mismo ciclo.
- Atractor caótico: Aparece en sistemas no lineales que tienen una gran sensibilidad a las condiciones. Un famoso ejemplo de estos atractores es el atractor de Lorenz.

1.2. Expresiones regulares

Las expresiones regulares nos permiten describir el lenguaje aceptado por un autómata finito, en nuestro caso buscamos ver como el uso de la expresiones regulares que describen a nuestras reglas afectan a la generación de autómatas celulares cuando se usan como condición inicial.

1.3. Entropía de Shannon

La entropía de Shannon es una de las métricas más importantes en la teoría de la información. La entropía mide la incertidumbre asociada con una variable aleatoria, es decir, el valor esperado de la información en el mensaje (en informática clásica se mide en bits).

El concepto fue introducido por Claude E. Shannon en el artículo “A Mathematical Theory of Communication” (1948). La entropía de Shannon permite estimar el número mínimo promedio de bits necesarios para codificar una cadena de símbolos en función del tamaño del alfabeto y la frecuencia de los símbolos y viene dada por la siguiente formula:

$$H(X) = - \sum_{i=1}^n P(x_i) \log_b P(x_i)$$

En donde b es la base del logaritmo utilizado. Los valores comunes de b son 2, el número de Euler e y 10, y las unidades correspondientes de entropía son los bits para b = 2 , nats para b = e y bans para b = 10 y $P(x_i)$ como la probabilidad de un evento, para nuestro caso de estudio b=2.

La situación de máxima incertidumbre nos genera que sea más difícil predecir el resultado siguiente, esto es debido a que hay probabilidad uniforme. La entropía, entonces, solo puede disminuir a partir del valor asociado con la probabilidad uniforme. El caso extremo es donde un estado tiene probabilidad de 1 ya que ahí no hay incertidumbre, es decir la entropía es cero.

2. Expresiones Regulares

2.1. Regla 22

La expresión regular que nos describe a la regla 22 es el siguiente:

$$(0 + 1(01)^*00)(0 + 0(01)^*00)^*$$

La expresión regular anterior nos permite poder generar cadenas aleatorias que nos servirán como condiciones iniciales para la generación de nuestros autómatas celulares elementales. Para poder visualizar como el uso de nuestra expresión regular afecta al comportamiento de nuestro autómata haremos las siguientes pruebas:

- Probabilidad del 50 % de unos.
- Probabilidad del 95 % de unos.
- Generación de condición inicial completamente aleatoria.

Estas serán comparadas con cadenas aleatorias generadas mediante la expresión regular, todo esto se hará en un espacio de 400x400 células y se compararán tanto la imagen que nos genera el autómata así como con diferentes métricas que el programa nos proporciona.

2.1.1. Probabilidad del 50 % de unos

En la figura 2 podemos ver los valores de entrada en nuestro programa, en donde definimos un espacio de 400 x 400 células con una probabilidad de unos del 50 %

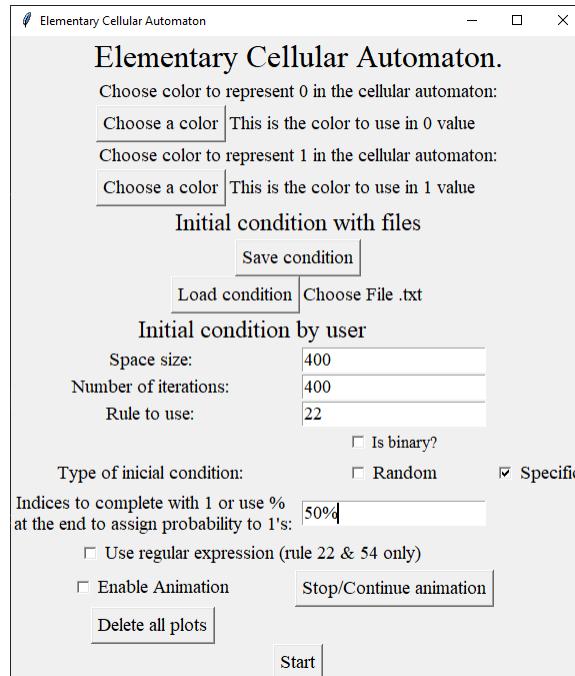


Figura 2: Entrada de nuestro programa para la primera prueba.

En la figura 3 vemos el autómata resultante así como las métricas que el programa nos genera.

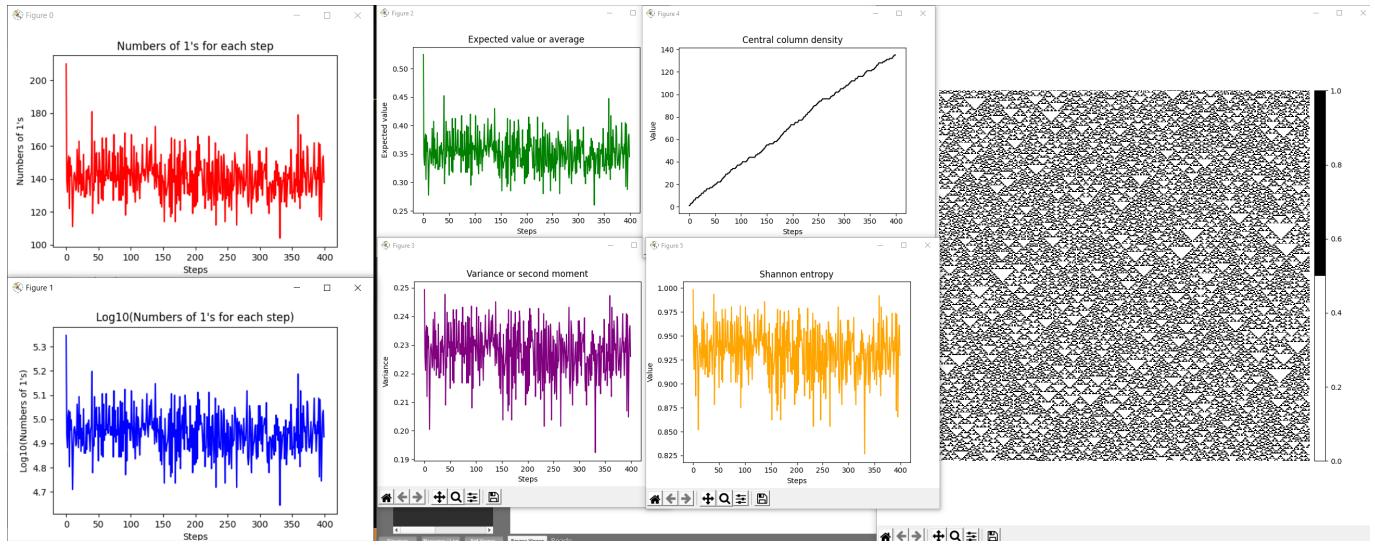


Figura 3: Autómata resultante con sus respectivas métricas.

En la figura 4 de igual manera que con la figura 2 podemos ver los valores de entrada de nuestro programa, en donde lo único que cambia es que en esta ocasión hacemos uso de la expresión regular de nuestra regla media la selección de la casilla con la etiqueta “Use regular expression (rule 22 & 54 only)”

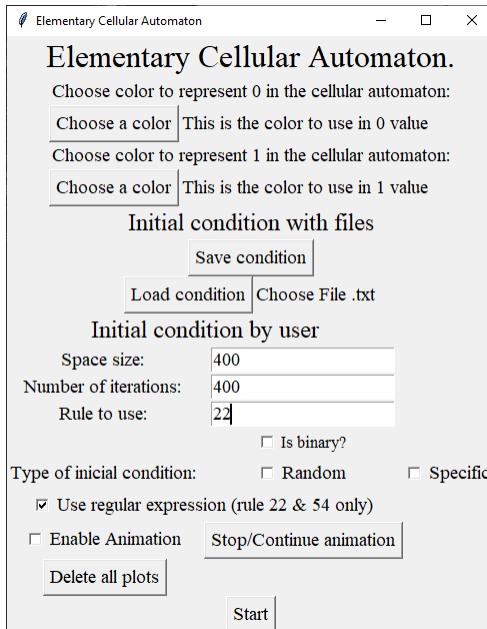


Figura 4: Entrada de nuestro usando nuestra expresión regular r22.

En la figura 5 vemos el autómata resultante así como las métricas que el programa nos genera.

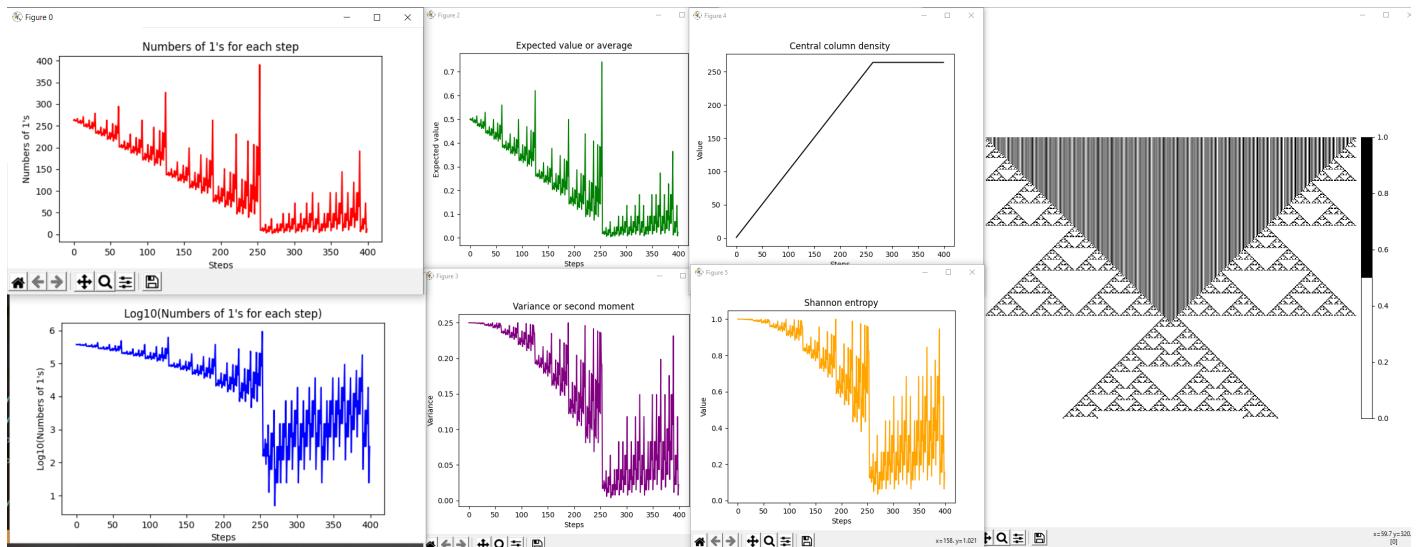


Figura 5: Autómata resultante con sus respectivas métricas.

Antes de empezar con el estudio de estos dos autómatas recordar que la expresión regular se utiliza para poder generar condiciones iniciales aleatorias en las cuales tenemos un control hasta cierto punto, por lo que el autómata mostrado en la figura 5 puede variar en forma y en consecuencia en sus métricas si los comparamos con los autómatas que se generaran en las dos pruebas restantes, una vez mencionado lo anterior, comparemos los resultados.

Para nuestro autómata con probabilidad del 50 % de unos en la condición inicial vemos como 5 de nuestras 6 gráficas tienen una silueta muy similar entre ellas, vemos como el numero de unos por paso se mantienen entre los valores de 160 y 120 en donde ocasionalmente hay pasos en donde se salen de esos valores, pero nunca llegan a superar a la primera iteración en donde el numero de pasos fue de un poco mas de 200, si vemos la entropía de Shannon vemos que el valor mas alto fue de 1 y el mas bajo de 0.825 esto nos quiere decir que tenemos casi una probabilidad uniforme, esto lo podemos afirmar ya que para poder obtener el valor máximo de la entropía de shannon necesitamos probabilidades iguales, de forma general podemos decir que tenemos una gran incertidumbre y nos vemos incapaces de poder predecir el próximo estado que tomara nuestro autómata, examinando nuestra gráfica de la densidad en la columna central vemos como nos recuerda a una linea recta, es decir su crecimiento es lineal, aunque hay ciertos puntos en donde la densidad fue constante y no tenemos un patrón reconocible en nuestro autómata.

Por otro lado tenemos al autómata generado por expresión regular y a simple vista podemos observar que es un caso completamente diferente al primero ya que podemos observar cierto patrón en nuestro autómata, el cual pareciese ser la formación de un triangulo isósceles de cabeza y a sus lados patrones formados de triángulos incrustados dentro de un triangulo superior que pareciese que avanzan hasta un punto de unión formando un triangulo diferente. Observado las métricas que tenemos disponibles podemos afirmar que hay un patrón en nuestro autómata ya que las mediciones parecen repetirse por determinados ciclos, podemos ver como el numero de unos va disminuyendo a como pasan las iteraciones y por ello vemos como la densidad de la columna central llega a un máximo de 265 células con valor 1 para pasar a ser constante, vemos a la entropía iniciar con el máximo valor que puede tomar en nuestro caso de estudio a tocar el valor mas bajo, esto quiere decir que en nuestro sistema pasamos de ser incapaces de predecir el siguiente estado a poder predecir el siguiente estado de nuestro autómata.

En resumen vemos como la expresión regular en este caso nos genera un patrón muy claro en nuestro autómata y como esto influye tanto en la generación de nuestro autómata como en sus métricas en especial caso podemos ver como la entropía de nuestros dos casos de estudio son muy distintos ya que pasamos

de tener una entropía casi máxima a tener una entropía casi mínima, es decir pasamos de tener una gran incertidumbre a prácticamente no tener incertidumbre.

2.1.2. Probabilidad del 95 % de unos

En la figura 6 podemos ver los valores de entrada en nuestro programa, en donde definimos un espacio de 400 x 400 células con una probabilidad de unos del 95 %

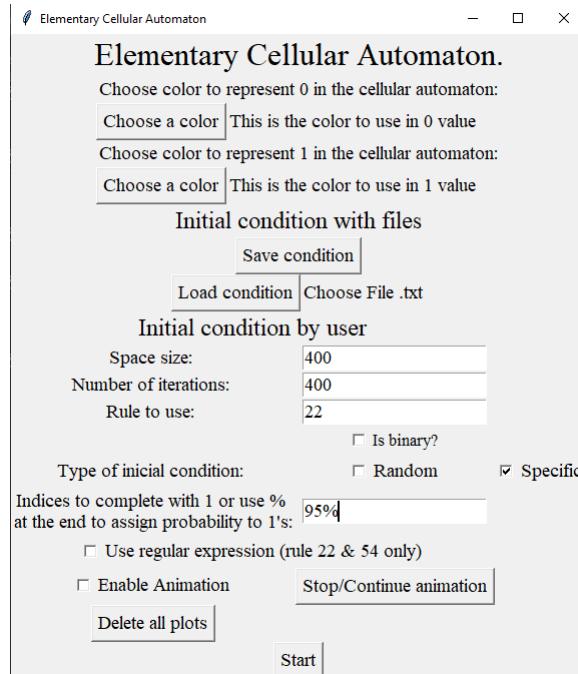


Figura 6: Entrada de nuestro programa para la segunda prueba.

En la figura 7 vemos el autómata resultante así como las métricas que el programa nos genera.

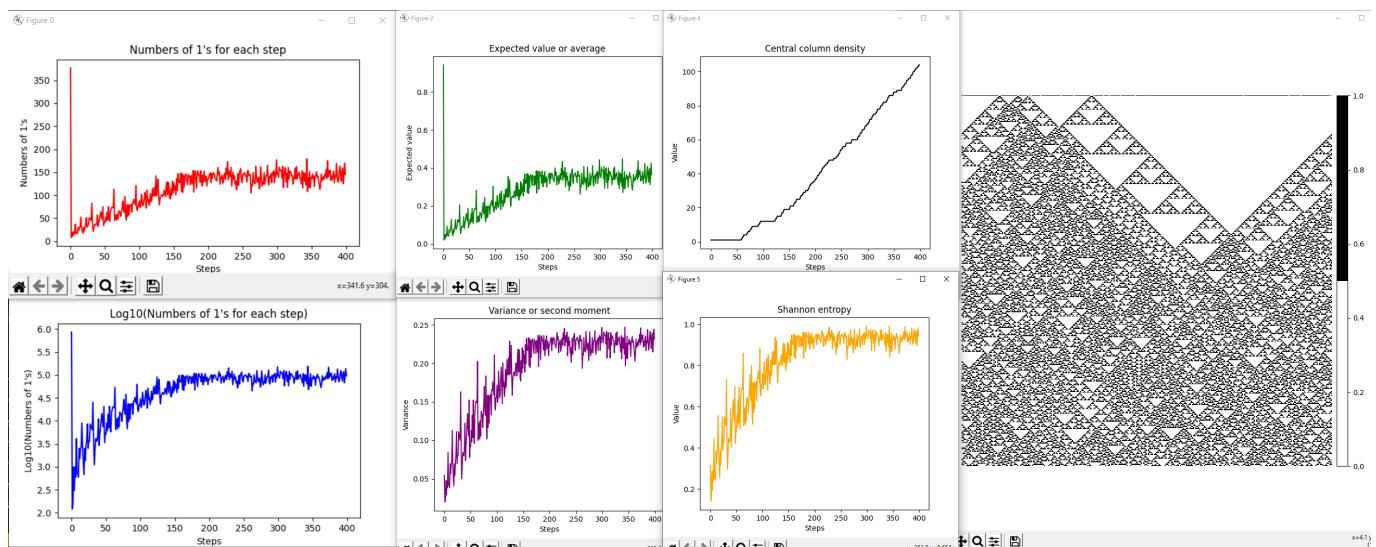


Figura 7: Autómata resultante con sus respectivas métricas.

En la figura 8 de igual manera que con la figura 6 podemos ver los valores de entrada de nuestro programa, en donde lo único que cambia es que en esta ocasión hacemos uso de la expresión regular de nuestra regla media la selección de la casilla con la etiqueta “Use regular expression (rule 22 & 54 only)”

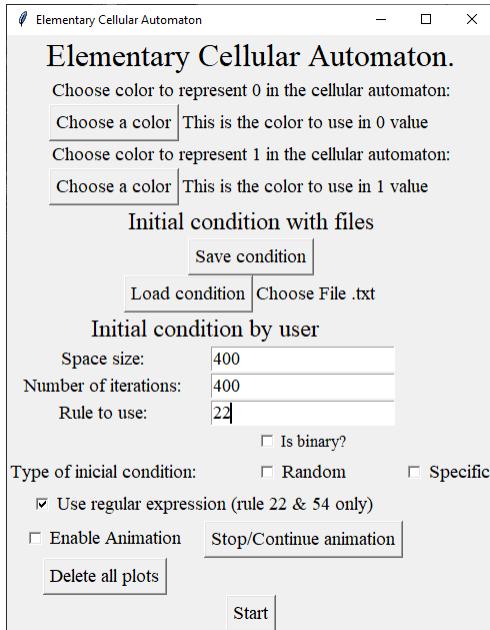


Figura 8: Entrada de nuestro usando nuestra expresión regular r22.

En la figura 9 vemos el autómata resultante así como las métricas que el programa nos genera.

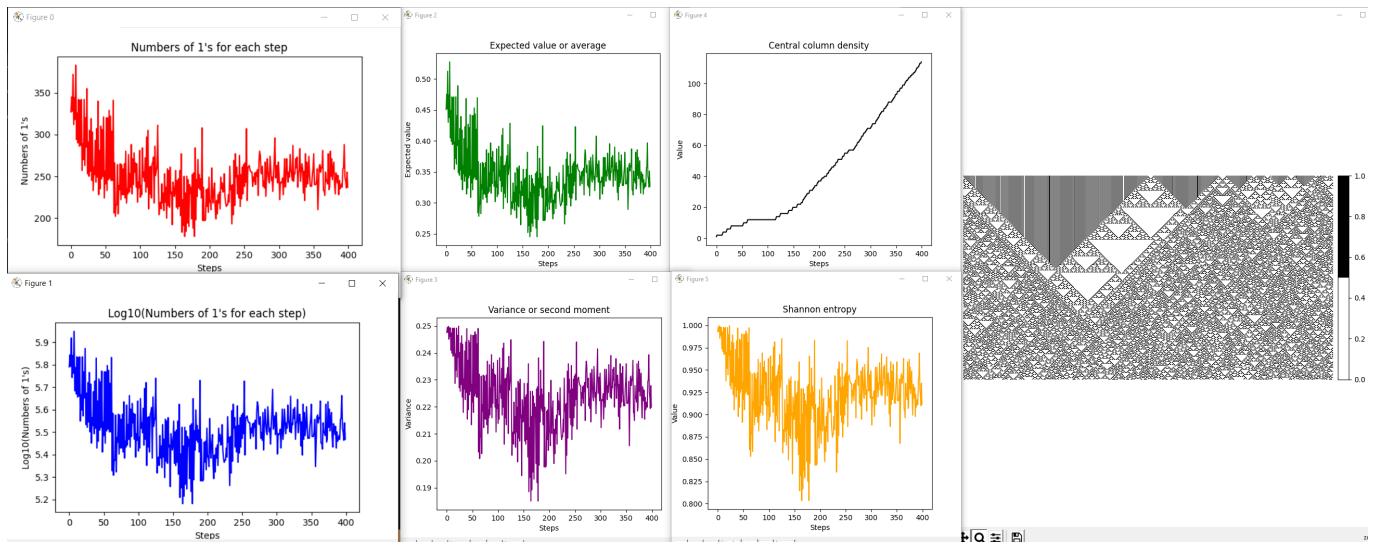


Figura 9: Autómata resultante con sus respectivas métricas.

Para nuestro autómata con probabilidad del 95 % de unos en la condición inicial vemos como el numero de unos, el \log_{10} y el valor esperado tienen una silueta muy similar entra ellas de igual forma la varianza con la entropía de Shannon, vemos como el numero de unos por paso tiene un máximo al inicio con un valor de 377 a tener un bajo en la primera iteración teniendo como valor de 9, después de este máximo y mínimo los unos tienden a crecer de una forma logarítmica ya que parece tener una asíntota horizontal en el valor 150, si vemos la entropía de Shannon podemos ver de manera clara como

al inicio tenemos el valor mínimo de entropía, es decir no tenemos incertidumbre es prácticamente nula, sin embargo, con el paso de las iteraciones esta incertidumbre va aumentando hasta llegar prácticamente al punto máximo de entropía, es decir tenemos una gran incertidumbre y nos vemos incapaces de poder predecir el próximo estado que tomara nuestro autómata, examinando nuestra gráfica de la densidad en la columna central vemos como es constante el numero de 1's hasta la iteración 58 ya que desde la siguiente iteración comienza a ver un crecimiento escalonado, finalizando el análisis de este autómata podemos decir que no tenemos un patrón reconocible, sin embargo, podríamos decir que al inicio teníamos formas triangulares inscritas en un triángulo mayor pero en un punto estas figuras se perdieron.

Por otro lado tenemos al autómata generado por expresión regular y a simple vista podemos observar como comparte un patrón con el autómata de la figura 5 y es que vemos como en una parte del autómata tenemos líneas rectas que nos forman un triángulo invertido y vemos como las gráficas no se parecen en nada a la prueba de 95 % de unos. Observado las métricas que tenemos disponibles observamos una gran variación en la información que nos da el autómata y es algo completamente diferente a la prueba que estudiamos ya que en esa prueba vemos como tienden a un valor, en cambio, en este autómata podemos ver como el numero de unos va disminuyendo a como pasan las iteraciones pero no es una disminución constante si no que es como saltado y no es una reducción continua, sin embargo, vemos como la densidad de la columna central va aumentando de una forma casi lineal, pero volvemos a ver como hay puntos en donde el valor no cambia pero en su mayoría son lapsos muy pequeños, pero en la entropía podemos ver como inicia prácticamente en el máximo valor que puede tomar y después de 16 iteraciones empieza a disminuir y oscilar entre los valores 0.8 y 0.975, esto quiere decir que en nuestro sistema dentro de todas las iteraciones nos vemos incapaces de predecir el siguiente estado nuestro autómata y tenemos una alta incertidumbre.

En resumen vemos como la expresión regular en este caso no nos genera un patrón muy claro en este autómata, sin embargo, vemos como comparte un pequeño patrón con el autómata de la figura 5, así vemos como la expresión regular sigue influyendo en la generación de nuestro autómata. En esta ocasión volvemos a ver como las métricas no coinciden, sin embargo, podemos ver como hay partes de nuestro autómata que se parecen un poco pero que nos proporcionan información distinta.

2.1.3. Generación de condición inicial completamente aleatoria

En la figura 10 podemos ver los valores de entrada en nuestro programa, en donde definimos un espacio de 400 x 400 células haciendo uso de la función random de la librería numpy de Python el cual nos generara un arreglo de determinado tamaño, en este caso de 400 células, lleno aleatoriamente de números 0 y 1.

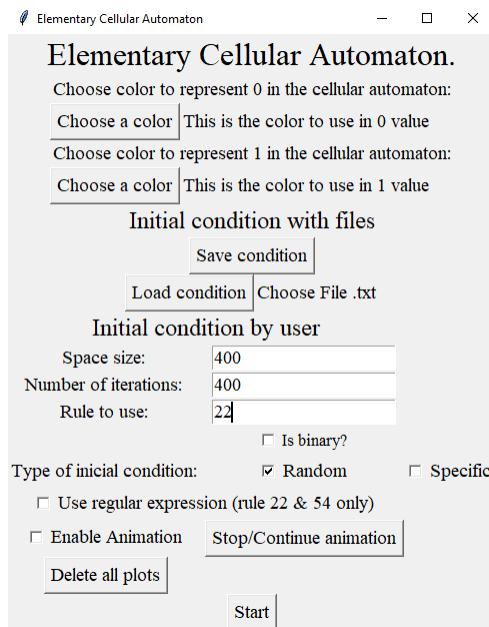


Figura 10: Entrada de nuestro programa para la tercera prueba.

En la figura 11 vemos el autómata resultante así como las métricas que el programa nos genera.

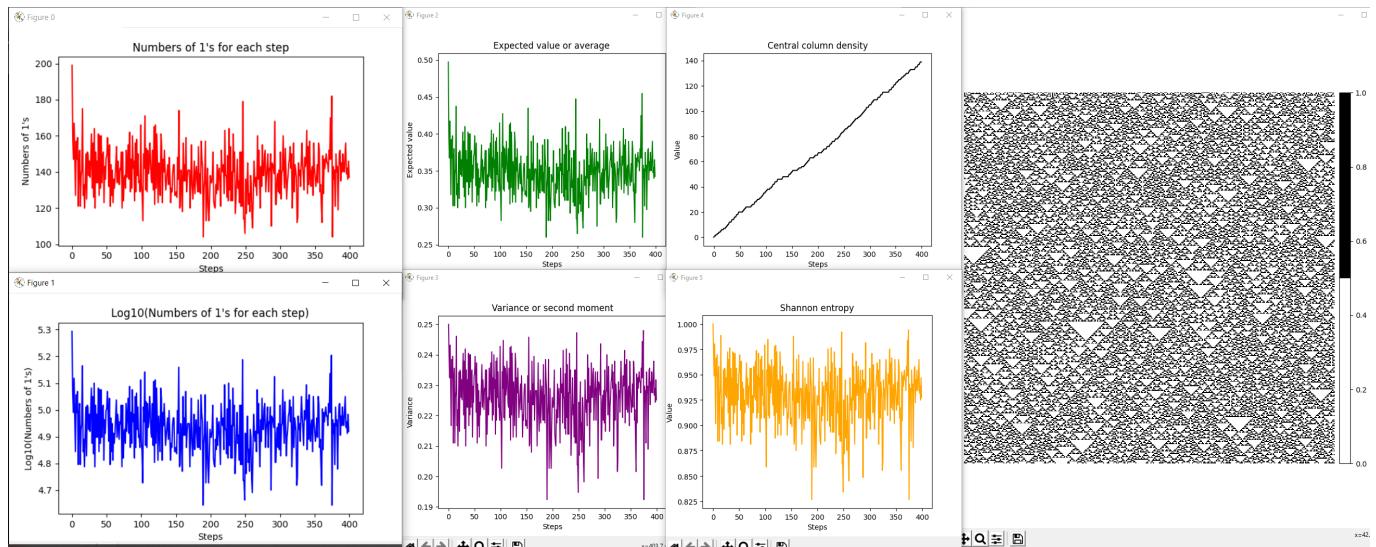


Figura 11: Autómata resultante con sus respectivas métricas.

En la figura 12 de igual manera que con la figura 10 podemos ver los valores de entrada de nuestro programa, en donde lo único que cambia es que en esta ocasión hacemos uso de la expresión regular de nuestra regla media la selección de la casilla con la etiqueta “Use regular expression (rule 22 & 54 only)”

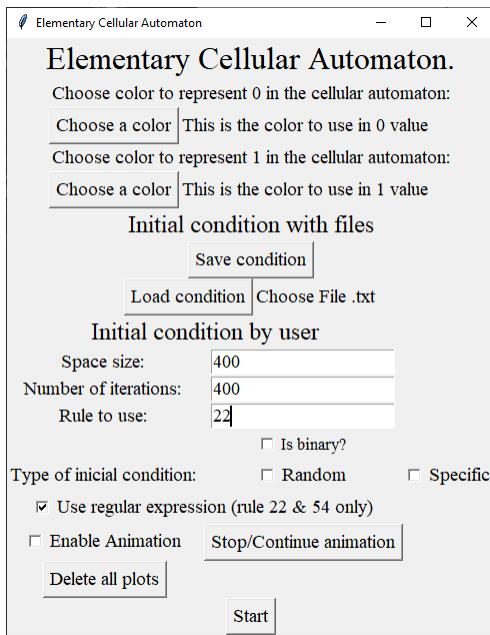


Figura 12: Entrada de nuestro usando nuestra expresión regular r22.

En la figura 13 vemos el autómata resultante así como las métricas que el programa nos genera.

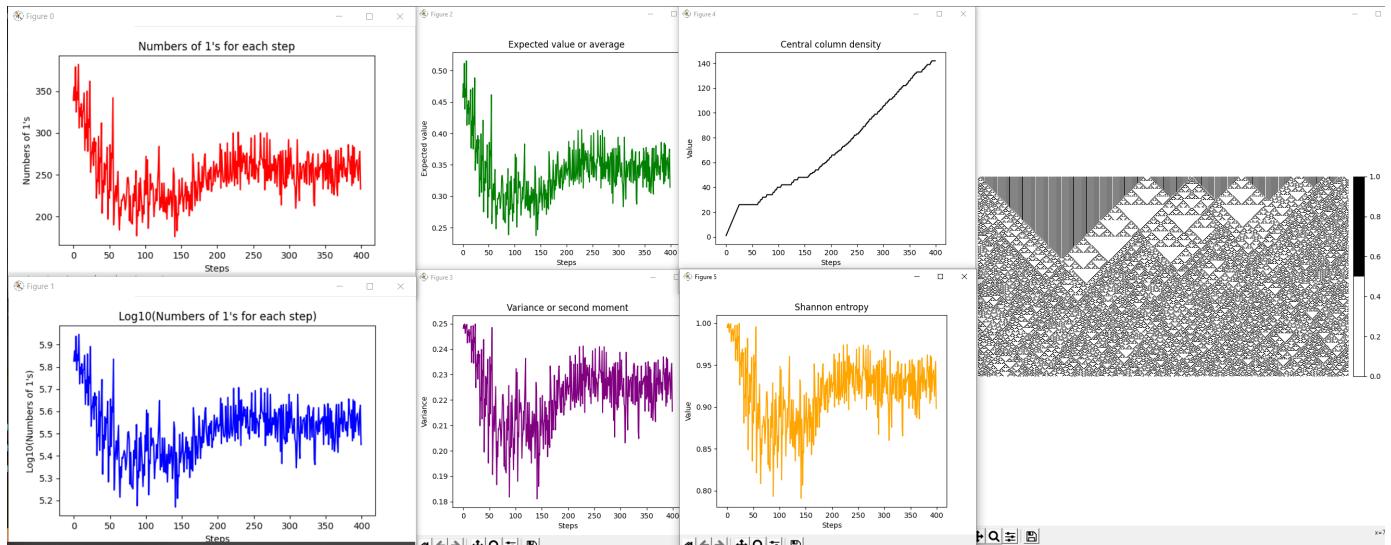


Figura 13: Autómata resultante con sus respectivas métricas.

Para nuestro autómata generado de manera aleatoria vemos como las gráficas comparten una silueta muy similar entra, vemos como el numero de unos por paso tiene un máximo al inicio con un valor de 200 para después disminuir y rondar valores de 104 como el mas bajo a 182 como el valor mas alto, si vemos la entropía de Shannon podemos ver que volvemos a tener un patrón similar a las pruebas pasadas, en donde comenzamos con prácticamente la entropía máxima a disminuir un poco y oscilar entre los valores 0.825 y 0.9941, esto nos vuelve a indicar que tenemos incertidumbre, examinando nuestra gráfica de la densidad en la columna central vemos un crecimiento prácticamente lineal en el numero de 1's pero hay que mencionar que volvemos a ver partes en donde la gráfica de densidad es constante, finalizando el análisis de este autómata podemos decir que no tenemos un patrón reconocible a diferencia de pruebas

pasadas en donde si podríamos llegar a visualizar patrones.

Por otro lado tenemos al autómata generado por expresión regular y a simple vista podemos observar como volvemos a compartir el patrón de triangulo invertido formado por líneas rectas de valores 0 y 1 intercaladas de los autómatas generados por expresión regular de las pruebas anteriores y de hecho podemos ver un gran parecido con el autómata de la figura 9 tanto en las gráficas como en el mismo autómata solo que el autómata que estamos estudiando tenemos el patrón de los triángulos incrustados en otro aún mayor pero en esta ocasión podemos ver esa patrón mas veces pero volvemos a llegar un punto en donde los patrones desaparecen. Observado las métricas que tenemos disponibles observamos como el numero de unos tiene un máximo de 382 unos a tener un mínimo de 177 para después empezar a rondar los valores de 211 a 300 células con valor 1, sin embargo, vemos como la densidad de la columna central al inicio tienen un crecimiento lineal entre la iteración inicial hasta la 25 para después volverse constante hasta la iteración 60 en donde ya empieza a existir un incremento casi lineal pero volvemos a ver como hay partes en donde es constante, pero en la entropía podemos ver que es muy similar a la vista en la figura 9 en donde vemos como inicia prácticamente en el máximo valor que puede tomar y después de 16 iteraciones empieza a disminuir y oscilar entre los valores 0.79 y 0.99, esto quiere decir que en nuestro sistema dentro de todas las iteraciones nos vemos incapaces de predecir el siguiente estado nuestro autómata y tenemos una alta incertidumbre.

Una vez visto estas tres pruebas podemos decir que la expresión regular nos genera patrones comunes entre los diferentes autómatas aun a pesar de que esta se genera de manera aleatoria, el patrón más reconocible son las intercalaciones de columnas de ceros y unos que forman un triangulo invertido, mientras que también existe otro patrón en donde hay triángulos inscritos en un uno mayor y es común ver estos patrones juntos, también vemos como si usamos el 50 %, 95 % e inclusive la formación aleatoria nos generan autómatas en donde la mayor parte de la iteraciones tenemos una gran entropía aunque esto también lo podemos ver en la generada mediante la expresión regular aunque existen formaciones como la figura 5 en donde la entropía llega casi al mínimo valor.

2.2. Regla 54

La expresión regular que nos describe a la regla 54 es el siguiente:

$$(0 + 1(11)^*10)(0 + 0(11)^*10)^*$$

La expresión regular anterior nos permite poder generar cadenas aleatorias que nos servirán como condiciones iniciales para la generación de nuestros autómatas celulares elementales. Para poder visualizar como el uso de nuestra expresión regular afecta al comportamiento de nuestro autómata haremos las siguientes pruebas:

- Probabilidad del 50 % de unos.
- Probabilidad del 95 % de unos.
- Generación de condición inicial completamente aleatoria.

Estas serán comparadas con cadenas aleatorias generadas mediante la expresión regular, todo esto se hará en un espacio de 400x400 células y se compararan tanto la imagen que nos genera el autómata así como con diferentes métricas que el programa nos proporciona.

2.2.1. Probabilidad del 50 % de unos

En la figura 14 podemos ver los valores de entrada en nuestro programa, en donde definimos un espacio de 400 x 400 células con una probabilidad de unos del 50 %

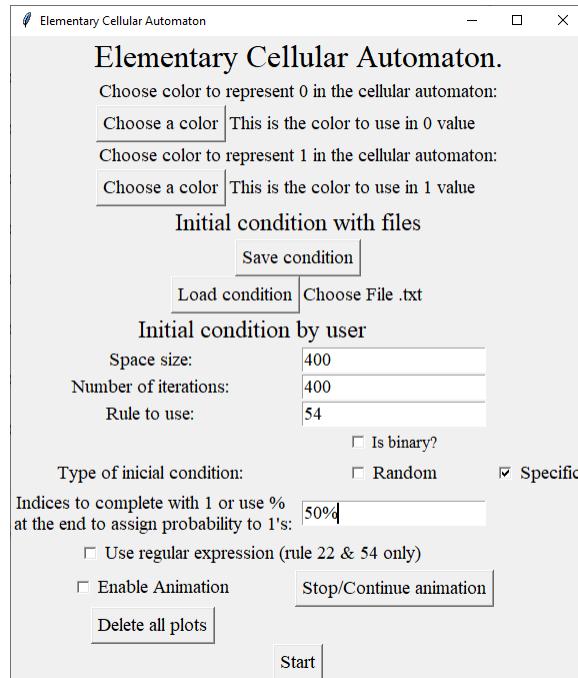


Figura 14: Entrada de nuestro programa para la primera prueba.

En la figura 15 vemos el autómata resultante así como las métricas que el programa nos genera.

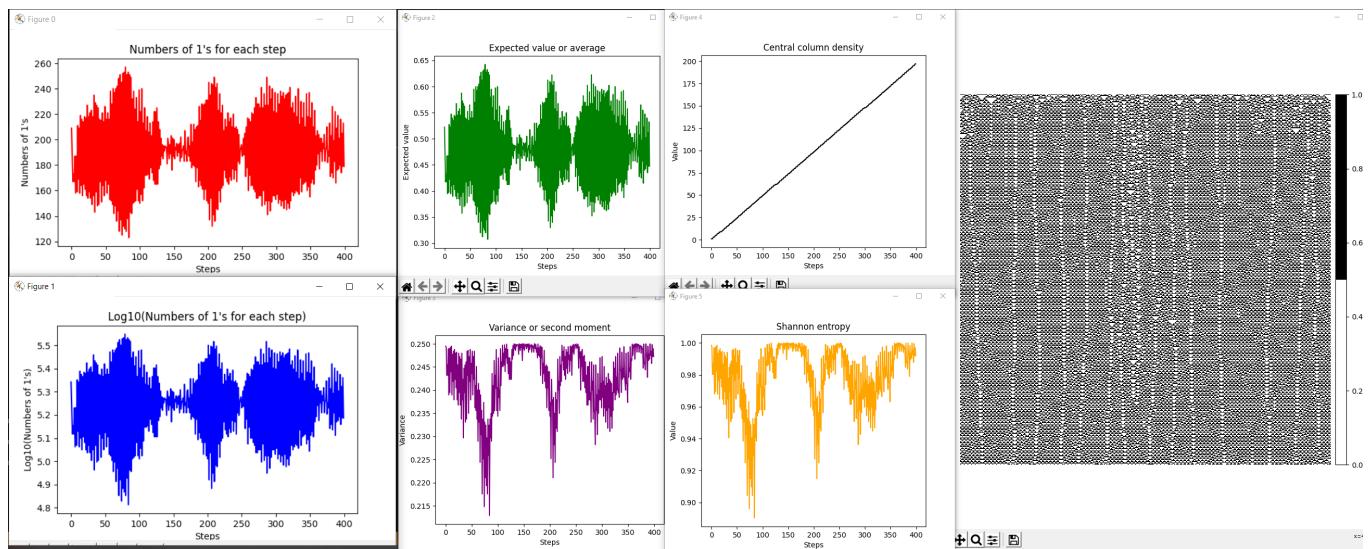


Figura 15: Autómata resultante con sus respectivas métricas.

En la figura 16 de igual manera que con la figura 14 podemos ver los valores de entrada de nuestro programa, en donde lo único que cambia es que en esta ocasión hacemos uso de la expresión regular de nuestra regla media la selección de la casilla con la etiqueta “Use regular expression (rule 22 & 54 only)”

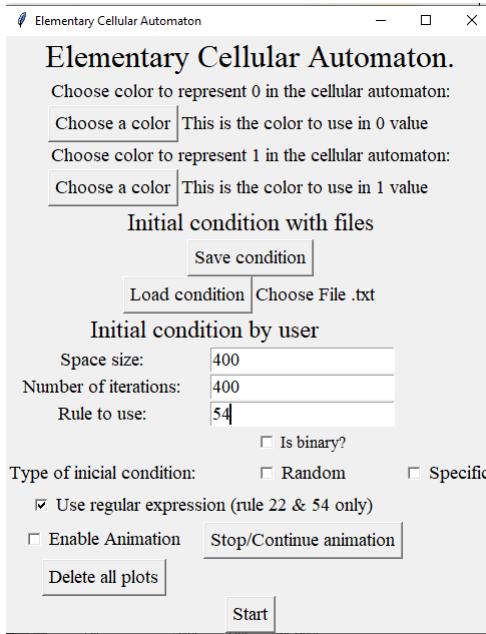


Figura 16: Entrada de nuestro usando nuestra expresión regular r54.

En la figura 17 vemos el autómata resultante así como las métricas que el programa nos genera.

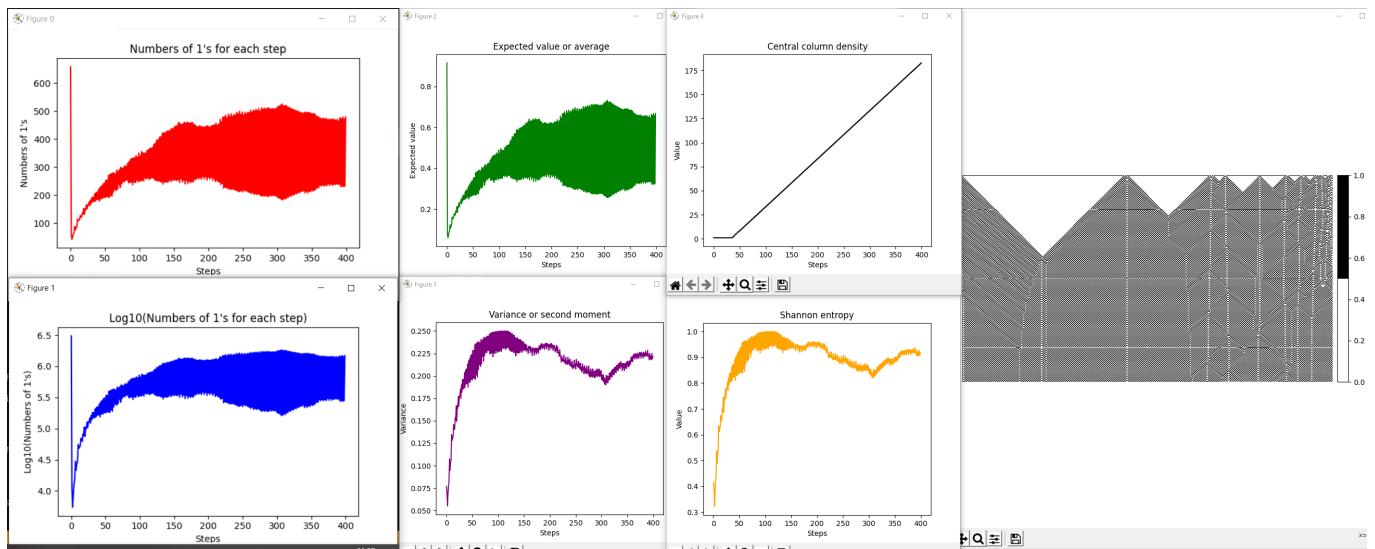


Figura 17: Autómata resultante con sus respectivas métricas.

Para nuestro autómata con probabilidad del 50 % de unos en la condición inicial vemos como 3 de nuestras gráficas tienen una silueta muy similar entre ellas, vemos como el numero de unos por paso se mantienen entre los valores de 123 y 258, si examinamos un poco la gráfica vemos como hay pasos intermedios en donde se encuentran prácticamente a la mitad de los valores anteriormente mencionados y si nos ponemos algo imaginativos podría llegar a parecer la gratificación de una parte de algún audio, si vemos la entropía de Shannon vemos que el valor prácticamente se mantiene en el valor mas alto que es 1 tenemos pasos intermedios en donde la entropía disminuye un poco, las partes en donde la entropía es prácticamente el valor máximo coincide con los puntos en donde el numero de 1 por paso se encuentra cercano a 200 1's por paso, esto es lógico ya que la probabilidad de los eventos sera igualmente probable

provocando así la entropía mas alta, como ya sabemos al tener esta entropía tan alta tenemos una gran incertidumbre y nos vemos incapaces de poder predecir el próximo estado que tomara nuestro autómata, examinando nuestra gráfica de la densidad en la columna central vemos como nos recuerda a una linea recta, es decir su crecimiento es lineal, pero si examinamos a detalle nuestra gráfica parece una escalera en donde los escalones tienen diferente longitud, esto se debe a que hay ciertas iteaciones en donde el numero de 1's se mantiene constante, y no tenemos un patrón reconocible en nuestro autómata pero si tenemos lineas formadas por triángulos invertidos pequeños a través de todo nuestro autómata.

Por otro lado tenemos al autómata generado por expresión regular y a simple vista podemos observar que es un caso diferente al primero ya que podemos observar cierto patrón en nuestro autómata, el cual tiene triángulos invertidos compuesto de células con valor 0 al inicio, para después desaparecer, y también los triángulos que se complementan con los anteriores justo a la mitad, empiezan a generar lineas prácticamente rectas que se dirigen hacia la parte baja. Observando las métricas que tenemos disponibles podemos ver como el numero de unos tiene un comportamiento muy interesante y es que vemos como al inicio de nuestro autómata tiene un máximo de 1's teniendo un valor de 657 que disminuye a un mínimo de 42 en la iteración 2 y a partir de ahí va incrementando de una manera casi logarítmica pero con el detalle de oscilar entre los valores 184 y 525, de hecho la densidad de la columna central al inicio es constante hasta la iteración 36 ya que desde ahí empieza a tener un incremento casi lineal ya que volvemos a ver ese escalonado pero en esta ocasión es prácticamente de una iteración, vemos a la entropía iniciar con un valor de 0.41 de entropía para luego caer a 0.32 y a partir de ahí comportarse como las demás gráficas y empezar a subir prácticamente hasta el valor máximo esto quiere decir que en nuestro sistema pasamos de poder tener la capacidad de predecir el siguiente estado a ser incapaces de predecir el siguiente estado de nuestro autómata, en otras palabras pasamos de tener baja incertidumbre a tener prácticamente la máxima incertidumbre posible.

En resumen vemos como la expresión regular en este caso nos genera algo completamente a la prueba del 50 %, algo curioso de estas pruebas es que en la entropía a pesar de que el autómata generado por medio de la expresión regular tiene baja entropía al inicio pocas iteraciones después tenemos prácticamente la máxima al igual que el caso de estudio del 50 % de probabilidad, también vemos que la gráfica de la columna de densidad central ambas tienen un crecimiento casi lineal y en los autómatas nos encontramos con pequeñas columnas que atraviesan al autómata en su totalidad.

2.2.2. Probabilidad del 95 % de unos

En la figura 18 podemos ver los valores de entrada en nuestro programa, en donde definimos un espacio de 400×400 células con una probabilidad de unos del 95 %

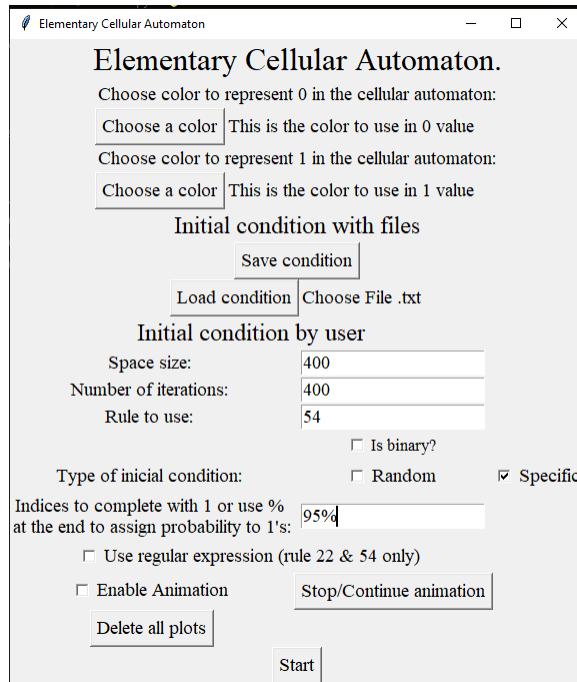


Figura 18: Entrada de nuestro programa para la segunda prueba.

En la figura 19 vemos el autómata resultante así como las métricas que el programa nos genera.

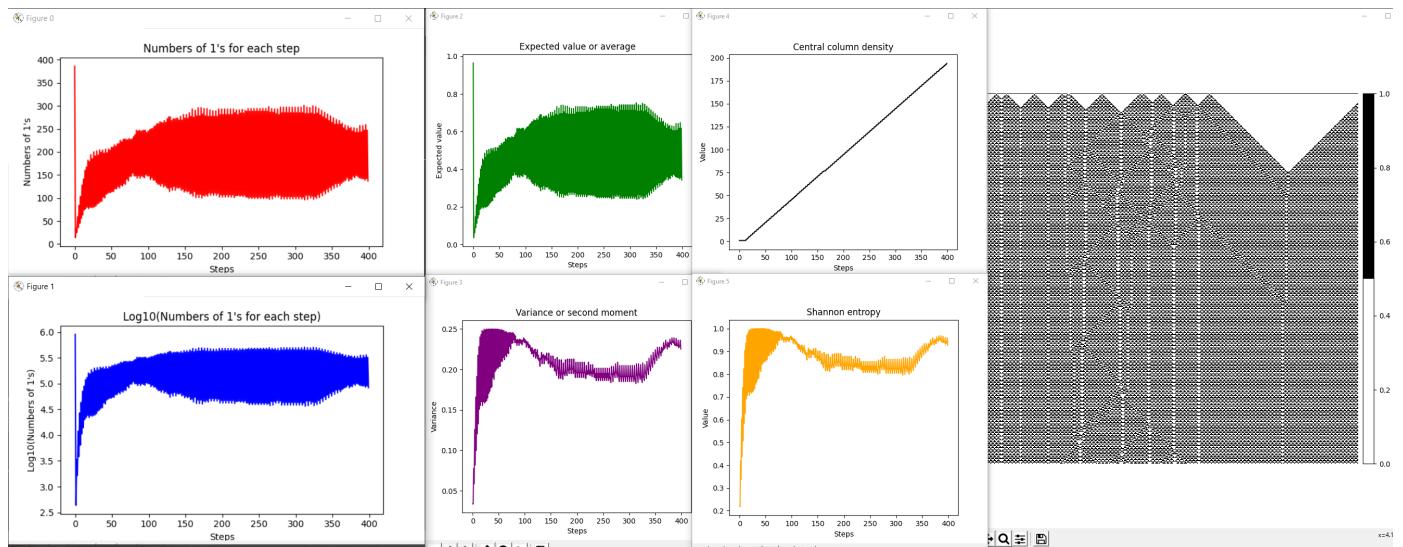


Figura 19: Autómata resultante con sus respectivas métricas.

En la figura 20 de igual manera que con la figura 18 podemos ver los valores de entrada de nuestro programa, en donde lo único que cambia es que en esta ocasión hacemos uso de la expresión regular de nuestra regla media la selección de la casilla con la etiqueta “Use regular expression (rule 22 & 54 only)”

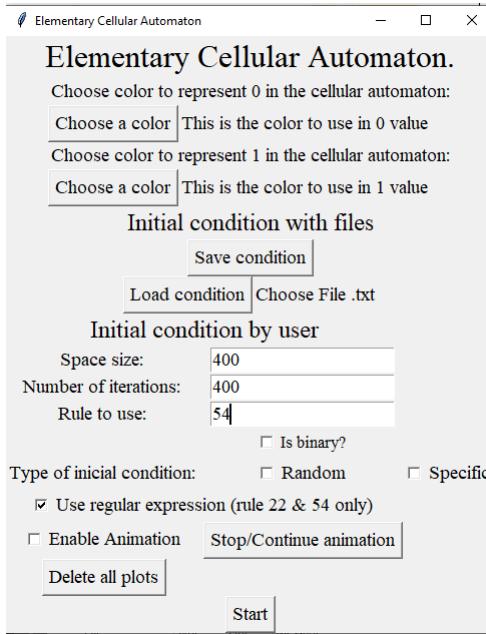


Figura 20: Entrada de nuestro usando nuestra expresión regular r54.

En la figura 21 vemos el autómata resultante así como las métricas que el programa nos genera.

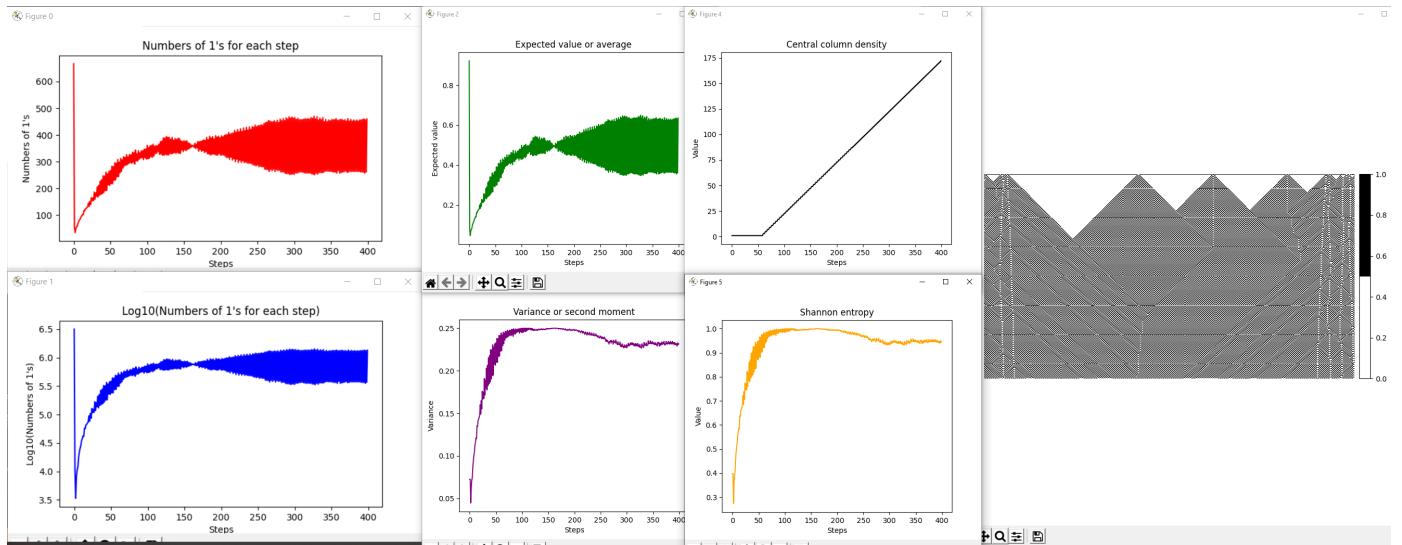


Figura 21: Autómata resultante con sus respectivas métricas.

Para nuestro autómata con probabilidad del 95 % de unos en la condición inicial vemos nos recuerda un poco al autómata de la figura 17 el cual fue generado con base en la expresión regular, ya que vemos métricas con siluetas prácticamente idénticas, de hecho a simple vista podríamos llegar a pensar en que nuestro autómata generado forma parte del autómata de la figura 17, entrando directamente en el estudio de las métricas podemos observar como el numero de unos por paso coincide con el comportamiento del autómata de la figura 17, ya que pasamos de un máximo, en este caso con valor de 386, para pasar a un mínimo de 14 unos en el paso 1, para empezar a subir en el numero de unos como si fuese una función logarítmica, solo que se mantiene oscilando entre los valores 98 como el límite inferior y de 300 como el límite superior, si vemos la entropía de Shannon podemos ver sin un duda alguna un comportamiento

parecido a la entropía de nuestro autómata de la figura 17, ya que de forma general partimos de un valor muy bajo, en este caso de estudio partimos del valor 0.2189 a empezar a subir de manera muy rápida a prácticamente el valor máximo posible de entropía, de hecho entre el paso 10 al 80 tiene una oscilación en los valores pasando de la entropía máxima a una inferior, el mínimo fue de 0.70, del paso 81 en adelante vemos como decrece y crece la entropía, esto entre los valores 0.80 y 0.9680, como algo interesante y al mismo tiempo curioso vemos que entre paso 81 al 400 nos recuerda un poco al electrocardiograma, por supuesto, esto en silueta, para terminar la parte de la entropía podemos decir que en nuestro sistema pasamos de poder tener la capacidad de poder saber el siguiente estado de nuestro autómata a perder dicha capacidad, en otras palabras pasamos de una incertidumbre muy baja a prácticamente la más alta posible, por último, examinando nuestra gráfica de la densidad en la columna central vemos nuevamente un comportamiento parecido con el autómata de la figura 17, en donde tenemos una parte constante de 1's para empezar a subir prácticamente de forma lineal, pero viendo de nuevo ese parte escalonada, en donde hay pasos en donde la densidad no es afectada, en este caso la parte constante es hasta el paso 11 y es ahí en donde vemos el crecimiento en la densidad de la columna central.

Por otro lado tenemos al autómata generado por expresión regular y a simple vista podemos observar una similitud con el autómata de la figura 17, tanto en comportamiento de las gráficas como en el autómata mismo, solo que en esta ocasión no tenemos una linea formada de triángulos que atravesen todo nuestro autómata en algunos de nuestros triángulos. Observando las métricas que tenemos disponibles podemos ver como el numero de unos tiene un comportamiento muy interesante y es que vemos como al inicio de nuestro autómata tiene un máximo de 1's teniendo un valor de 667 que disminuye a un mínimo de 34 en la iteración 2 y a partir de ahí va incrementando de una manera casi logarítmica con un oscilamiento muy interesante ya que pareciese que se complementan los pasos entre si, para que a partir del paso 167 oscile entre los valores 252 y 471, de igual forma los pasos parecen complementarse, de hecho la densidad de la columna central al inicio es constante hasta la iteración 58 ya que desde ahí empieza a tener un incremento casi lineal ya que volvemos a ver ese escalonado en donde tenemos 2 pasos que comparten la misma densidad hasta el paso 230 en donde ahora 1 paso es el que comparte la densidad anterior, vemos a la entropía iniciar con un valor de 0.39 de entropía para luego caer a 0.2743 y a partir de ahí comportarse como las demás gráficas y empezar a subir prácticamente hasta el valor máximo, sin embargo, vale la pena mencionar que parece que la entropía va decayendo poco a poco, pero esto quiere decir que en nuestro sistema pasamos de poder tener la capacidad de predecir el siguiente estado a ser incapaces de predecir el siguiente estado de nuestro autómata, en otras palabras pasamos de tener baja incertidumbre a tener prácticamente la máxima incertidumbre posible.

En resumen vemos como la expresión regular en este caso nos genera algo similar a la prueba del 95 %, esto nos quiere decir que nuestra expresión regular nos genera condiciones iniciales en donde podríamos encontrar un mínimo del 95 % de unos ya que los autómatas que se generaron fueron muy similares, principalmente en el comportamiento del autómata en donde el podemos ver ciertas similitudes entre ambos, también así en el comportamiento de nuestras métricas.

2.2.3. Generación de condición inicial completamente aleatoria

En la figura 22 podemos ver los valores de entrada en nuestro programa, en donde definimos un espacio de 400 x 400 células haciendo uso de la función random de la librería numpy de Python el cual nos generara un arreglo de determinado tamaño, en este caso de 400 células, lleno aleatoriamente de números 0 y 1.

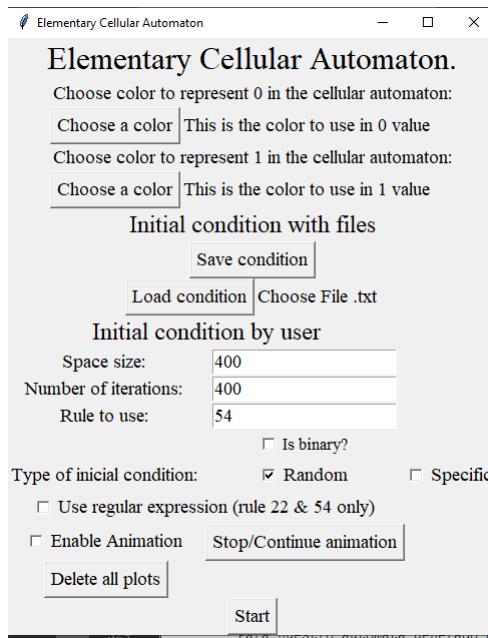


Figura 22: Entrada de nuestro programa para la tercera prueba.

En la figura 23 vemos el autómata resultante así como las métricas que el programa nos genera.

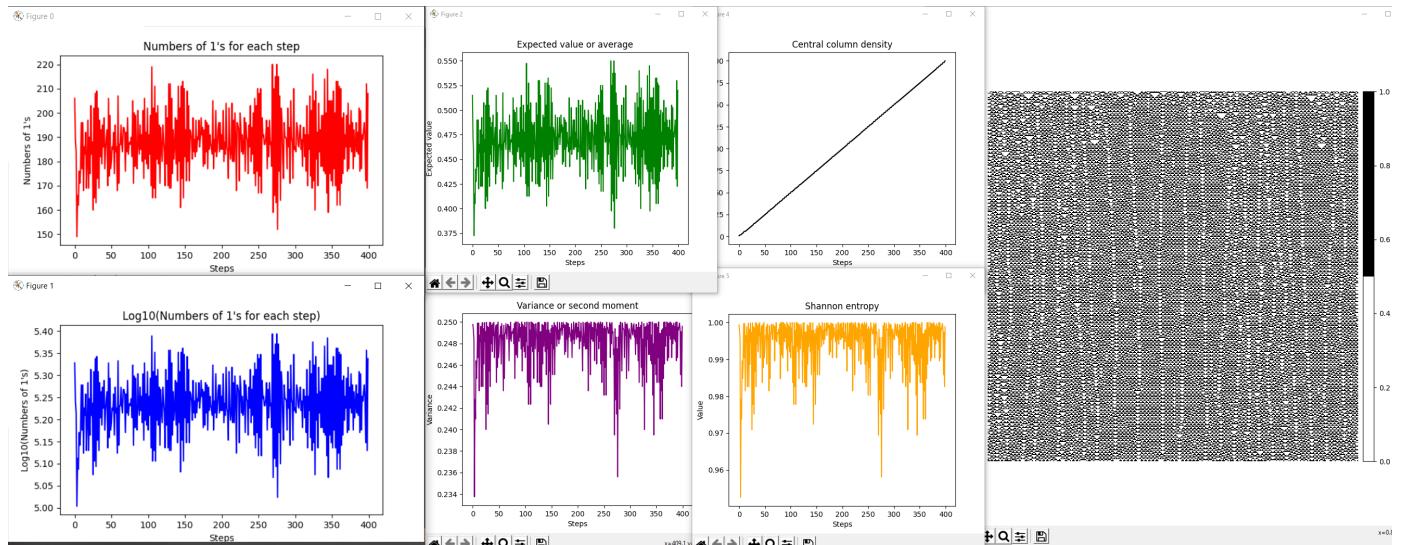


Figura 23: Autómata resultante con sus respectivas métricas.

En la figura 12 de igual manera que con la figura 10 podemos ver los valores de entrada de nuestro programa, en donde lo único que cambia es que en esta ocasión hacemos uso de la expresión regular de nuestra regla media la selección de la casilla con la etiqueta “Use regular expression (rule 22 & 54 only)”

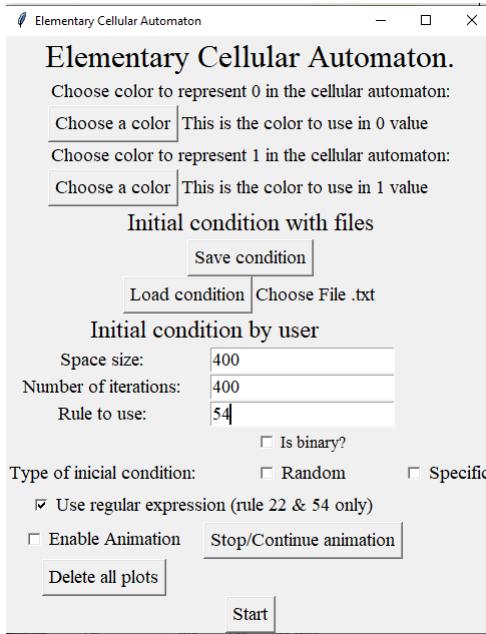


Figura 24: Entrada de nuestro usando nuestra expresión regular r22.

En la figura 13 vemos el autómata resultante así como las métricas que el programa nos genera.

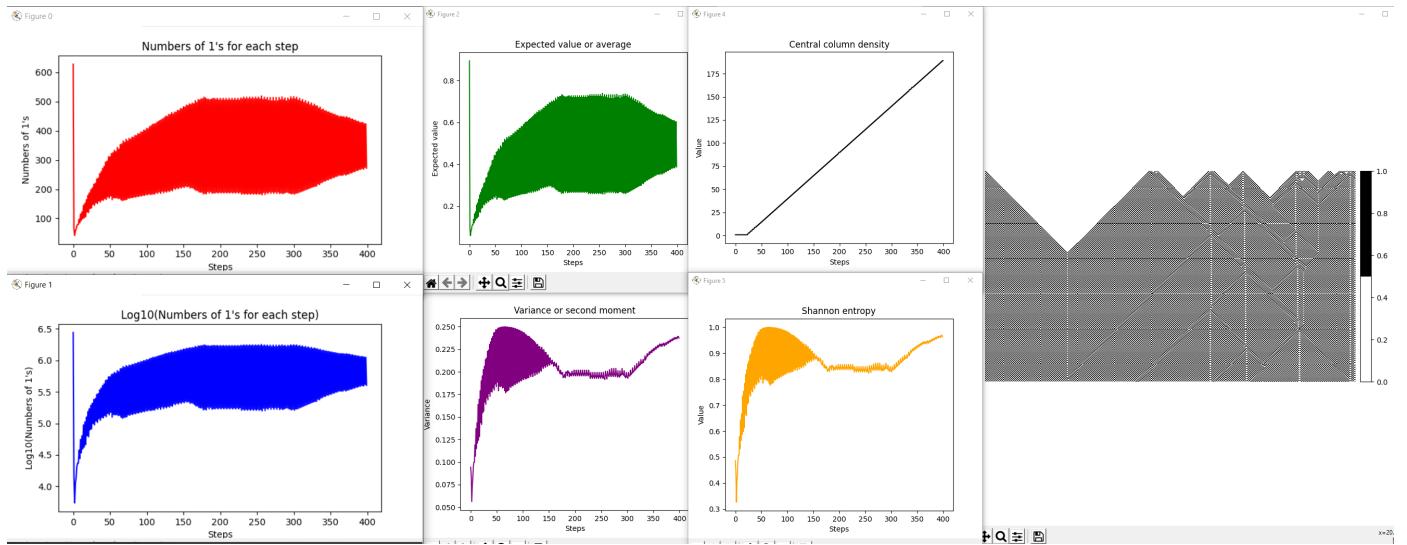


Figura 25: Autómata resultante con sus respectivas métricas.

Para nuestro autómata generado de manera aleatoria vemos como las gráficas son completamente diferentes a los autómatas generados con la misma regla anteriores, esto se debe a la generación aleatoria que tenemos y es que no tenemos control alguno, vemos como el numero de unos por paso tiene un mínimo global en el paso 3 con un valor de 149 1's por paso, y un máximo global en el paso 269 con un valor de 220 1's, si vemos la entropía de Shannon podemos ver que comenzamos con prácticamente la entropía máxima a diminuir a un mínimo global de 0.9525, que es relativamente poco, la entropía se mantiene prácticamente en el máximo por todos los pasos, esto nos vuelve a indicar que tenemos incertidumbre prácticamente máxima, examinando nuestra gráfica de la densidad en la columna central vemos un crecimiento prácticamente lineal en el numero de 1's pero hay que mencionar que volvemos a

ver partes en donde la gráfica de densidad es constante, finalizando el análisis de este autómata podemos decir que no tenemos un patrón reconocible.

Por otro lado tenemos al autómata generado por expresión regular y a simple vista podemos observar como volvemos a ver un comportamiento muy similar a los autómatas generados por la expresión regular. Observado las métricas que tenemos disponibles observamos como el numero de unos tiene un máximo de 652 unos a tener un mínimo de 35 para después incrementar y empezar a rondar los valores de 152 a 500 células con valor 1, también vemos como la densidad de la columna central al inicio tienen un crecimiento constante hasta el paso 23 para después crecer de una forma linea pero volvemos a ver como hay partes en donde es constante, estudiando la entropía podemos ver que es ligeramente diferentes a las anteriormente estudiadas, aunque el inicio es similar, empezamos de valores bajos para poco después subir prácticamente al máximo, oscilar entre el valor máximo y 0.775, para después no oscilar tanto y al final, en este autómata, para que la entropía crece entre la iteración 300 y la 400 llegando a un valor de 0.965, esto quiere decir que en nuestro sistema empezamos con una baja entropía para después oscilar entre valores altos de entropía, esto nos dice que somos incapaces de predecir el siguiente estado nuestro autómata y tenemos una alta incertidumbre.

Una vez visto estas tres pruebas podemos decir que la expresión regular nos genera autómatas comunes, sin embargo, en la prueba del 95 % vemos como se nos genera un autómata muy similar a los generados por la expresión regular, al igual es similar en las métricas que se nos genera, esto nos quiere decir que nuestros autómatas que genera la expresión regular tienen una gran cantidad de 1's, volvemos a ver como el uso de la opción de generar una condición completamente aleatoria nos genera resultados completamente diferentes tanto a la expresión regular como al porcentaje de unos.

3. Atractores

Para el desarrollo de esta parte se uso Python como lenguaje de programación, los atractores son guardados en carpetas con el nombre de regla, así también, el nombre del archivo que se genera se guarda con el siguiente formato: atractor_’numero de la regla’_size_’longitud’.graphml, para poder visualizar los resultados se necesita de un programa capaz de abrir archivos con formato .graphml, hay varias opciones gratis para diferentes sistemas operativos, las mas conocidas son las siguientes:

1. Windows

- a) Gephi. Gratuito
- b) yWorks yEd Graph Editor. Gratuito
- c) Cytoscape. Gratuito

2. MacOS

- a) Gephi. Gratuito
- b) yWorks yEd Graph Editor. Gratuito

3. Linux

- a) Gephi. Gratuito
- b) yWorks yEd Graph Editor. Gratuito

Esto con el fin de proporcionar la capacidad de modificar las formas de los nodos, los vértices, colores, forma de organización entre algunas otras opciones mas. En nuestro caso se uso el programa Cytoscape el cual nos proporciona las características anteriormente mencionadas y algunas otras metras. Empecemos así con los atractores.

3.1. Regla 22

El formato sera el siguiente, primero se pondrá la imagen del atractor y después evoluciones de los nodos para ver los atractores que existen.

3.1.1. Tamaño 2

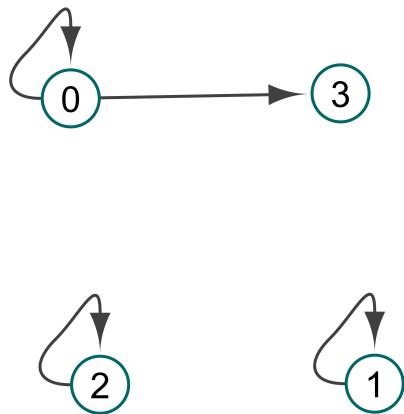


Figura 26: Atractor de tamaño 2.

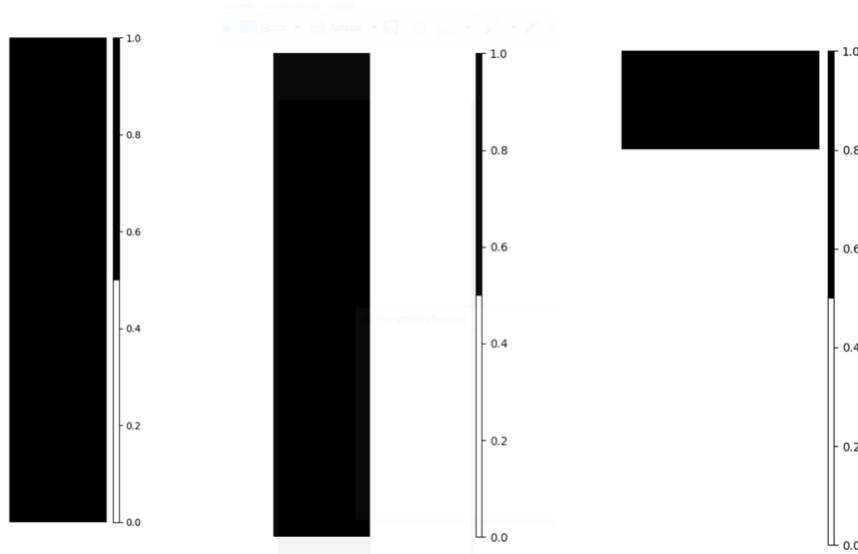


Figura 27: Evoluciones de los atractores.

Mencionar que en la imagen 27 y durante algunos tamaños las evoluciones serán acomodadas tal que del lado izquierdo se tenga el nodo de menor numero al mayor, ya que si lo hicieramos por cada nodo simplemente no acabaríamos, mencionar que para este atractor no vemos el caso del nodo 0 ya que el nodo 3 al dirigirse al nodo 0 entra en un ciclo en donde vemos el comportamiento del nodo 0.

3.1.2. Tamaño 3

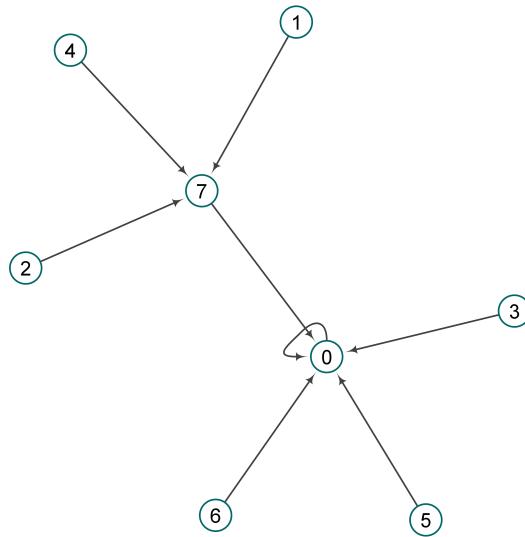


Figura 28: Atractor de tamaño 3.

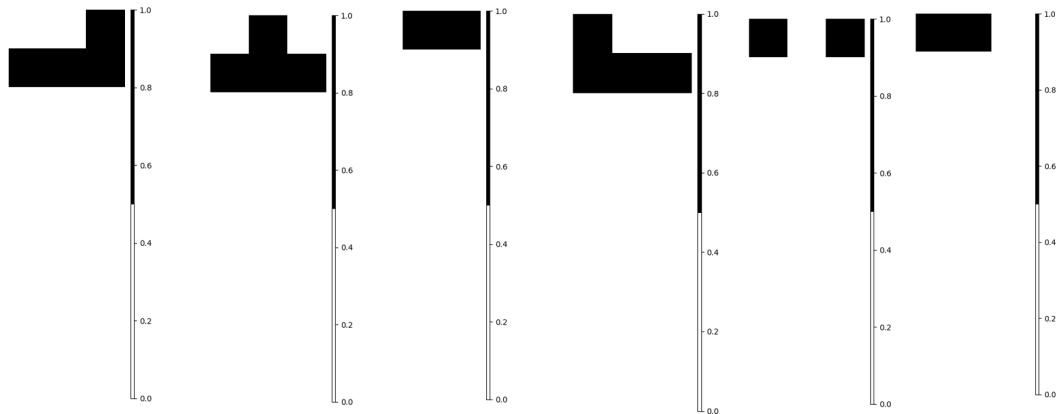


Figura 29: Evoluciones de los atractores.

Mencionar que para este atractor no vemos el caso del nodo 0 ya que el nodo 7, 3, 5, 6 se dirigen al nodo 0 el cual entra en un ciclo en donde vemos el comportamiento del nodo 0.

3.1.3. Tamaño 4

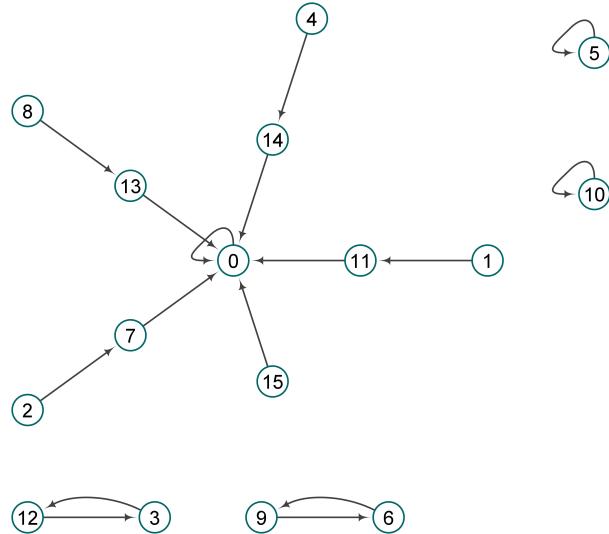


Figura 30: Atractor de tamaño 4.

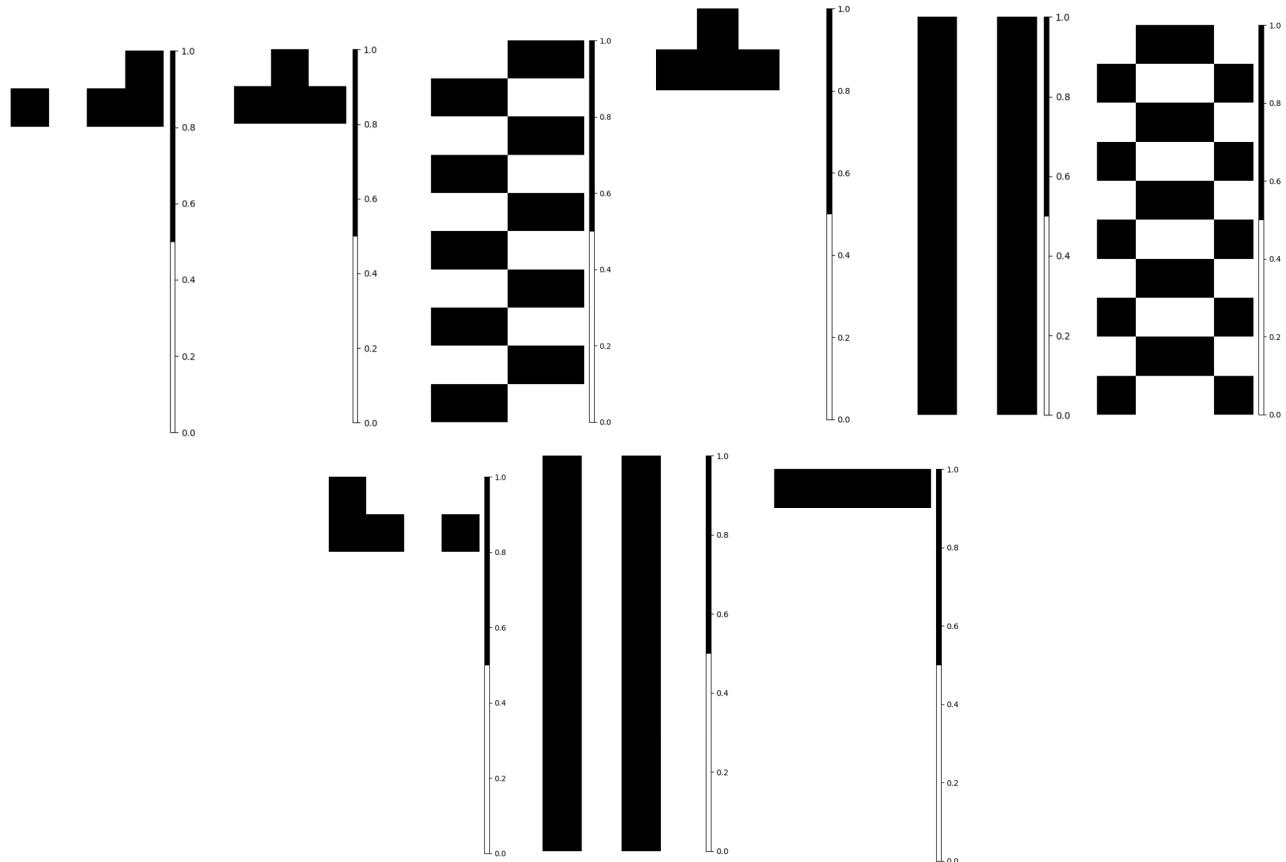


Figura 31: Evoluciones de los atractores.

Mencionar que para este atractor no vemos el caso del nodo 0, 13, 14, 7, 12 y 9 ya se pueden ver sus evoluciones por medio de otros nodos, por ejemplo, para ver la evolución del nodo 12 hacemos uso del nodo 3 ya que es un ciclo entre ambos, en el caso del nodo 9 y 6 ocurre lo mismo, mientras que los nodos 8, 4, 1, 2 y 15 acceden al nodo 0, los 4 primeros pasan por otros nodos, los cuales son 13, 14, 11 y 7 respectivamente.

3.1.4. Tamaño 5

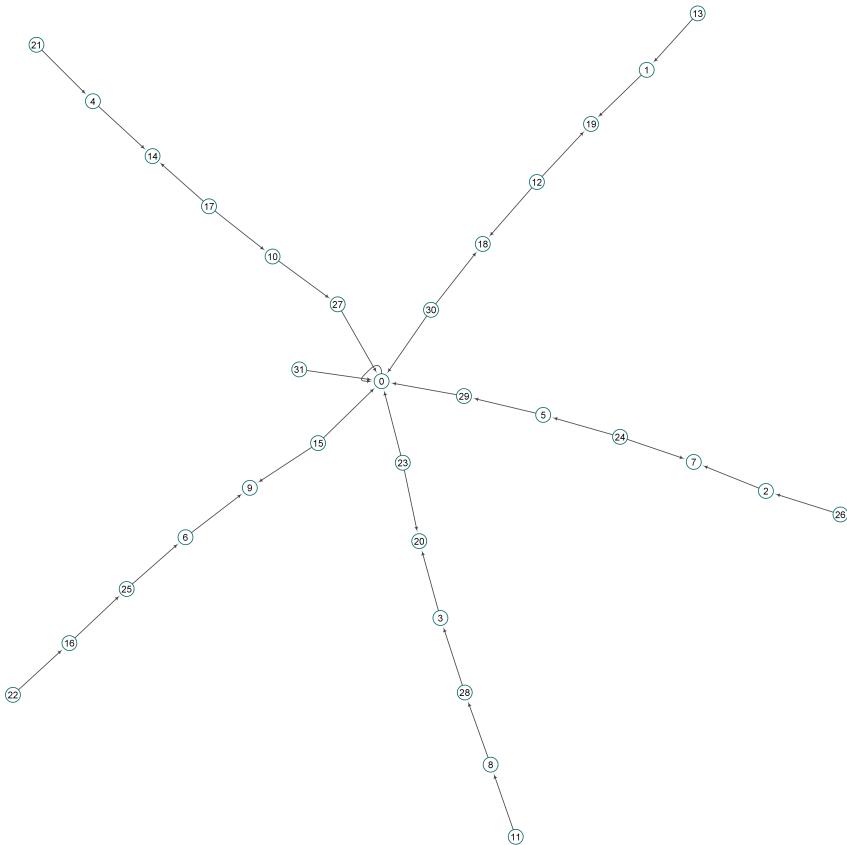


Figura 32: Atractor de tamaño 5.

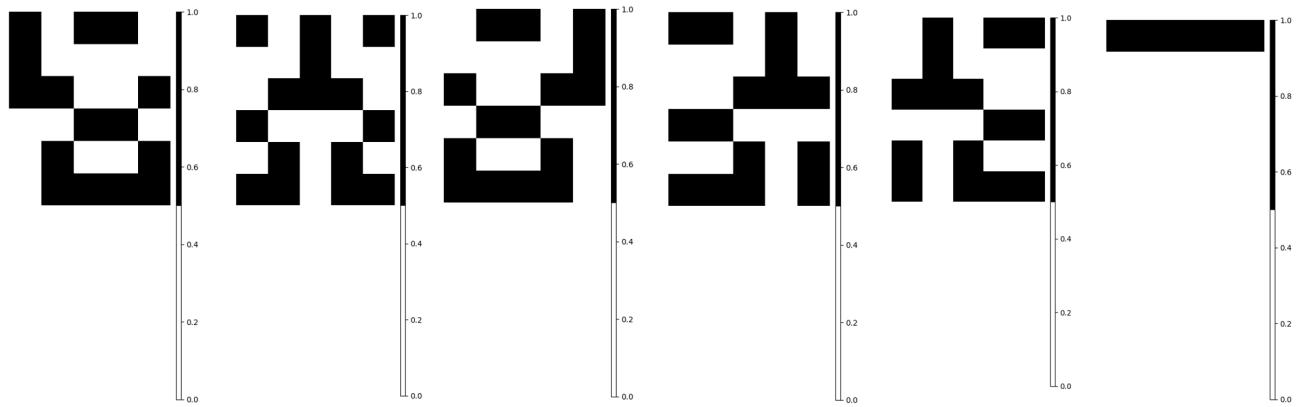


Figura 33: Evoluciones de los atractores.

Mencionar que para este atractor solamente vemos la evolución de los nodos 11, 22, 31, 21, 13 y 26, ya que son los nodos del exterior y estos pasan por los demás nodos y podemos ver como funciona el atractor el cual es el nodo 0.

3.1.5. Tamaño 6

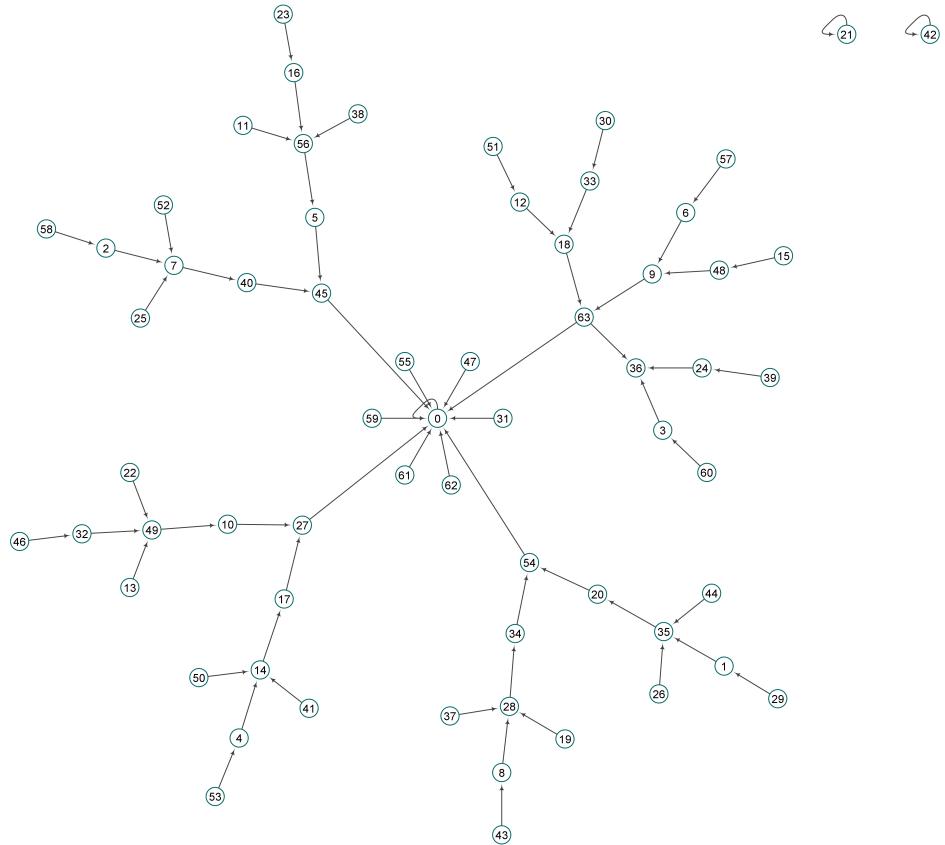


Figura 34: Atractor de tamaño 6.

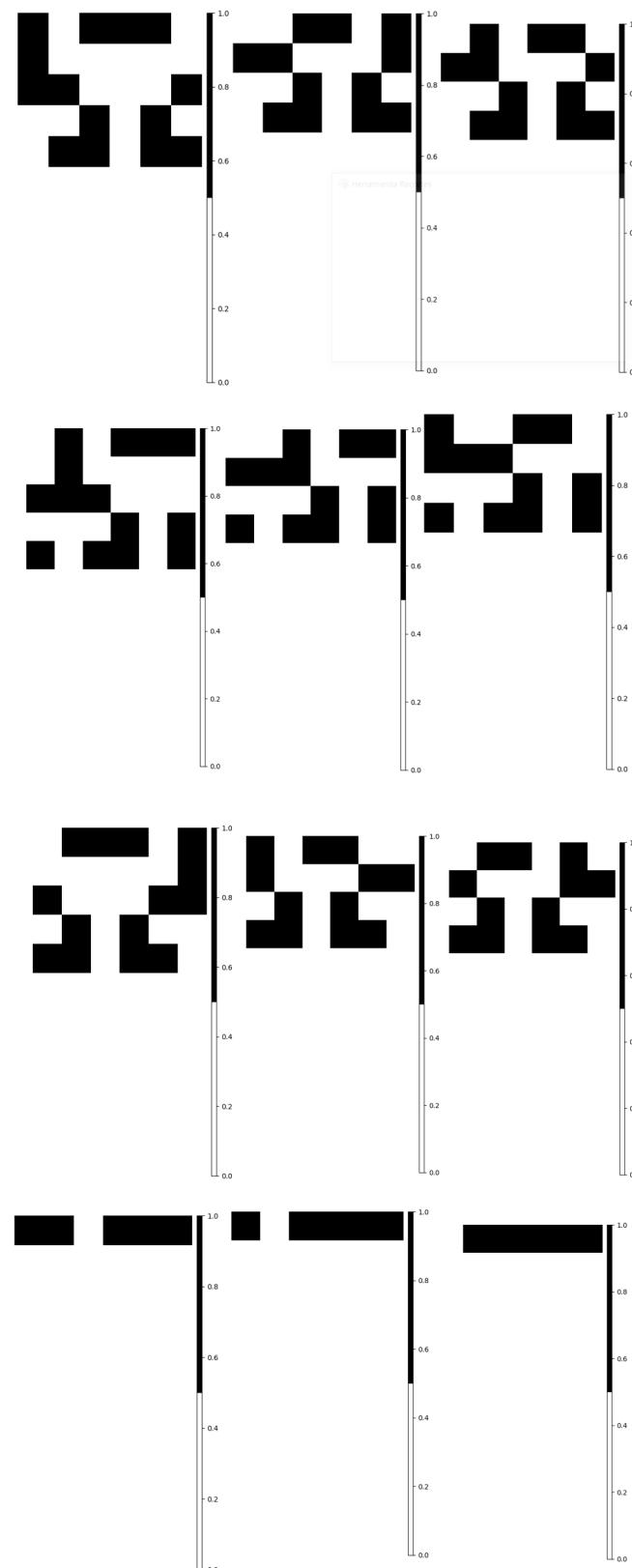


Figura 35: Evoluciones de los atractores.

Mencionar que a partir de este atractor solamente tomaremos unos cuantos nodos de prueba ya que abarcar la mayoría tomaría mucho espacio del reporte.

3.1.6. Tamaño 7

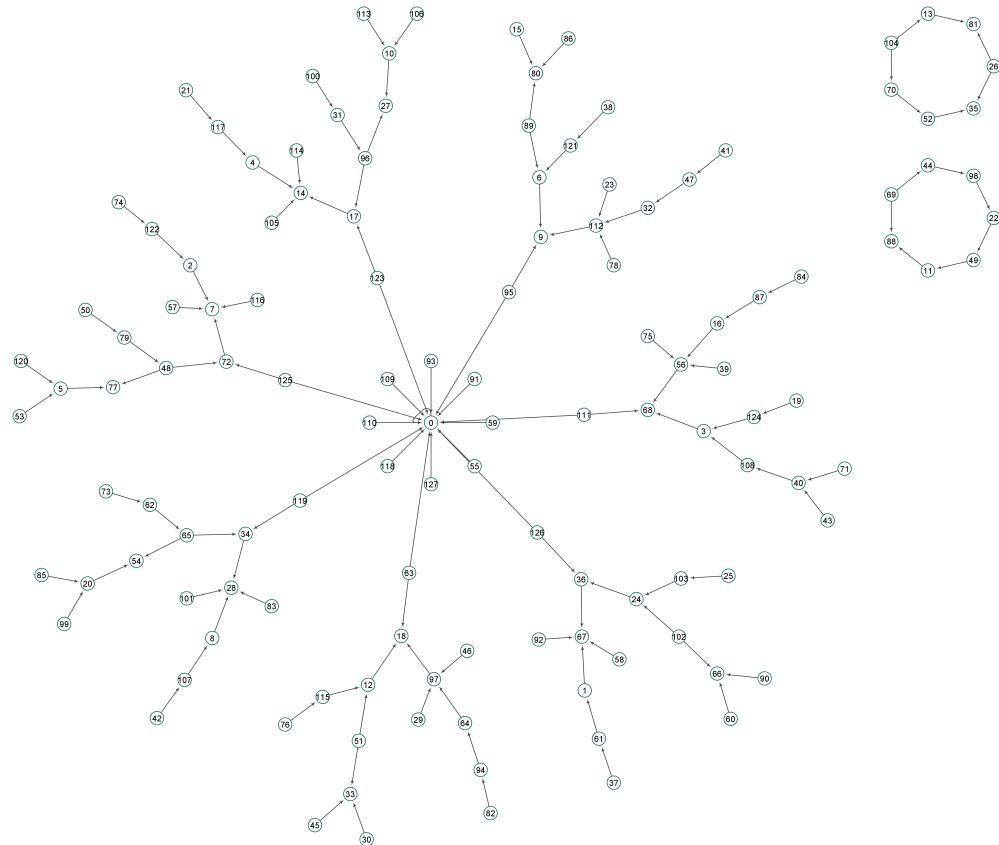


Figura 36: Atractor de tamaño 7.

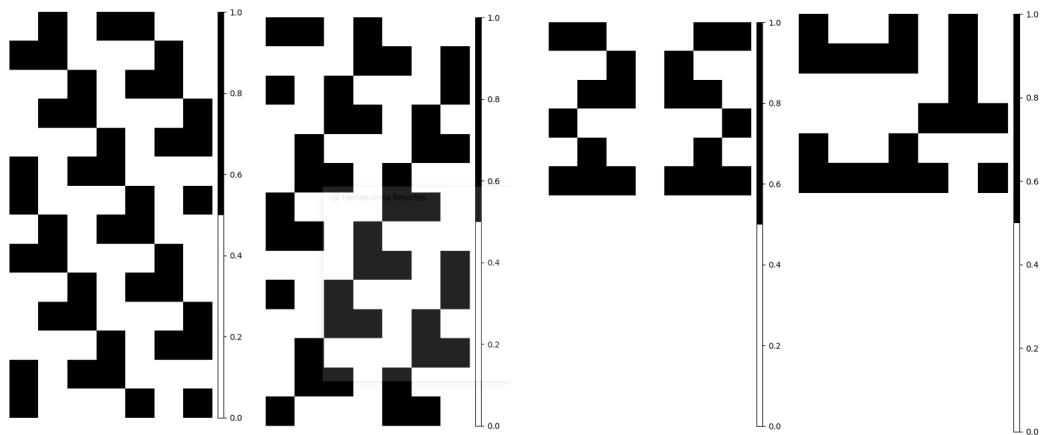


Figura 37: Evoluciones de los atractores.

Los nodos de prueba fueron 104, 44, 99 y 73, ya que empezamos a ver un patrón repetitivo de tener nodos que se dirigen directamente al atractor y algunos nodos que forman ciclos con otros nodos.

3.1.7. Tamaño 8

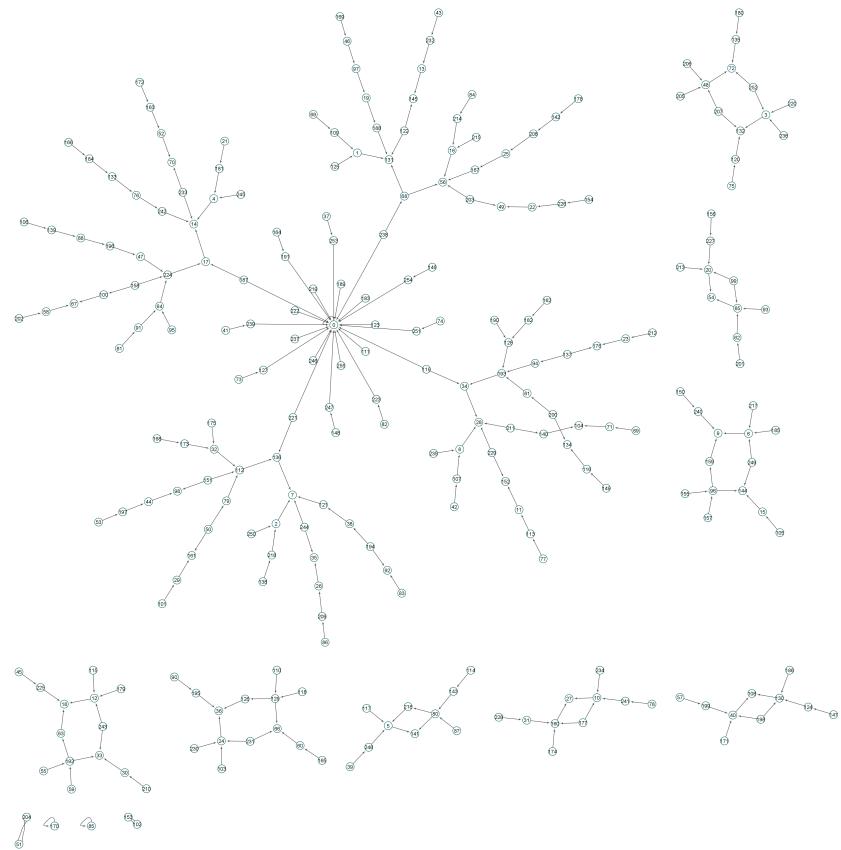


Figura 38: Atractor de tamaño 8.

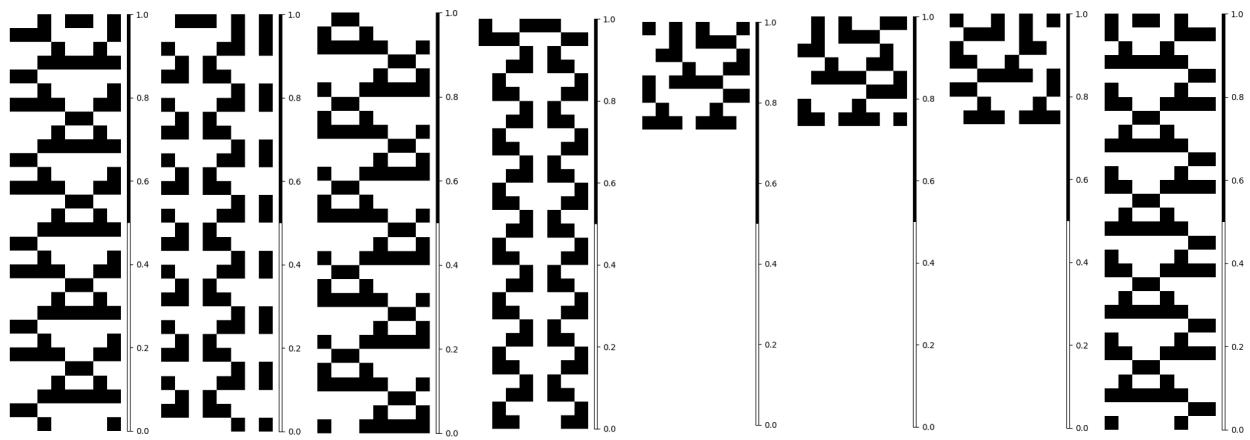


Figura 39: Evoluciones de algunos atractores.

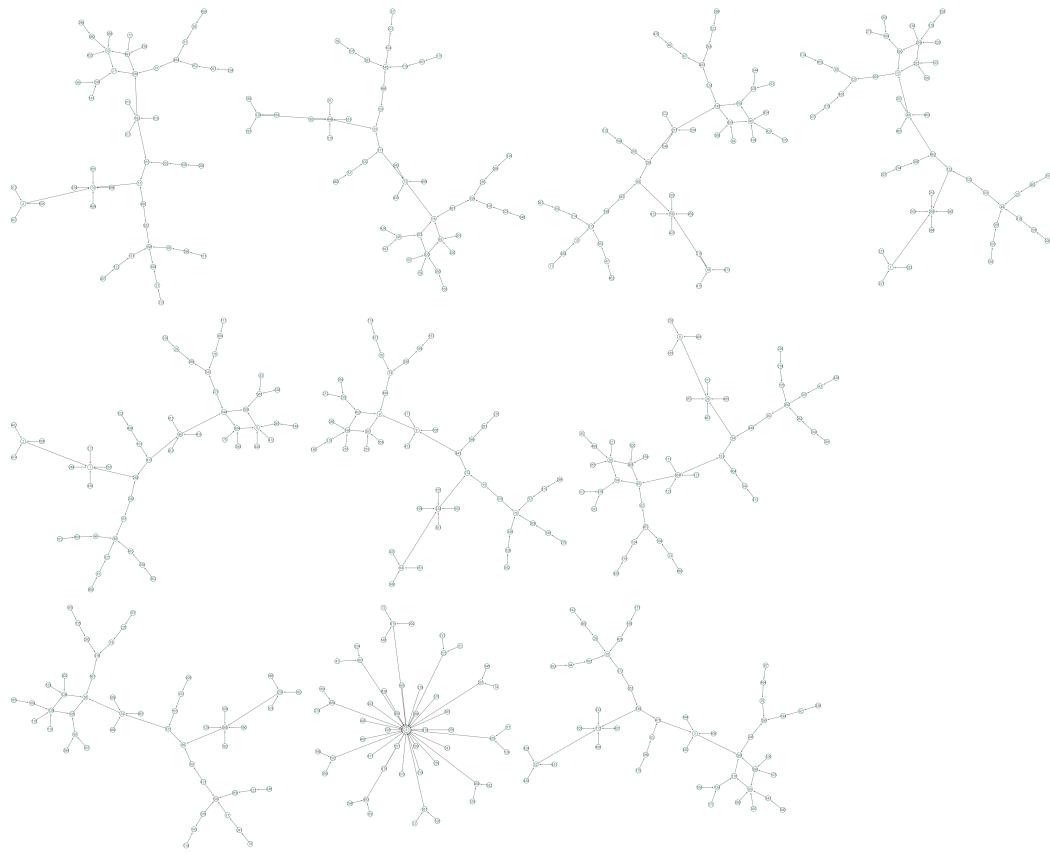
3.1.8. Tamaño 9

Figura 40: Atractor de tamaño 9.

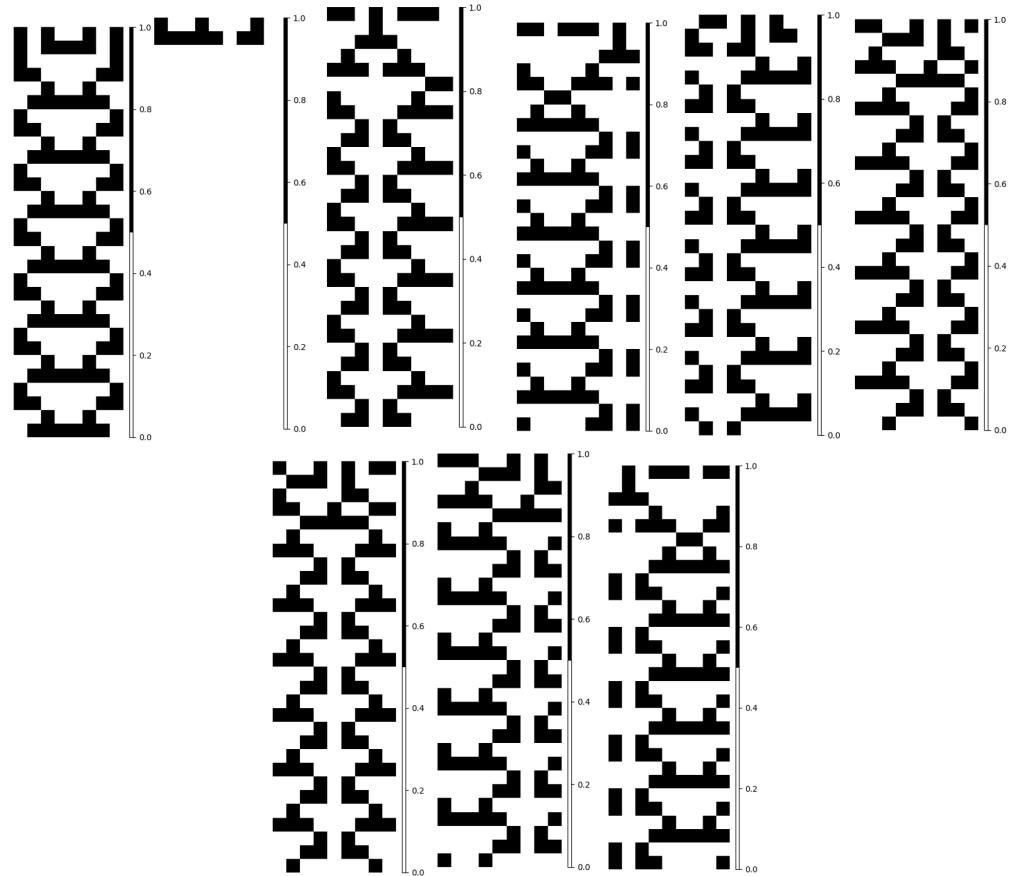


Figura 41: Evoluciones de los atractores.

3.1.9. Tamaño 10

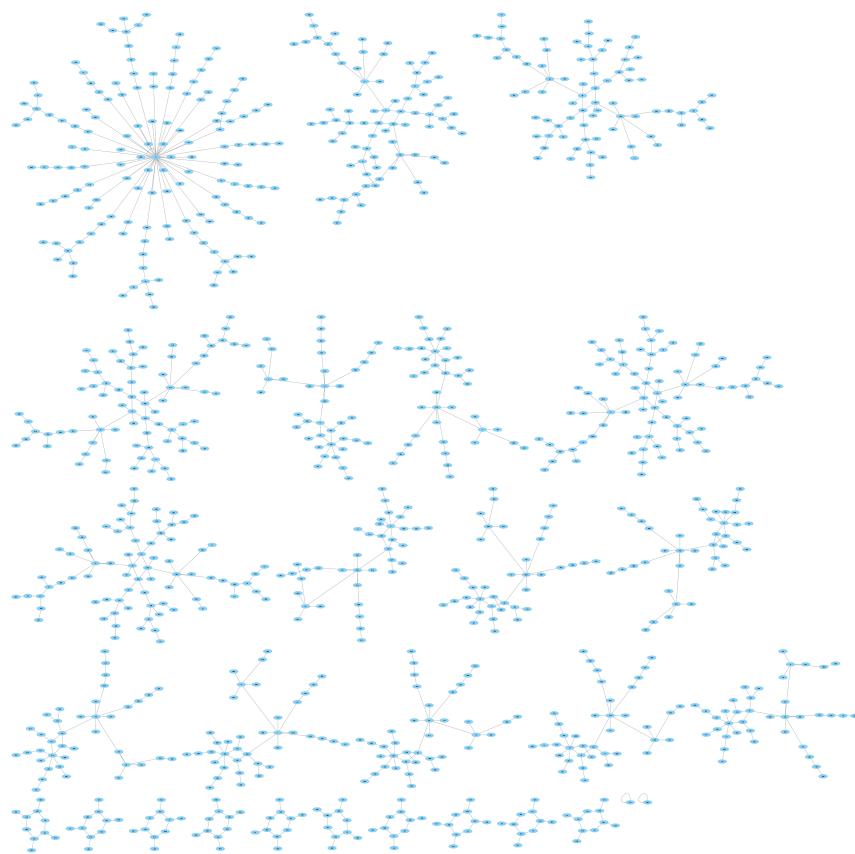


Figura 42: Atractor de tamaño 10.

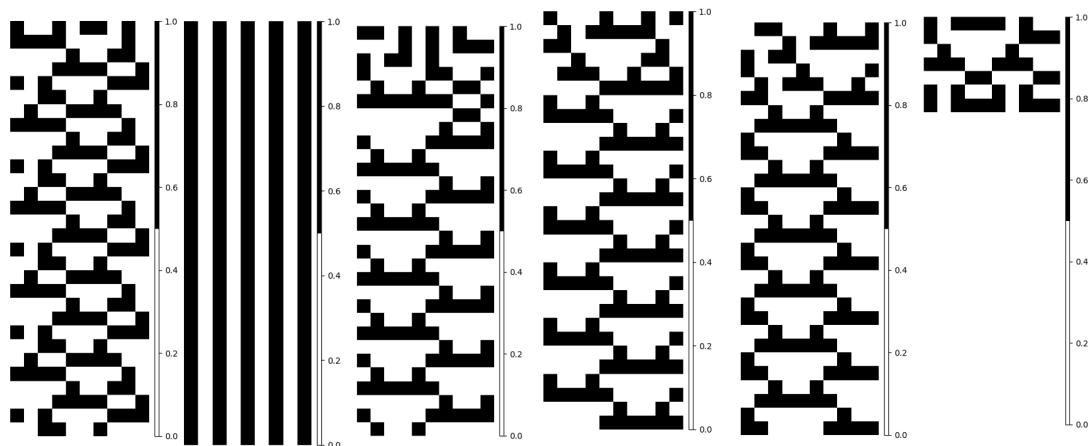


Figura 43: Evoluciones de los atractores.

3.1.10. Tamaño 11

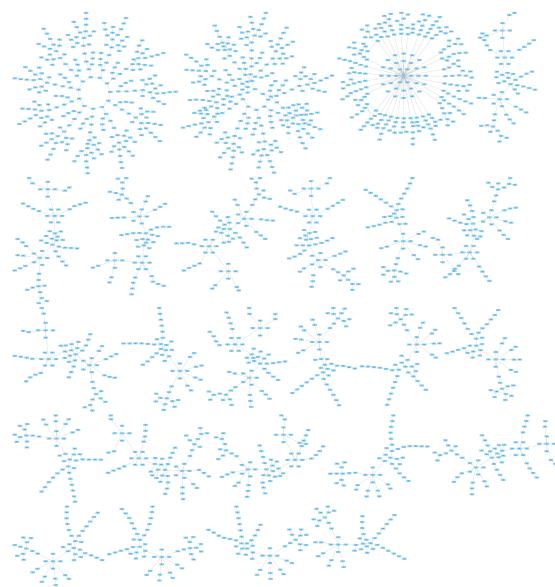


Figura 44: Atractor de tamaño 11.

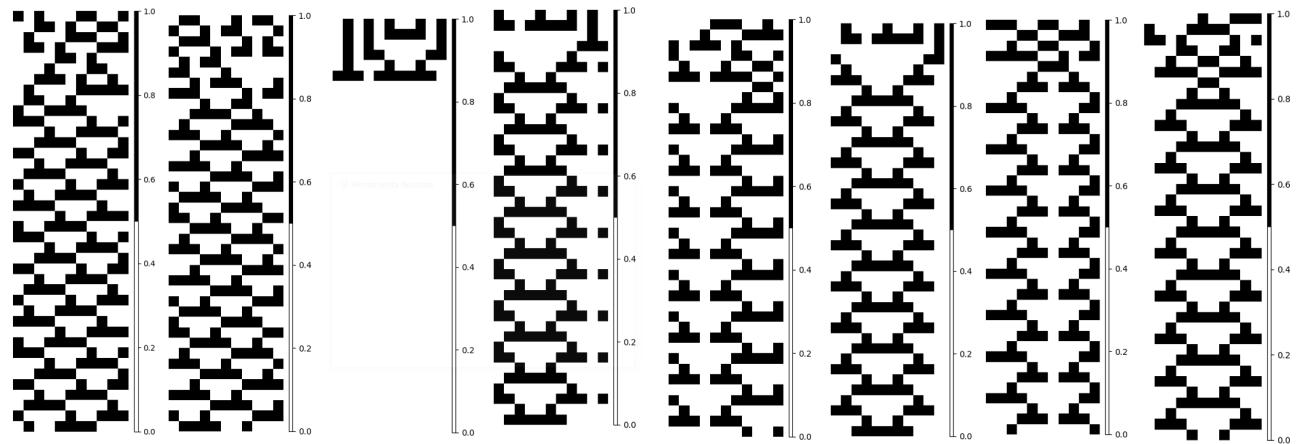


Figura 45: Evoluciones de los atractores.

3.1.11. Tamaño 12

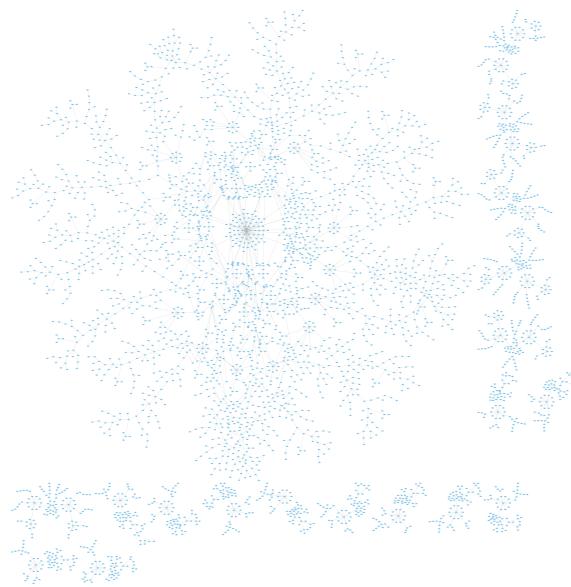


Figura 46: Atractor de tamaño 12.

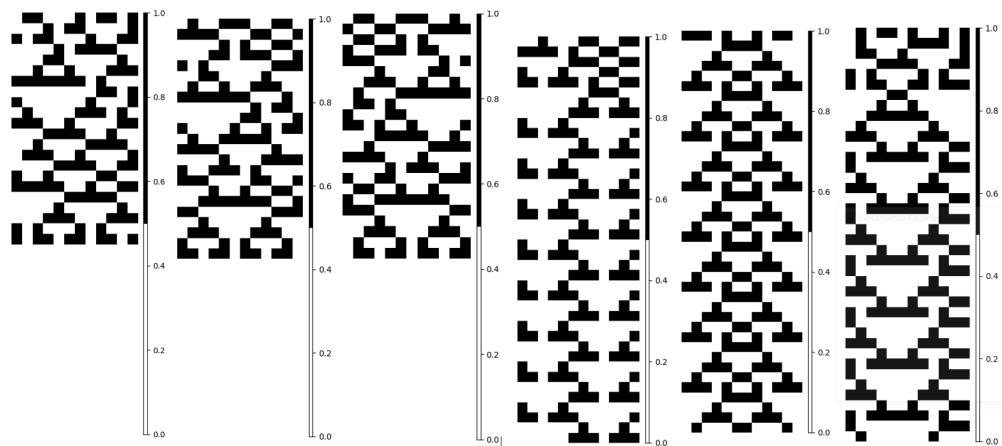


Figura 47: Evoluciones de los atractores.

3.1.12. Tamaño 13

Figura 48: Atractor de tamaño 13.

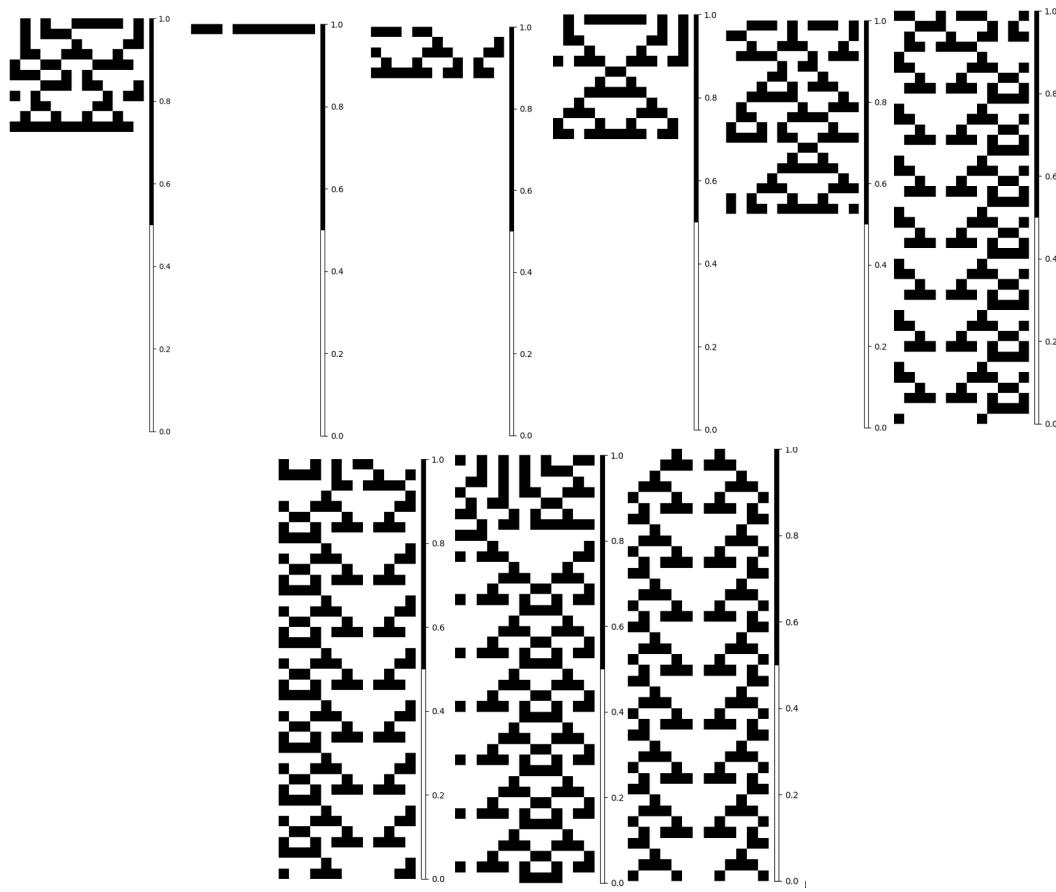


Figura 49: Evoluciones de los atractores.

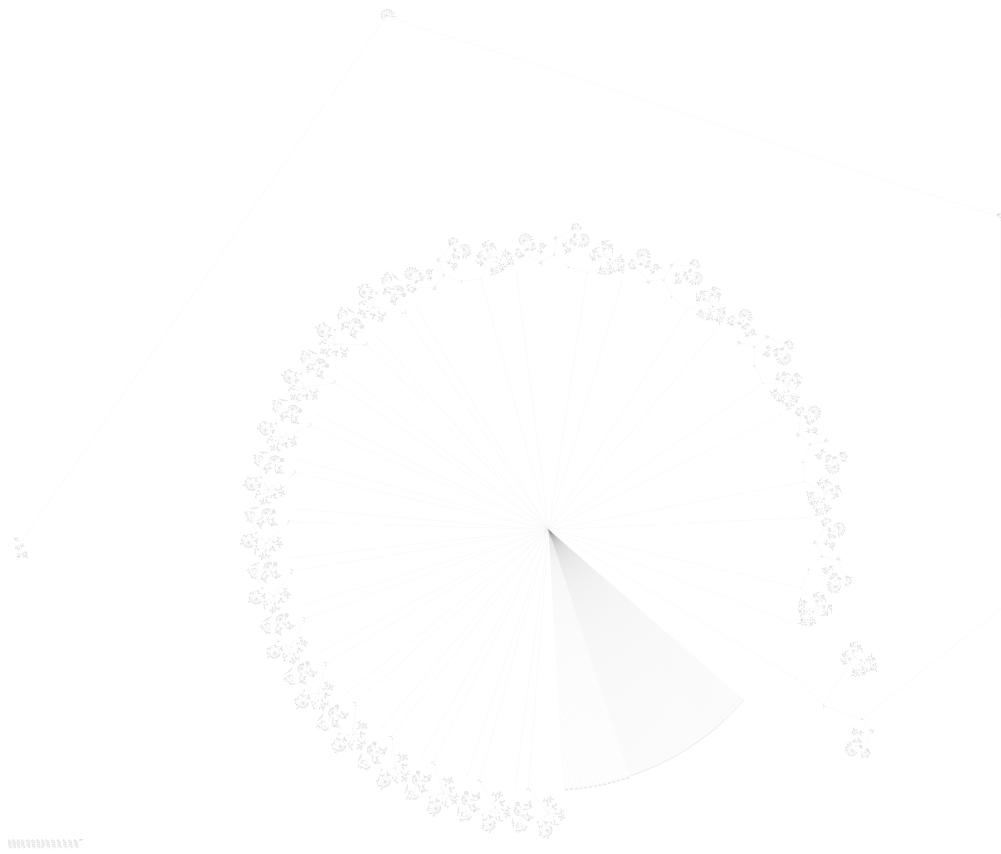
3.1.13. Tamaño 14

Figura 50: Atractor de tamaño 14.

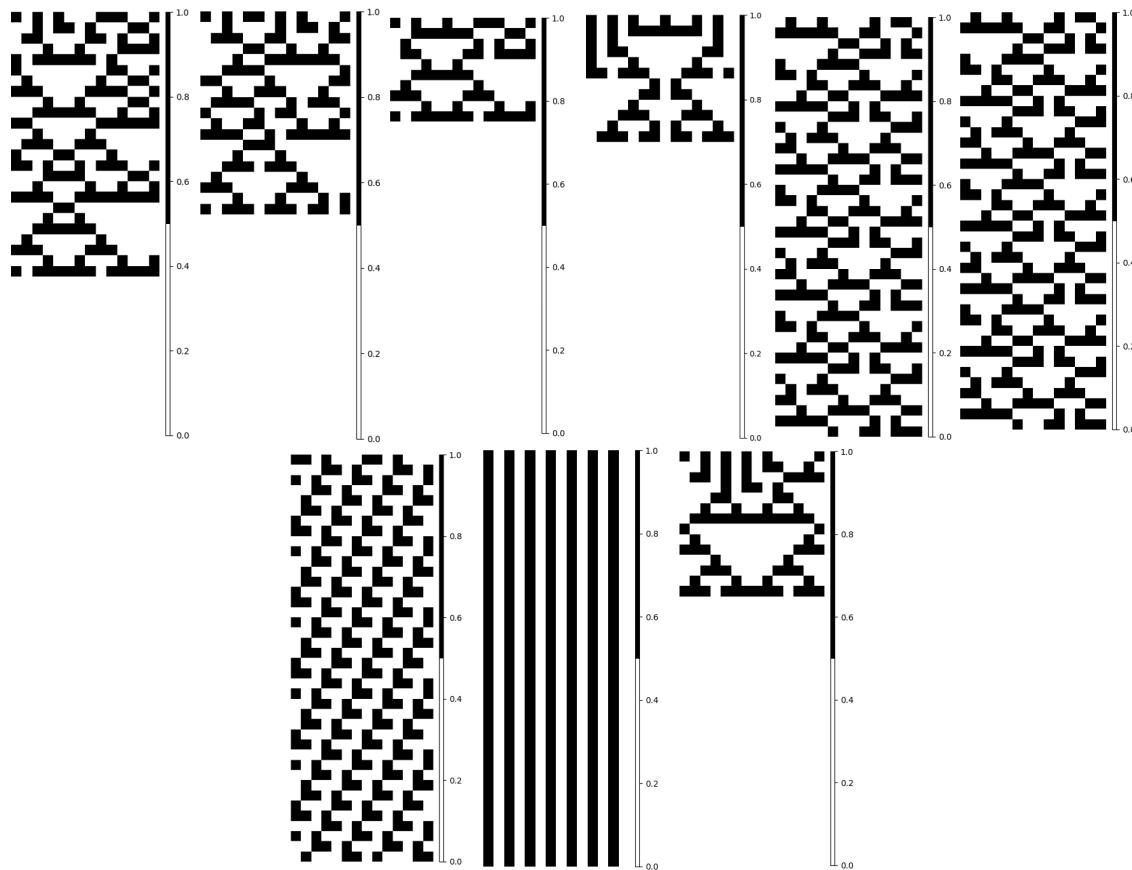


Figura 51: Evoluciones de los atractores.

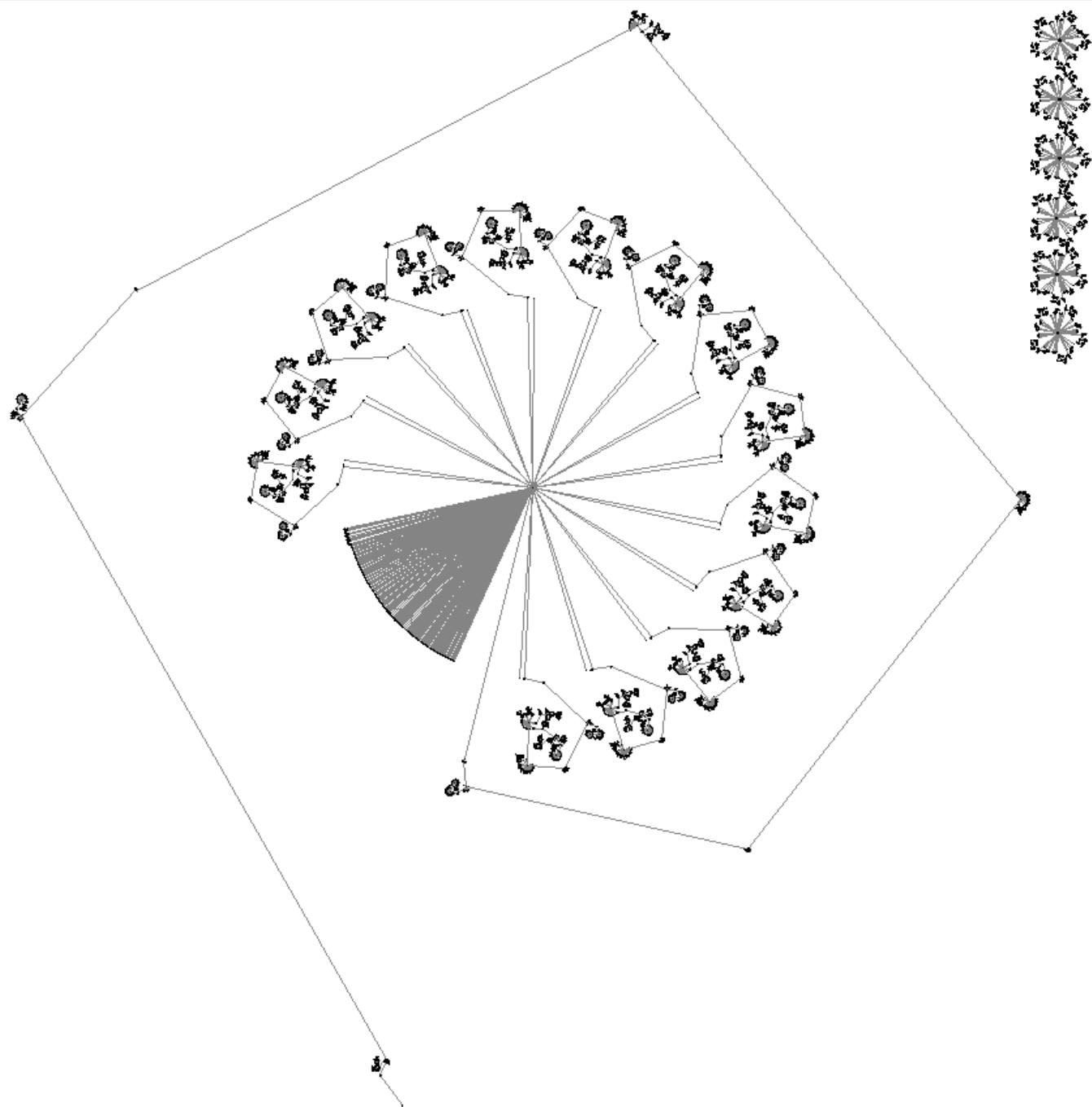
3.1.14. Tamaño 15

Figura 52: Atractor de tamaño 15.

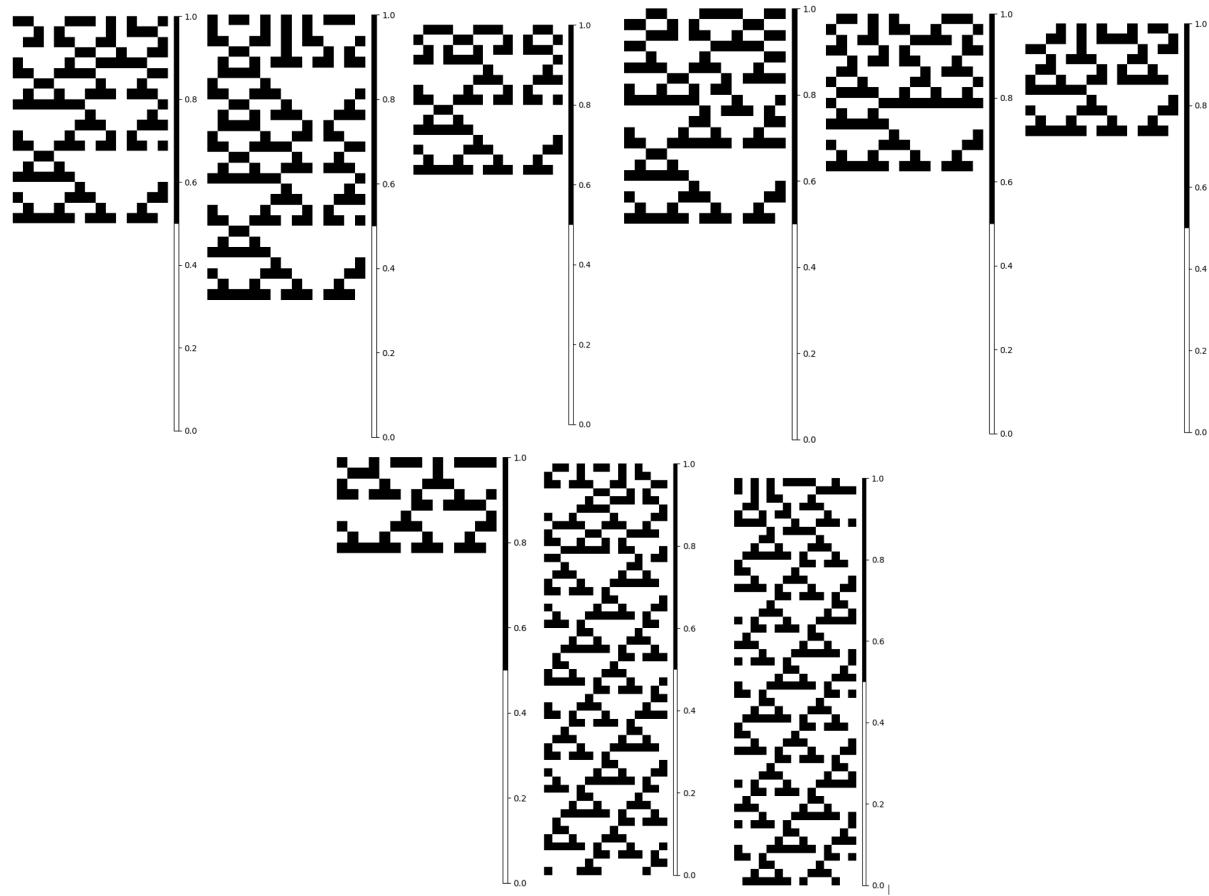


Figura 53: Evoluciones de los atractores.

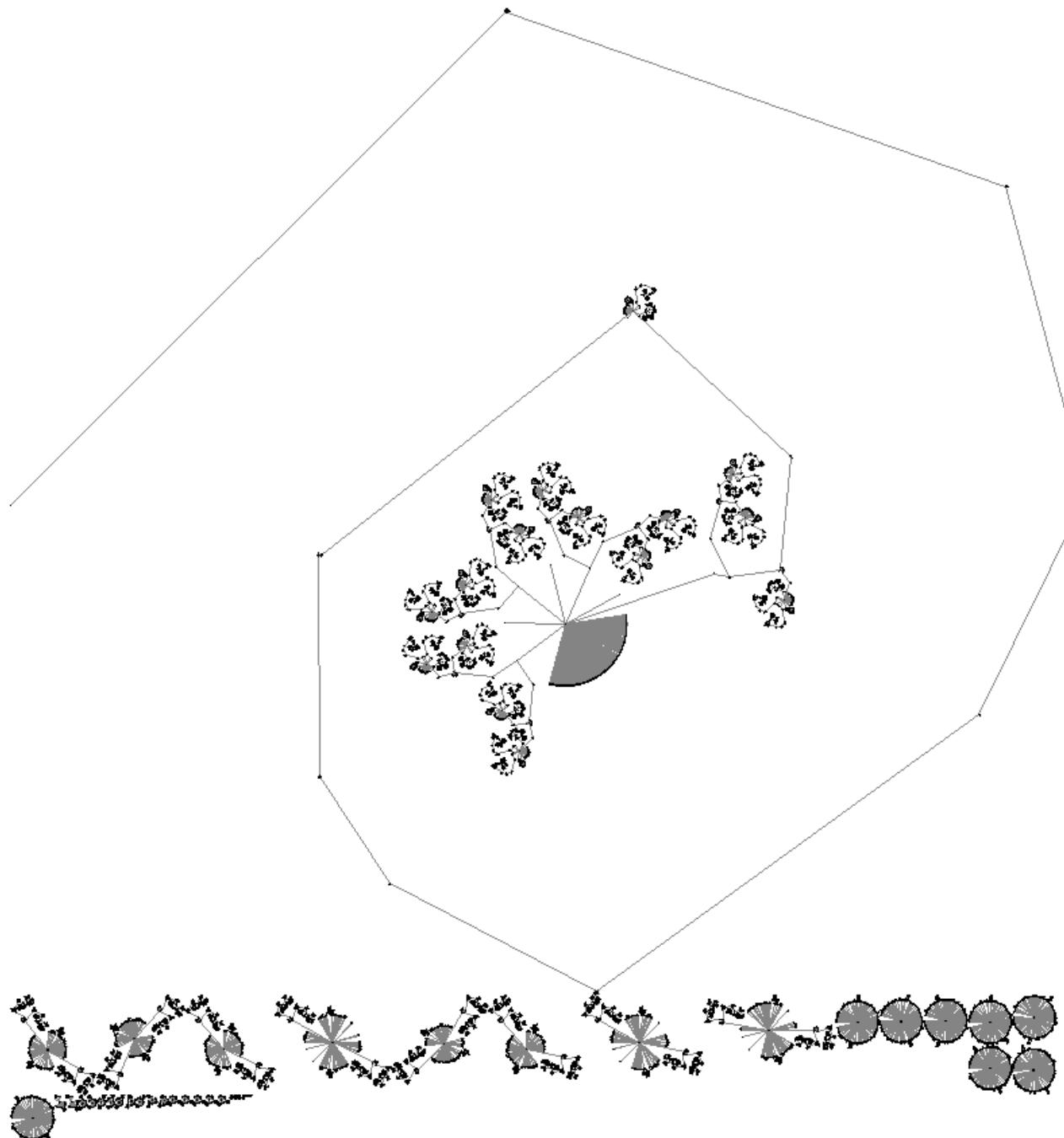
3.1.15. Tamaño 16

Figura 54: Atractor de tamaño 16.

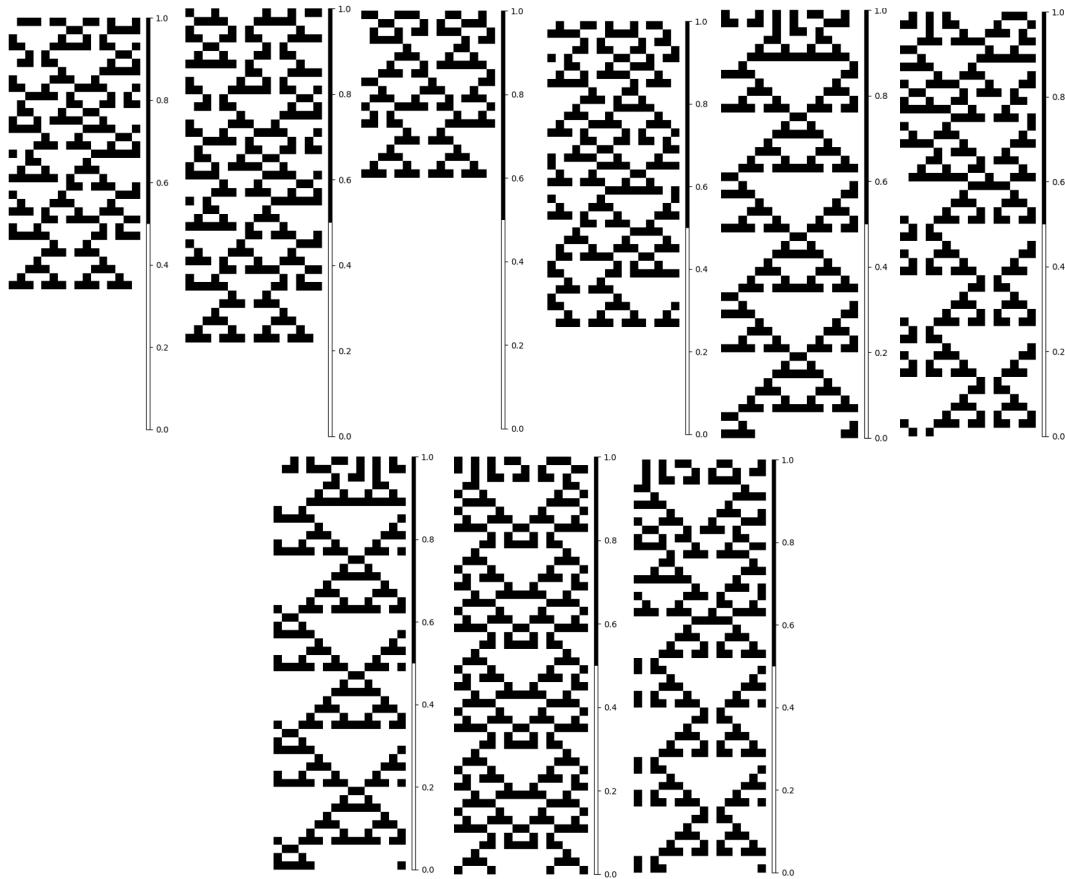


Figura 55: Evoluciones de los atractores.

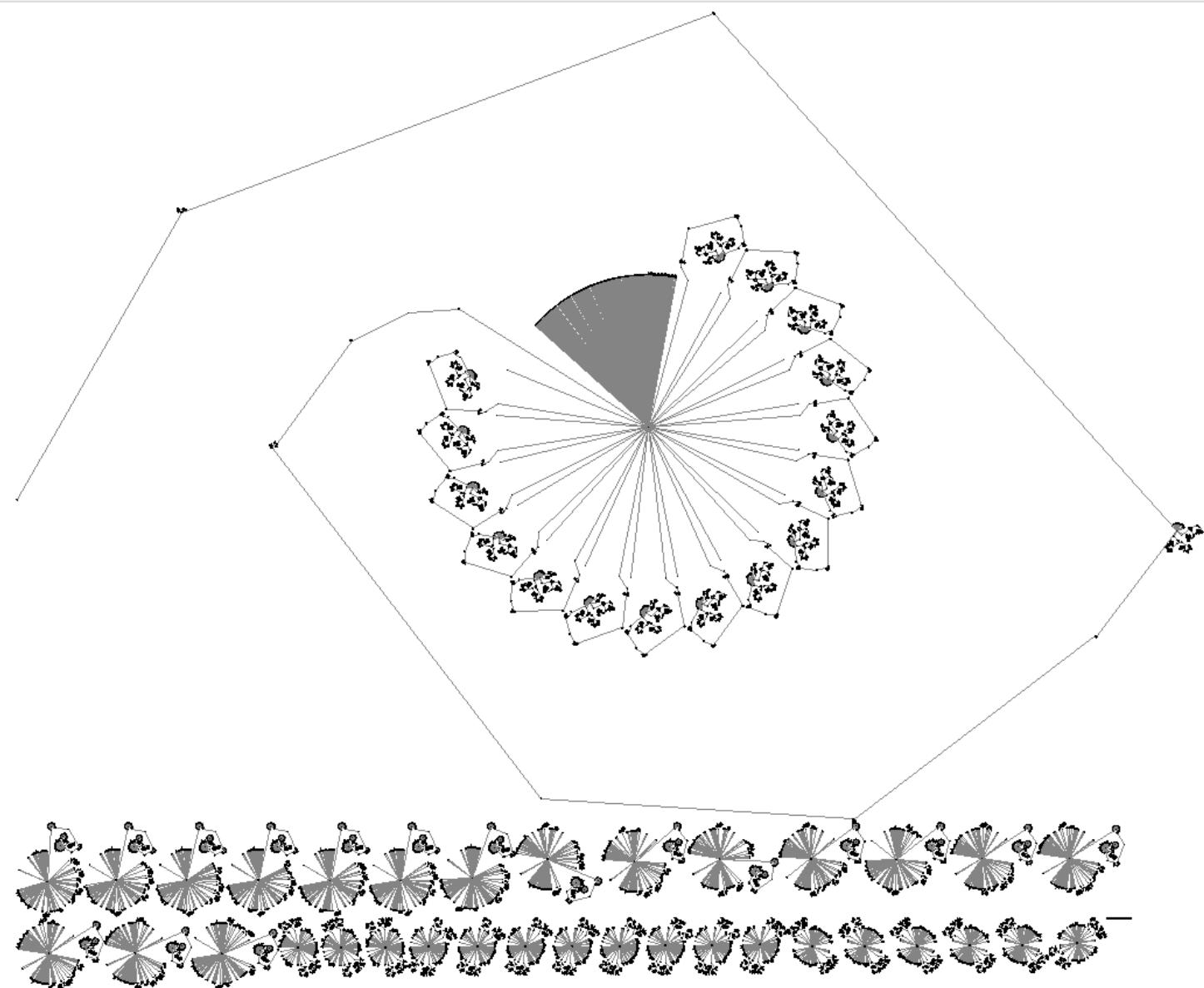
3.1.16. Tamaño 17

Figura 56: Atractor de tamaño 17.

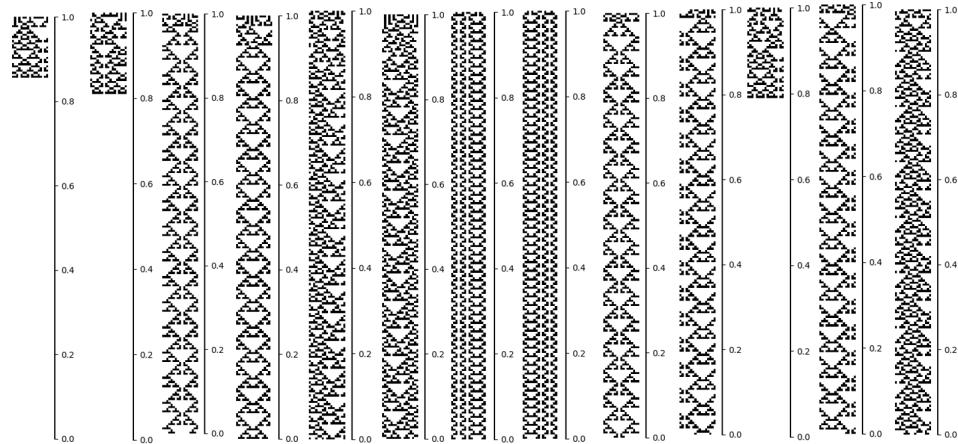


Figura 57: Evoluciones de los atractores.

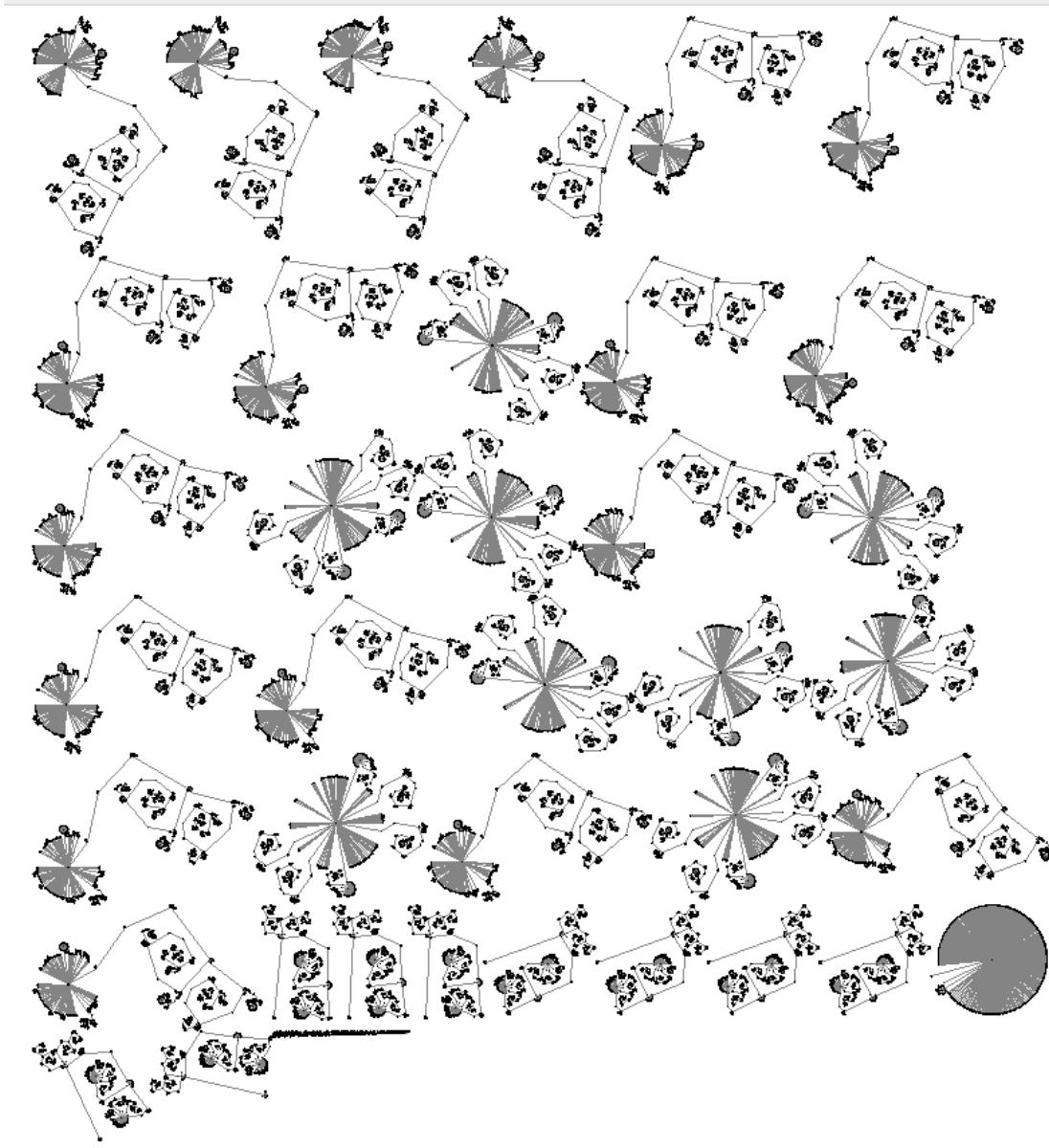
3.1.17. Tamaño 18

Figura 58: Atractor de tamaño 18.

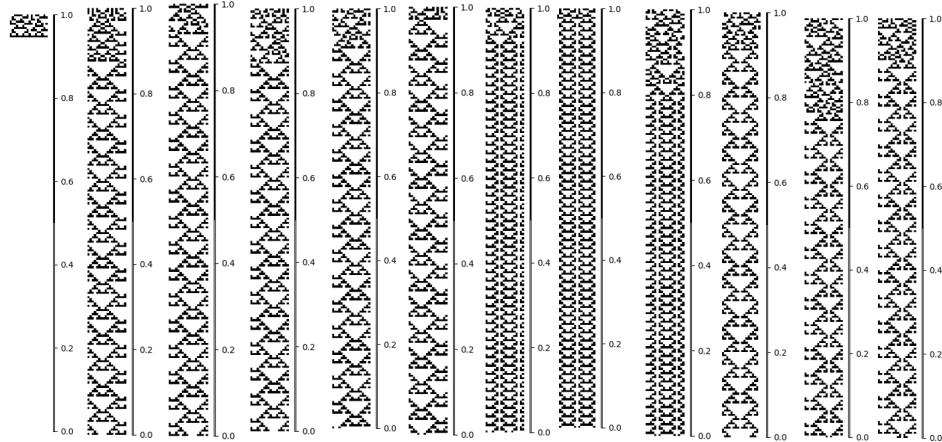


Figura 59: Evoluciones de los atractores.

Este fue el ultimo atractor que fue capaz de construir nuestro programa, sin embargo, mediante el uso de JetBrains Datalore, el cual nos sirve como una maquina externa que nos da mas poder de computo pudimos calcular hasta el tamaño 25, esto solo es guardado como archivos .txt que contienen los nodos y sus atractor correspondiente.

3.1.18. Tamaño 19

Para estos tamaños en donde solamente calculamos los atractores pero no calculamos la gráfica debido a que el incremento de nodos y por lo tanto del archivo es demasiado grande, solo mencionar que el archivo mas grande obtenido pesa medio Gb. Solo dos capturas de pantalla, el principio y el fin y algunas evoluciones con valores tomados aleatoriamente.

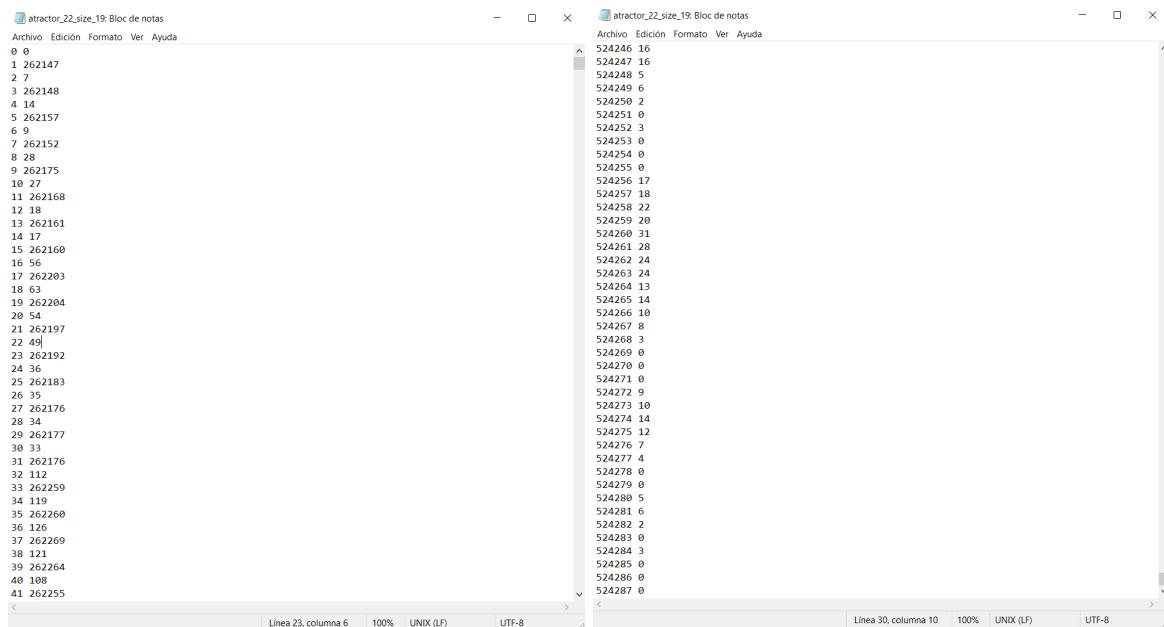


Figura 60: Atractor de tamaño 19.

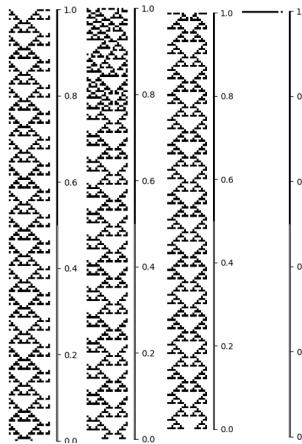


Figura 61: Evoluciones de los atractores.

3.1.19. Tamaño 20

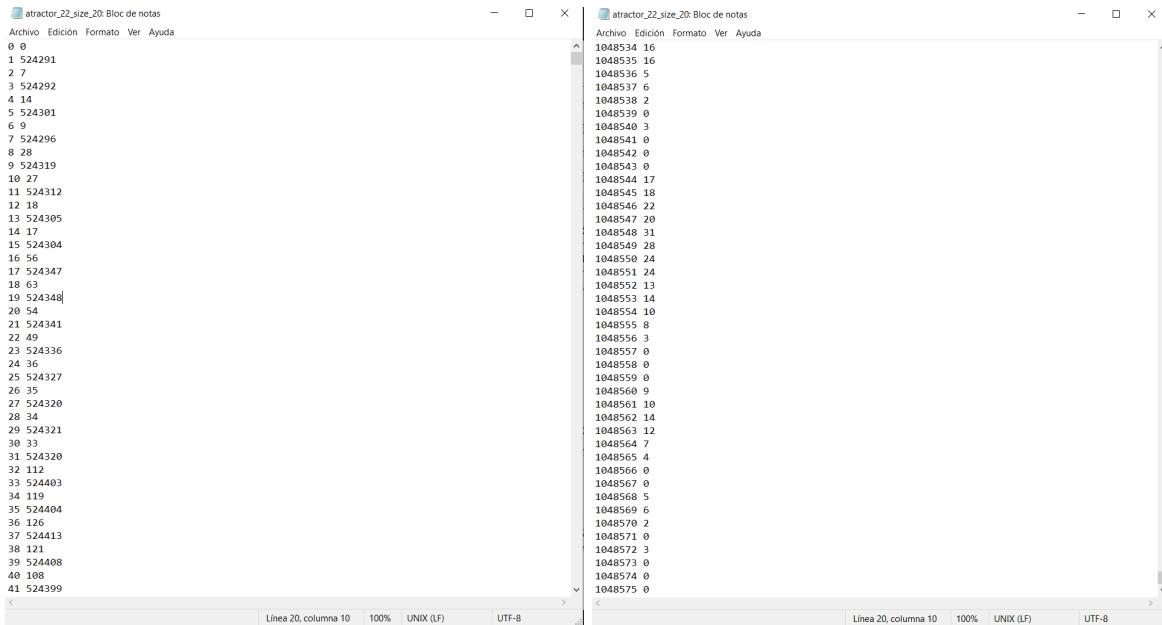


Figura 62: Atractor de tamaño 20.

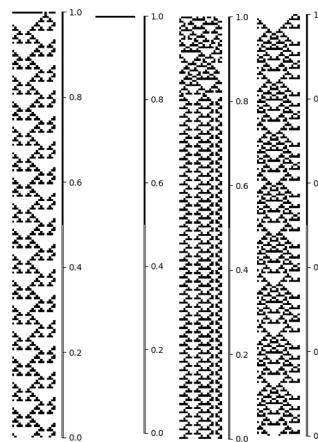


Figura 63: Evoluciones de los atractores.

3.1.20. Tamaño 21

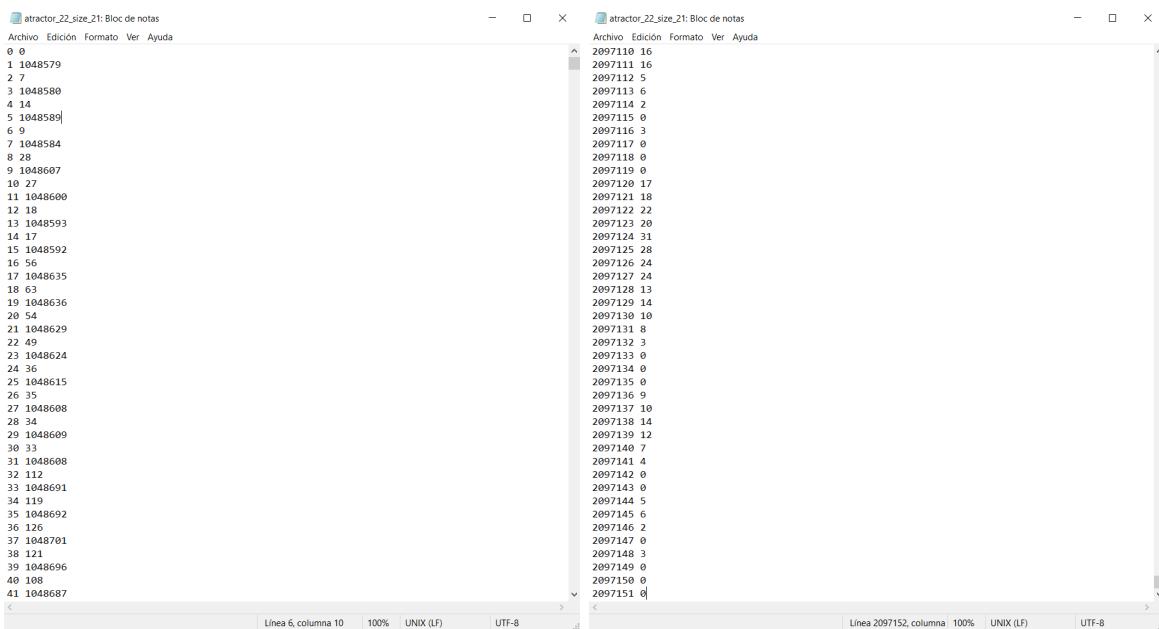


Figura 64: Atractor de tamaño 21.

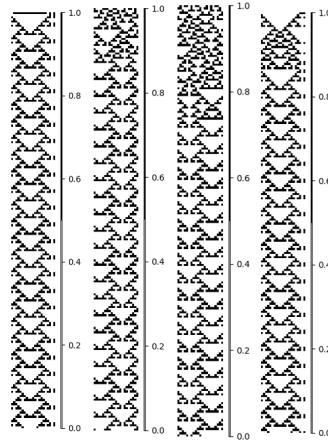


Figura 65: Evoluciones de los atractores.

3.1.21. Tamaño 22

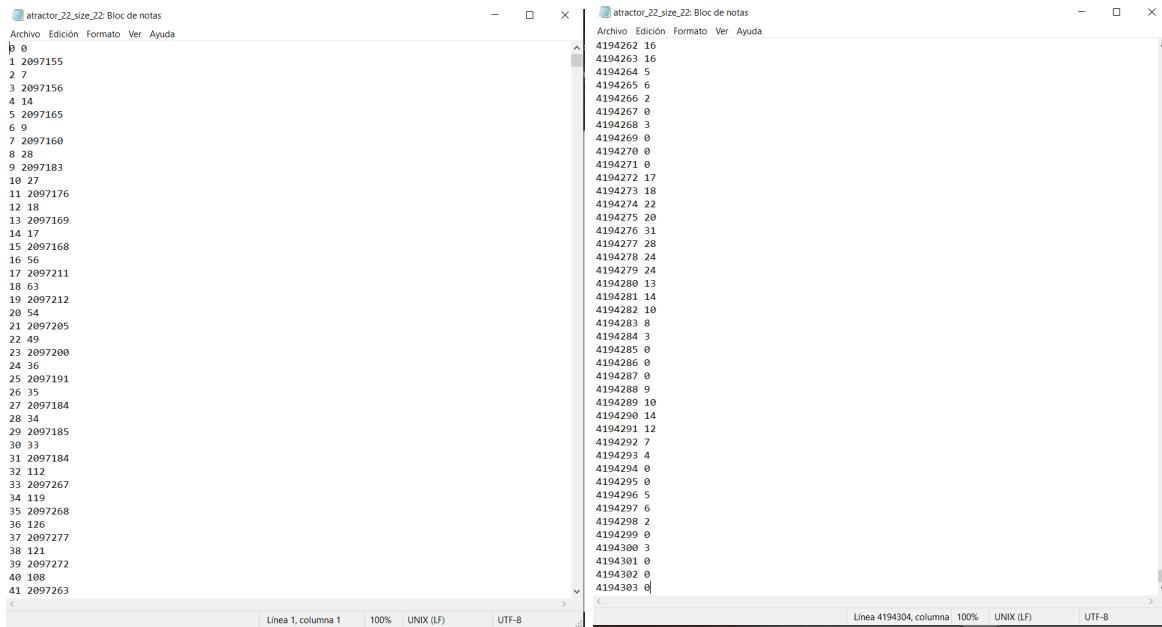


Figura 66: Atractor de tamaño 22.

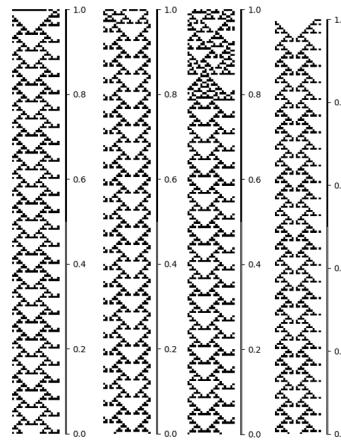


Figura 67: Evoluciones de los atractores.

3.1.22. Tamaño 23

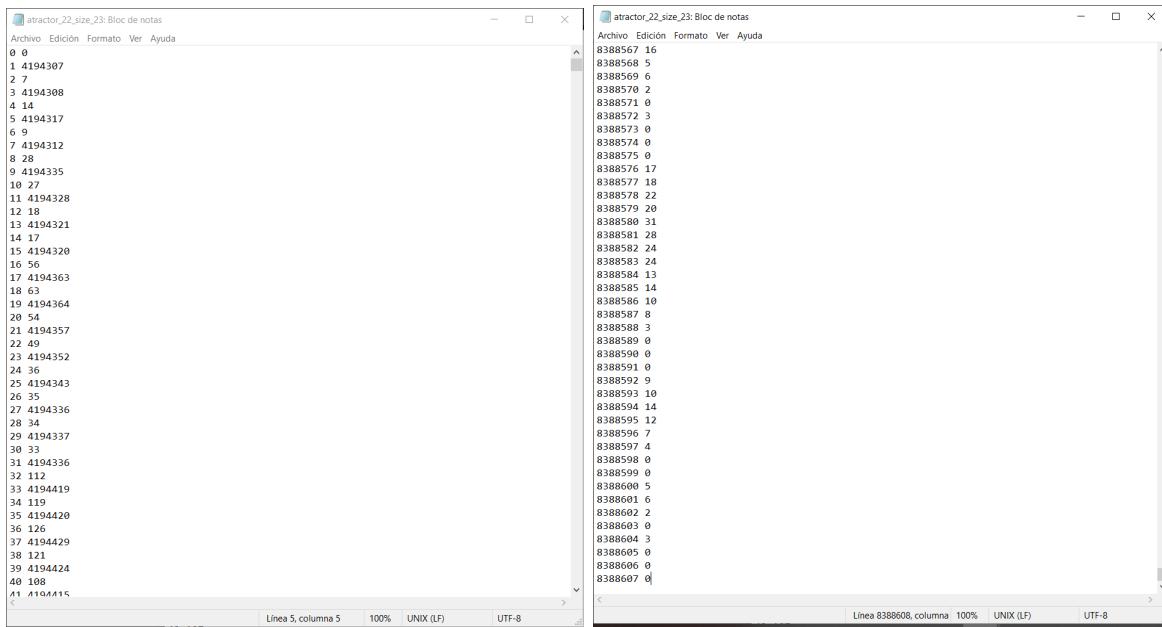


Figura 68: Atractor de tamaño 23.

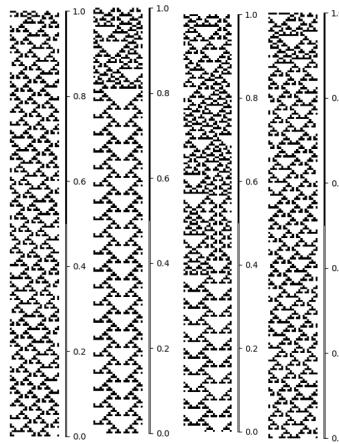


Figura 69: Evoluciones de los atractores.

3.1.23. Tamaño 24

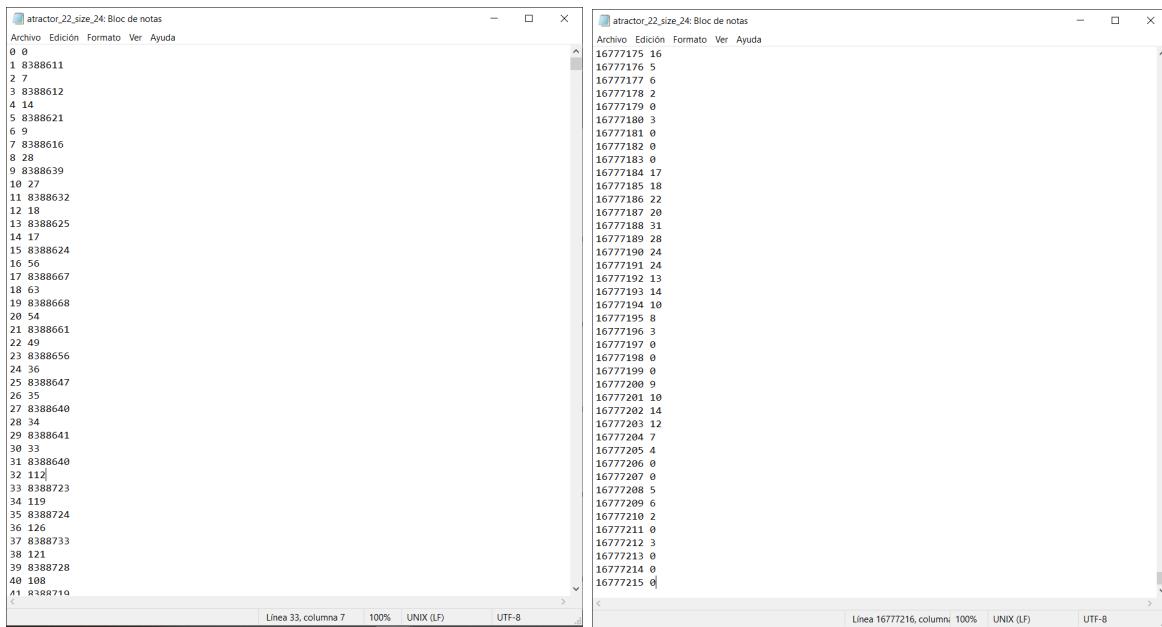


Figura 70: Atractor de tamaño 24.

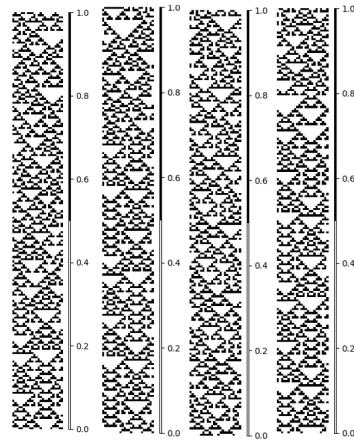


Figura 71: Evoluciones de los atractores.

3.1.24. Tamaño 25

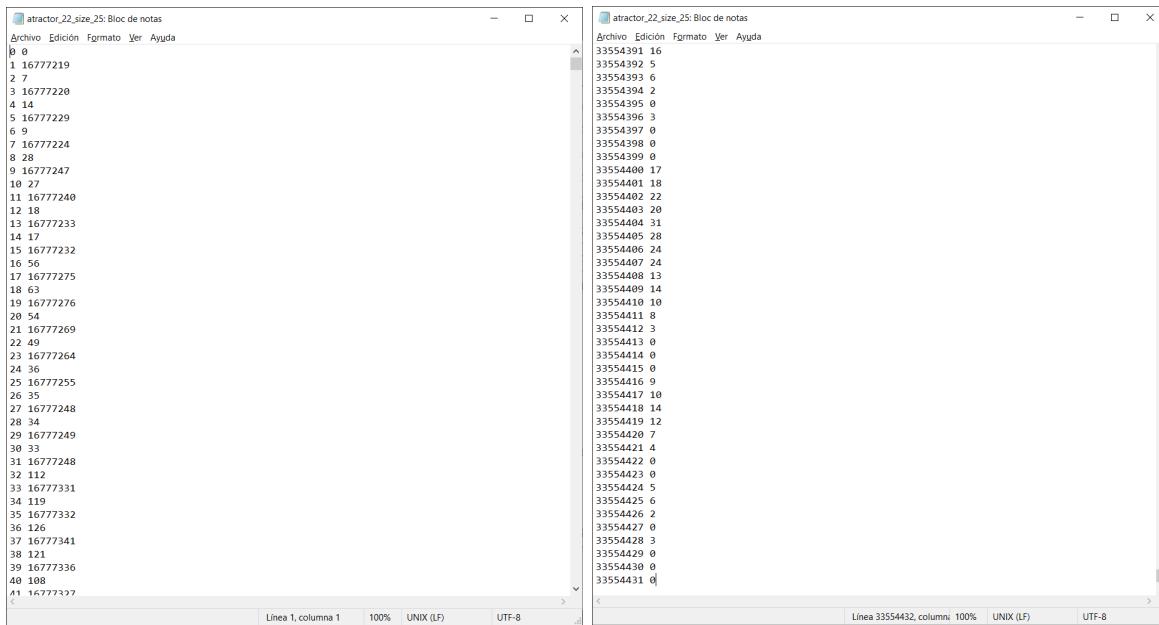


Figura 72: Atractor de tamaño 25.

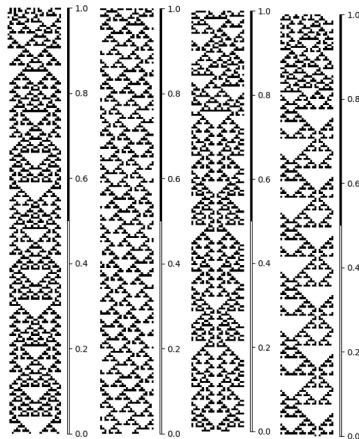


Figura 73: Evoluciones de los atractores.

3.2. Regla 54

El formato sera el siguiente, primero se pondrá la imagen del atractor y después evoluciones de los nodos para ver los atractores que existen.

3.2.1. Tamaño 2

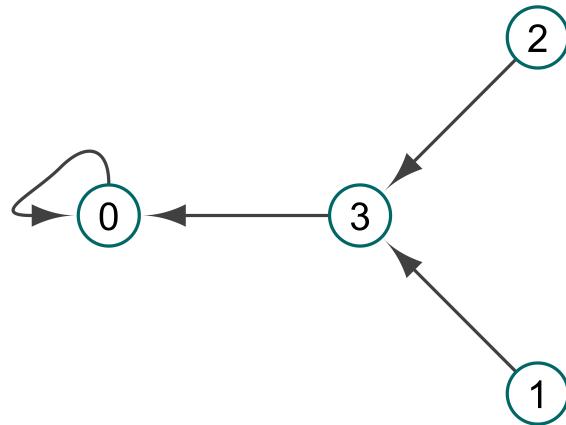


Figura 74: Atractor de tamaño 2.

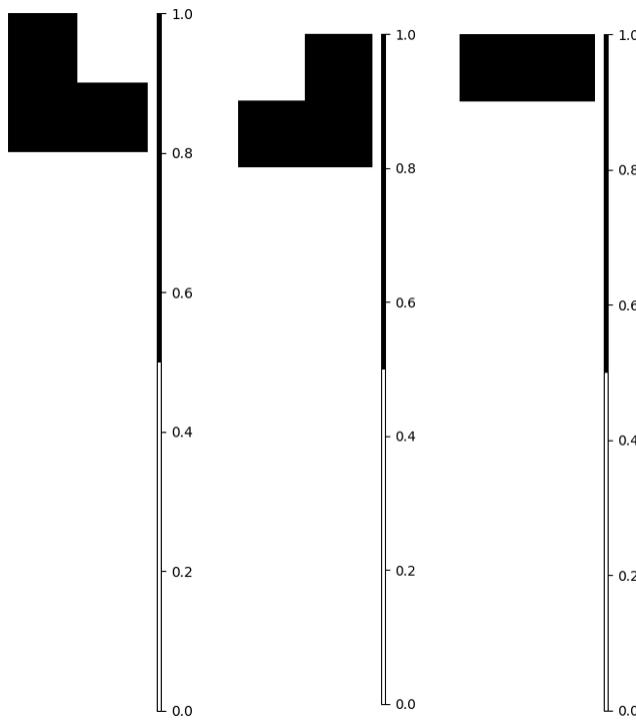


Figura 75: Evoluciones de los atractores.

Mencionar que en la imagen 57 y durante algunos tamaños las evoluciones serán acomodadas tal que del lado izquierdo se tenga el nodo de menor numero al mayor, ya que si lo hicieramos por cada nodo simplemente no acabaríamos, mencionar que para este atractor no vemos el caso del nodo 0 ya que el nodo 3 al dirigirse al nodo 0 entra en un ciclo en donde vemos el comportamiento del nodo 0.

3.2.2. Tamaño 3

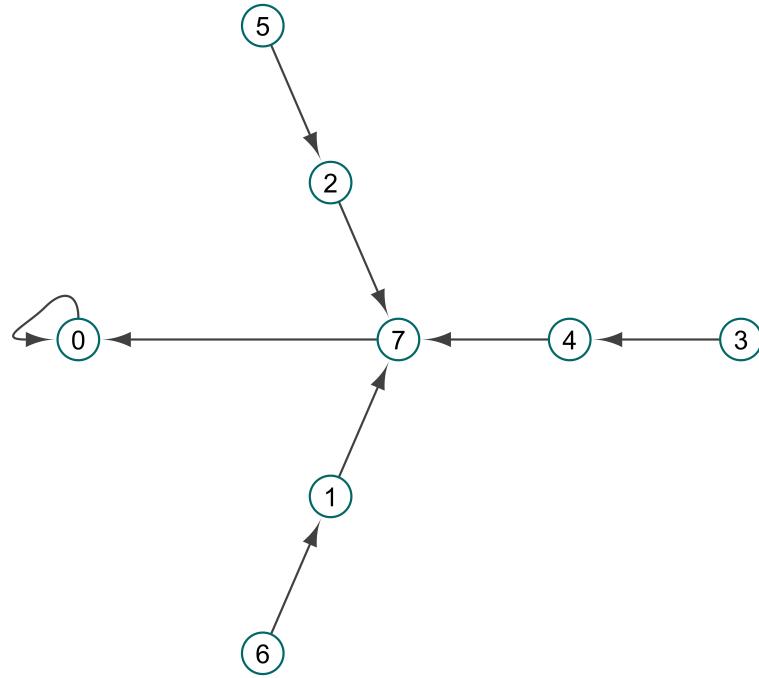


Figura 76: Atractor de tamaño 3.

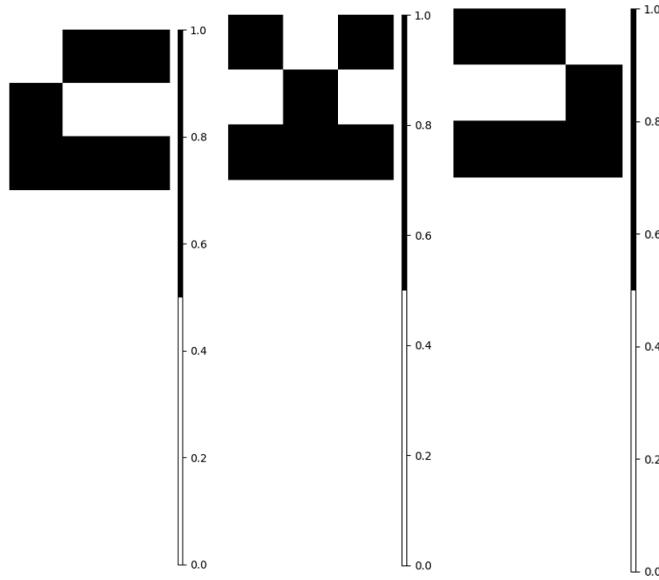


Figura 77: Evoluciones de los atractores.

Mencionar que para este atractor no vemos el caso del nodo 0, 1, 2, 4 y ya que los nodos 5, 6, 3 pasan por los nodos anteriormente mencionados y ver sus evoluciones es innecesario ya que las podemos apreciar por medio de las evoluciones de los nodos seleccionados.

3.2.3. Tamaño 4

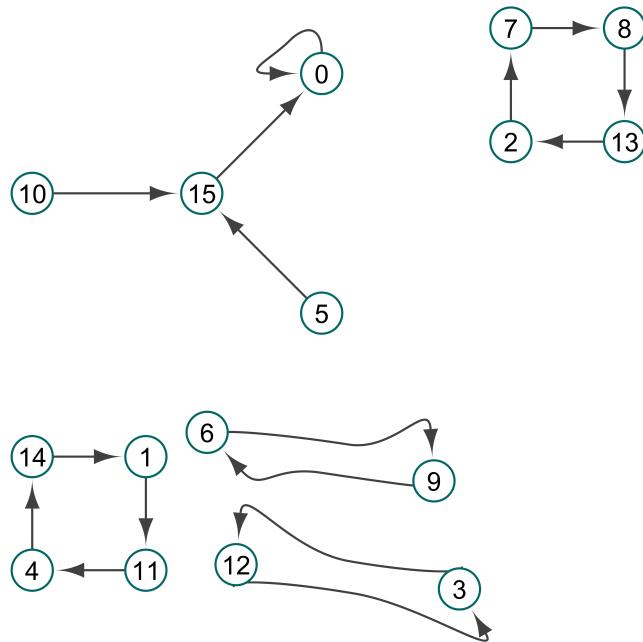


Figura 78: Atractor de tamaño 4.

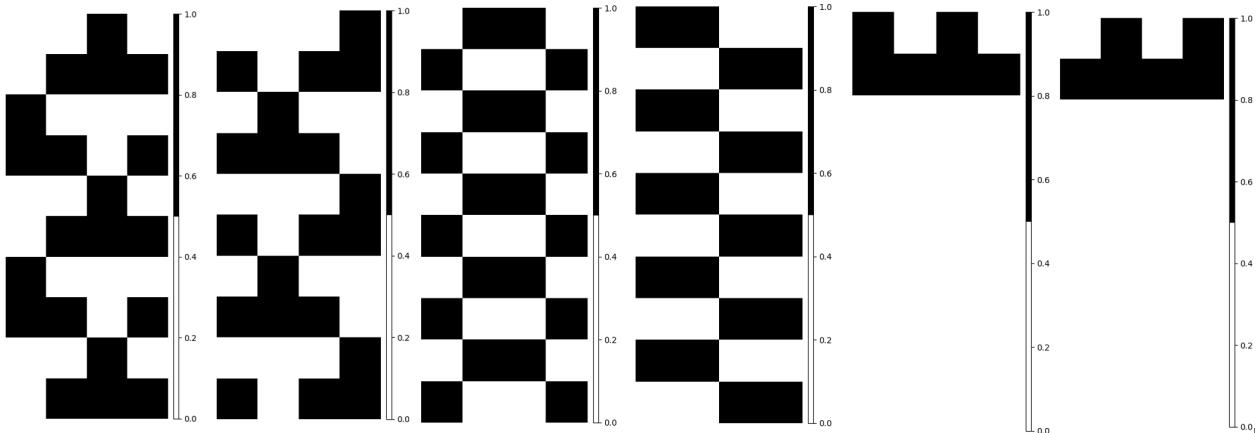


Figura 79: Evoluciones de los atractores.

Mencionar que para este atractor no vemos el caso de los nodos 0, 15, 14, 4, 11, 7, 8, 13, 3 y 9 ya se pueden ver sus evoluciones por medio de otros nodos, por ejemplo, para ver la evolución del nodo 3 hacemos uso del nodo 12 ya que es un ciclo entre ambos, en el caso del nodo 9 y 6 ocurre lo mismo, mientras que los nodos 14, 4 y 11 forman un ciclo junto con el nodo 1, los nodos 7, 8 y 13 junto con el nodo 2 forman otro ciclo y el nodo 0 es accesible por el nodo 15 el cual a su vez es accesible por el nodo 10 y 5.

3.2.4. Tamaño 5

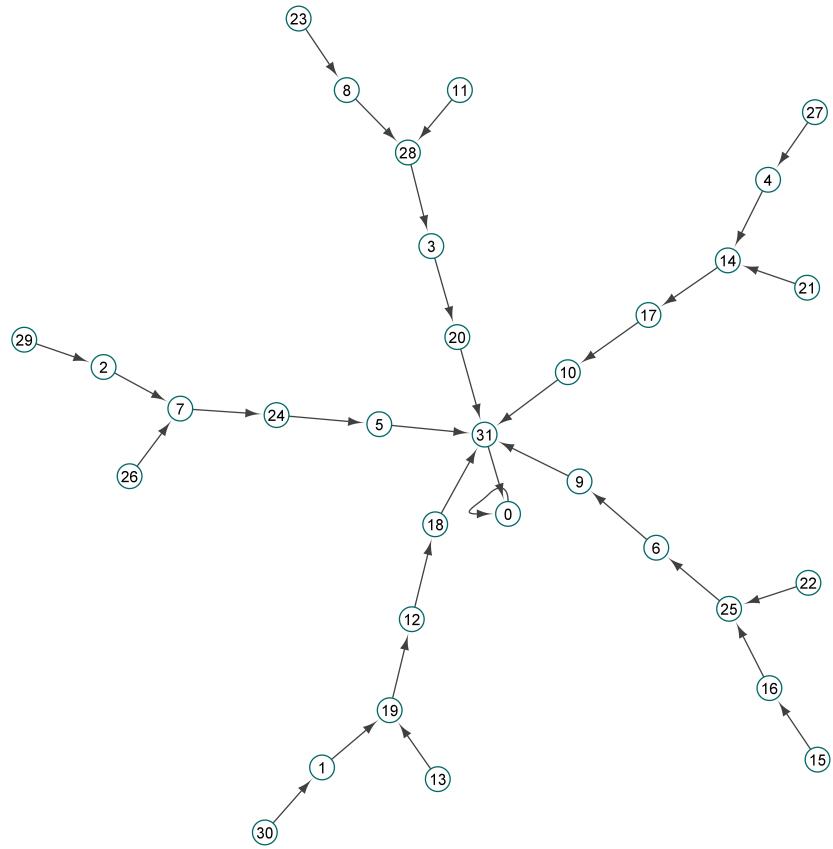


Figura 80: Atractor de tamaño 5.

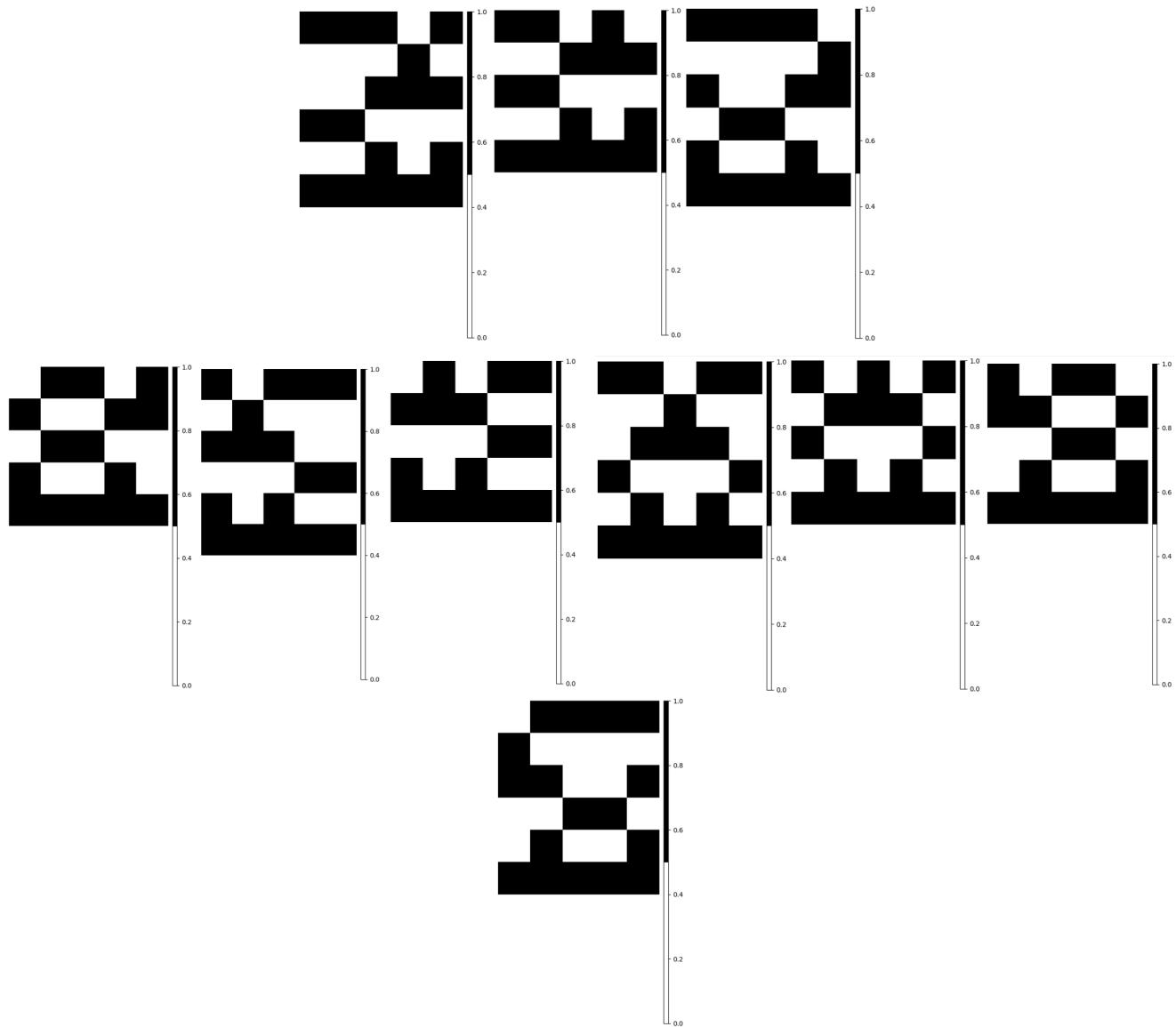


Figura 81: Evoluciones de los atractores.

Mencionar que para este atractor solamente vemos la evolución de los nodos 13, 30, 26, 29, 23, 11, 27, 21, 22 y 154 ya que son los nodos del exterior y estos pasan por los demás nodos y podemos ver como funciona el atractor el cual es el nodo 0.

3.2.5. Tamaño 6

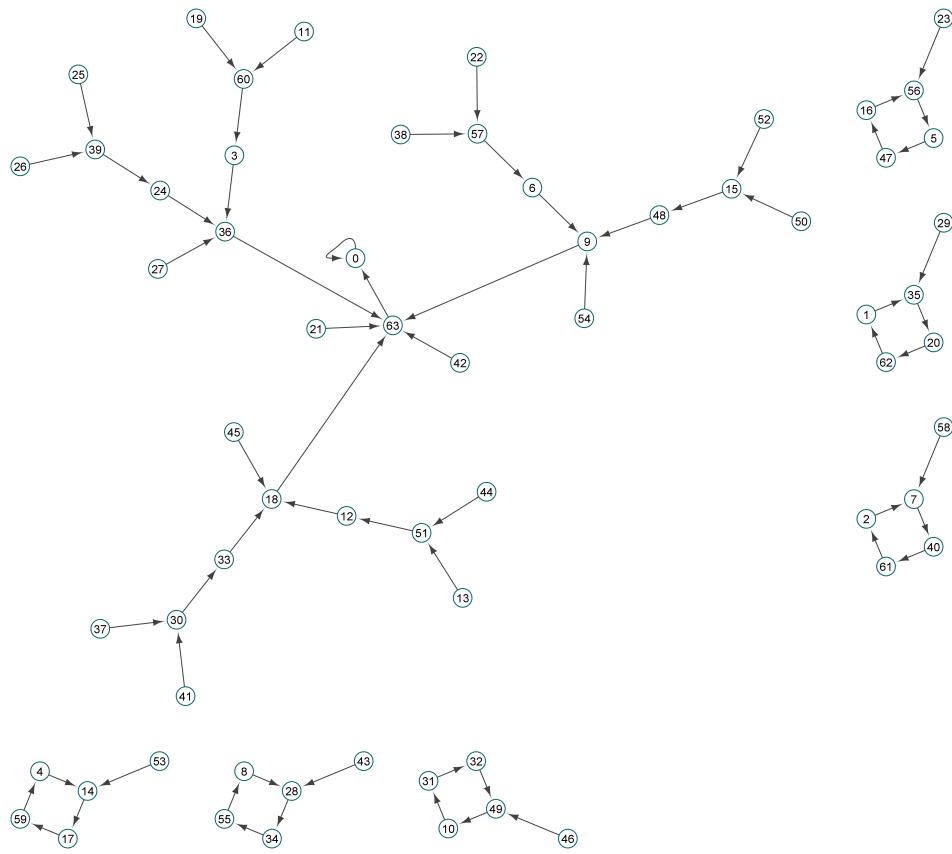


Figura 82: Atractor de tamaño 6.

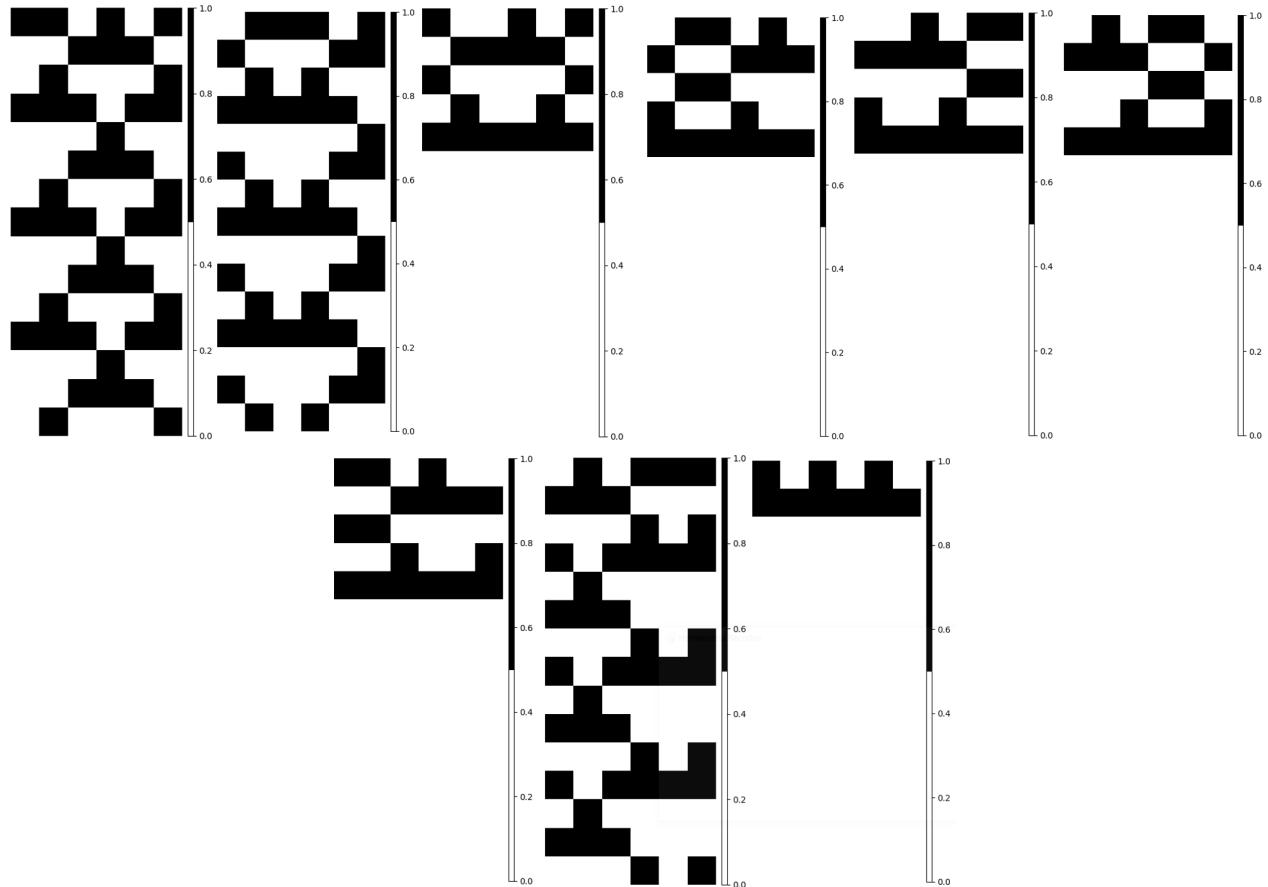


Figura 83: Evoluciones de los atractores.

Mencionar que a partir de este atractor solamente tomaremos unos cuantos nodos de prueba ya que abarcar la mayoría tomaría mucho espacio del reporte.

3.2.6. Tamaño 7

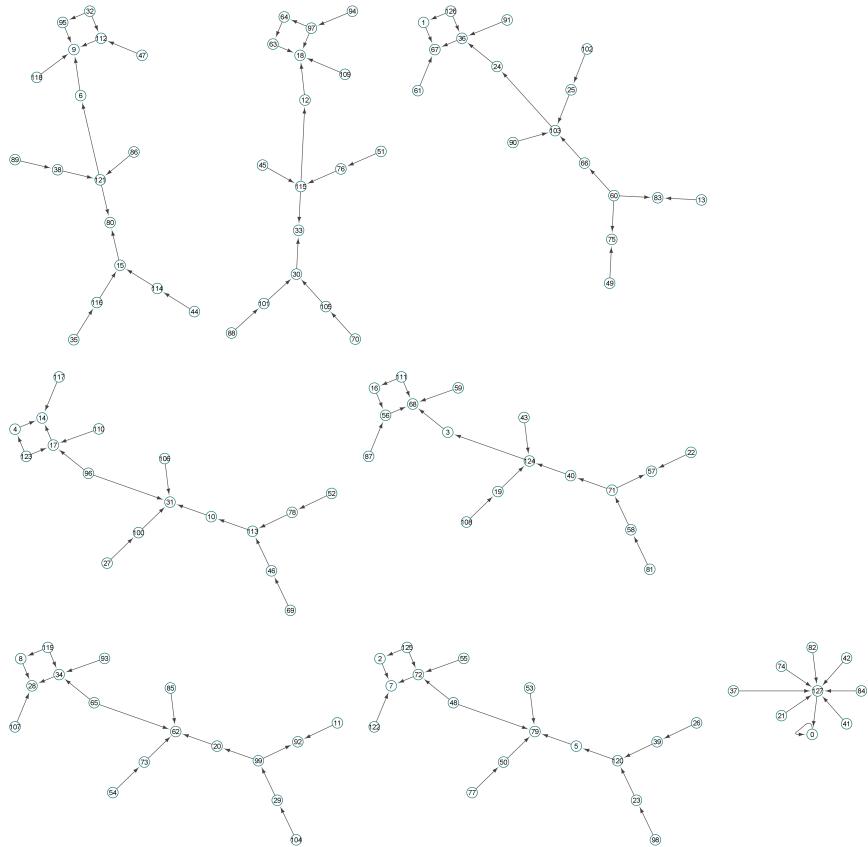


Figura 84: Atractor de tamaño 7.

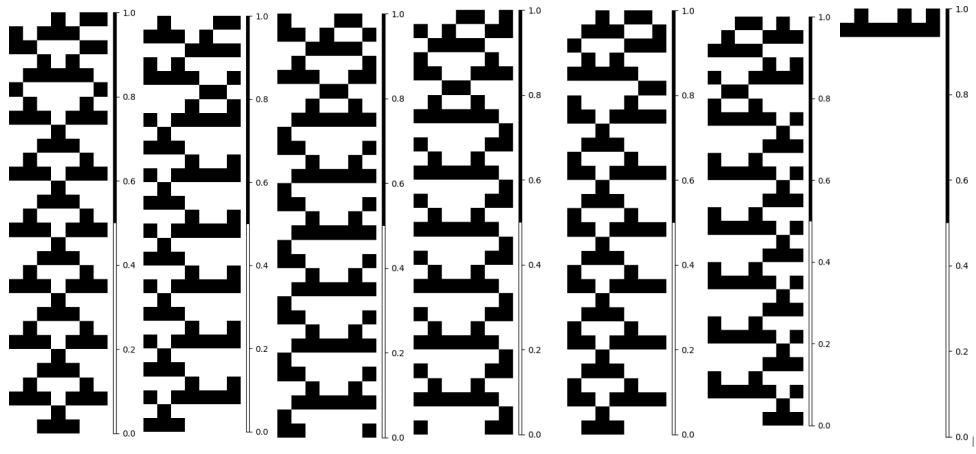


Figura 85: Evoluciones de los atractores.

Los nodos de prueba fueron nodos de cada uno de los atractores que podemos ver, se utilizo el mas lejano de cada atractor, esto con el fin de poder visualizar como se entra en el atractor con la simulación en nuestro primer programa.

3.2.7. Tamaño 8

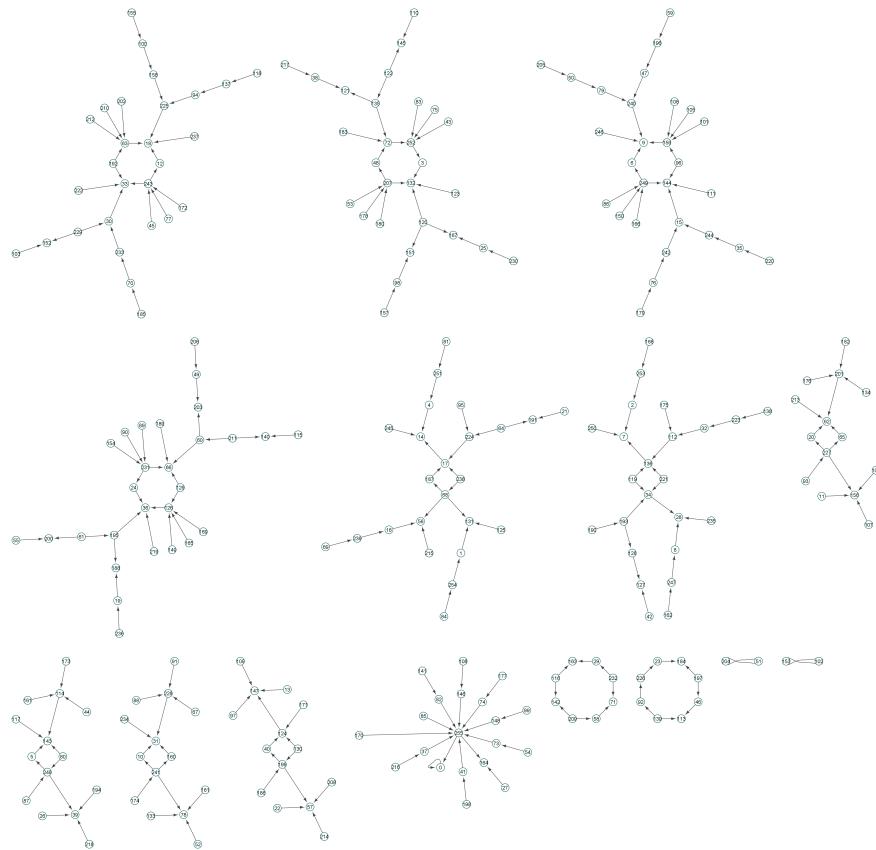


Figura 86: Atractor de tamaño 8.

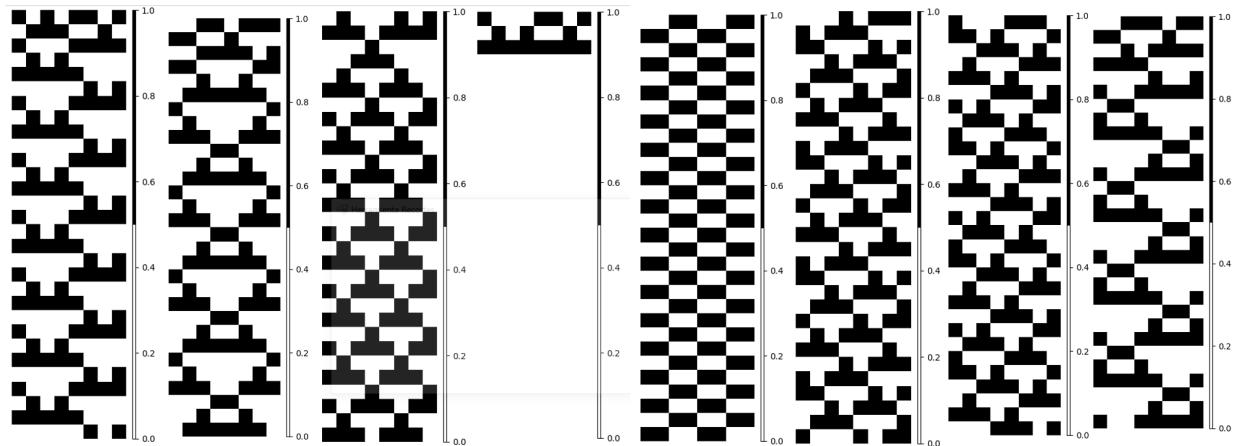


Figura 87: Evoluciones de algunos atractores.

3.2.8. Tamaño 9

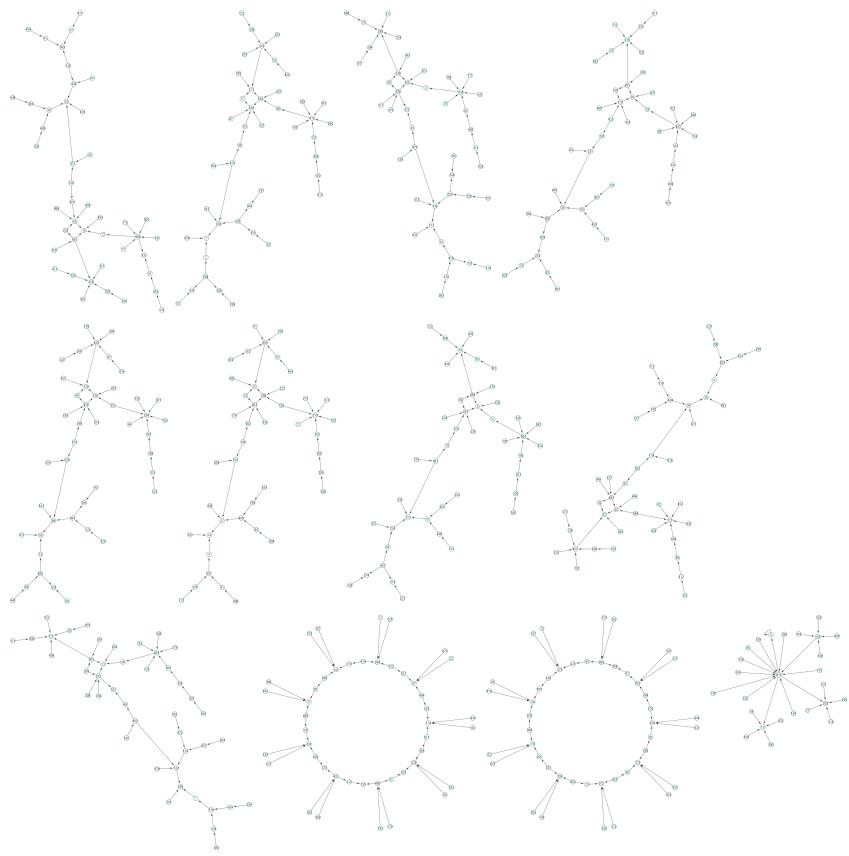


Figura 88: Atractor de tamaño 9.

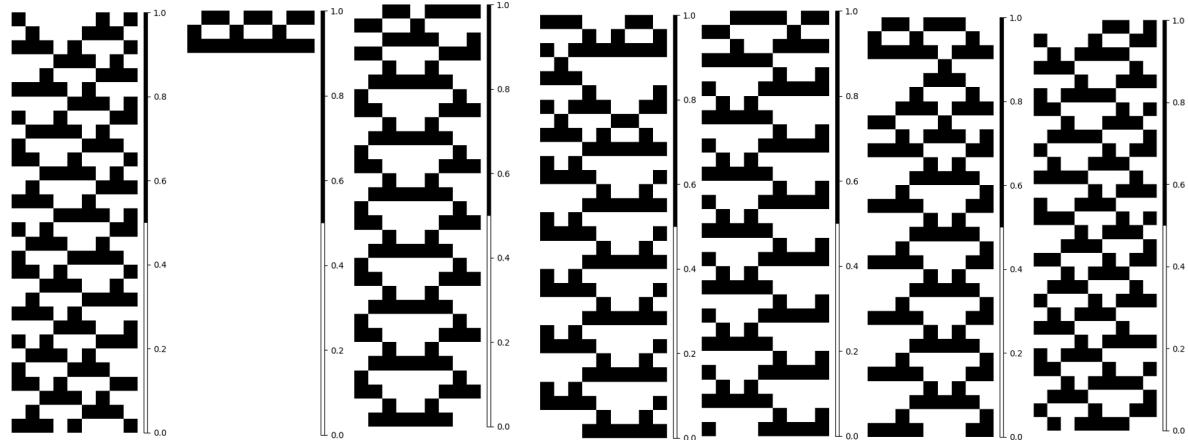


Figura 89: Evoluciones de los atractores.

3.2.9. Tamaño 10

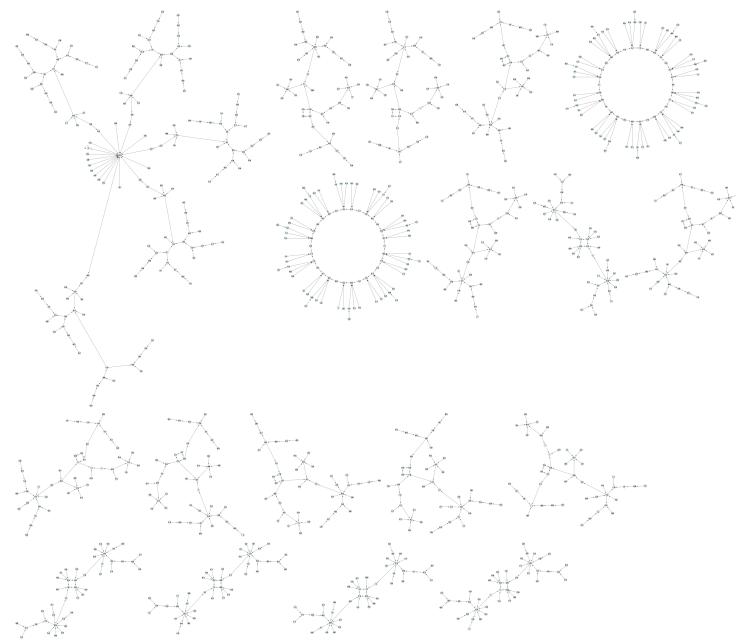


Figura 90: Atractor de tamaño 10.

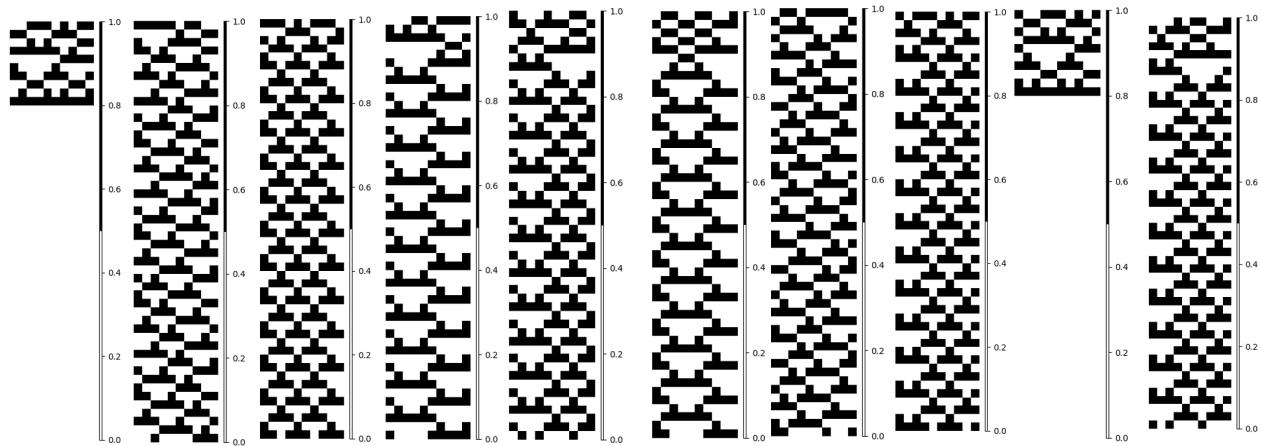


Figura 91: Evoluciones de los atractores.

3.2.10. Tamaño 11

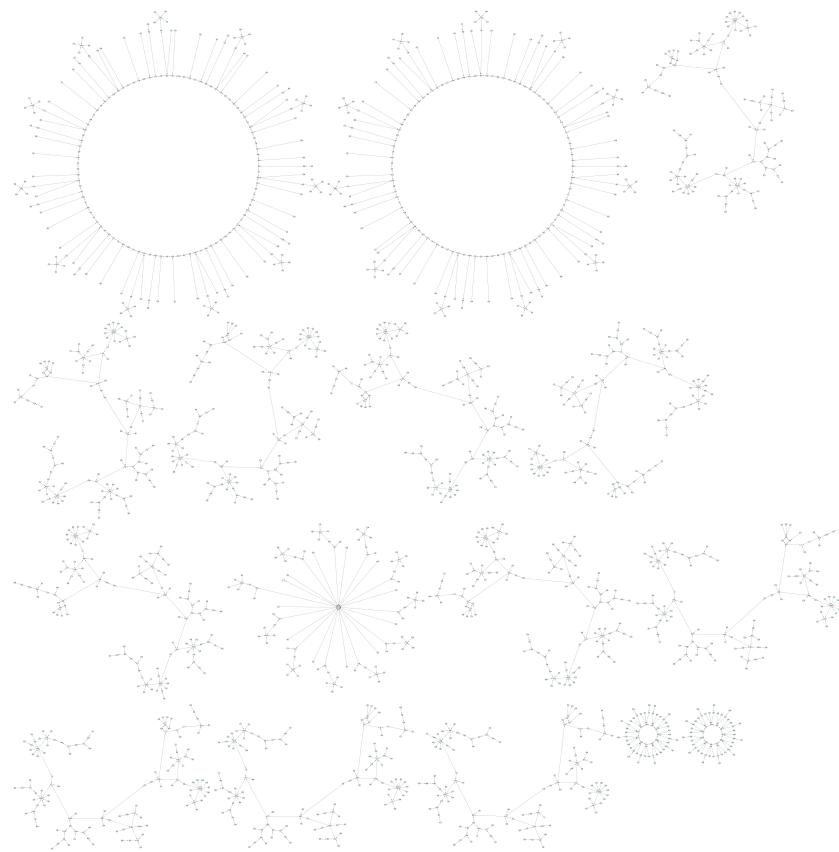


Figura 92: Atractor de tamaño 11.

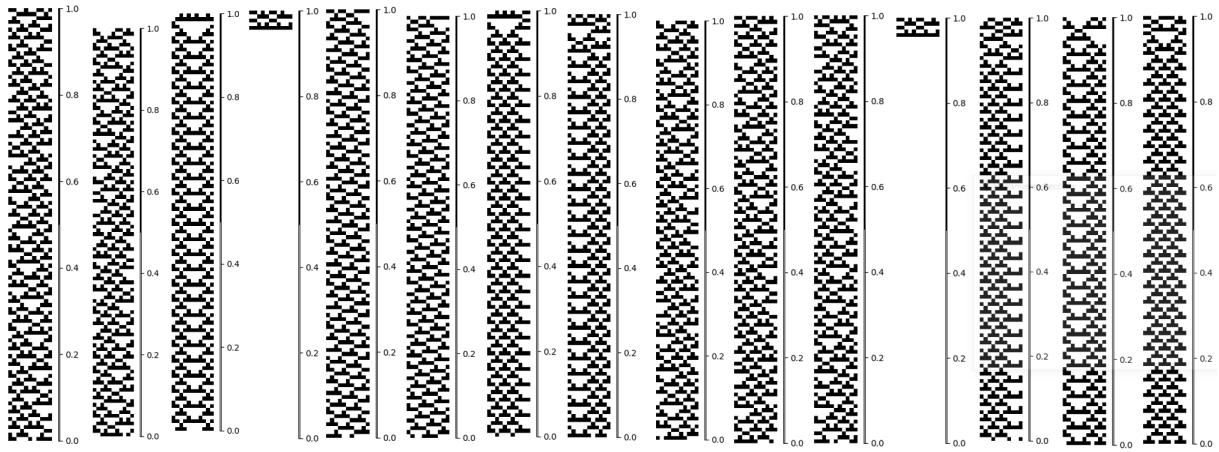


Figura 93: Evoluciones de los atractores.

3.2.11. Tamaño 12

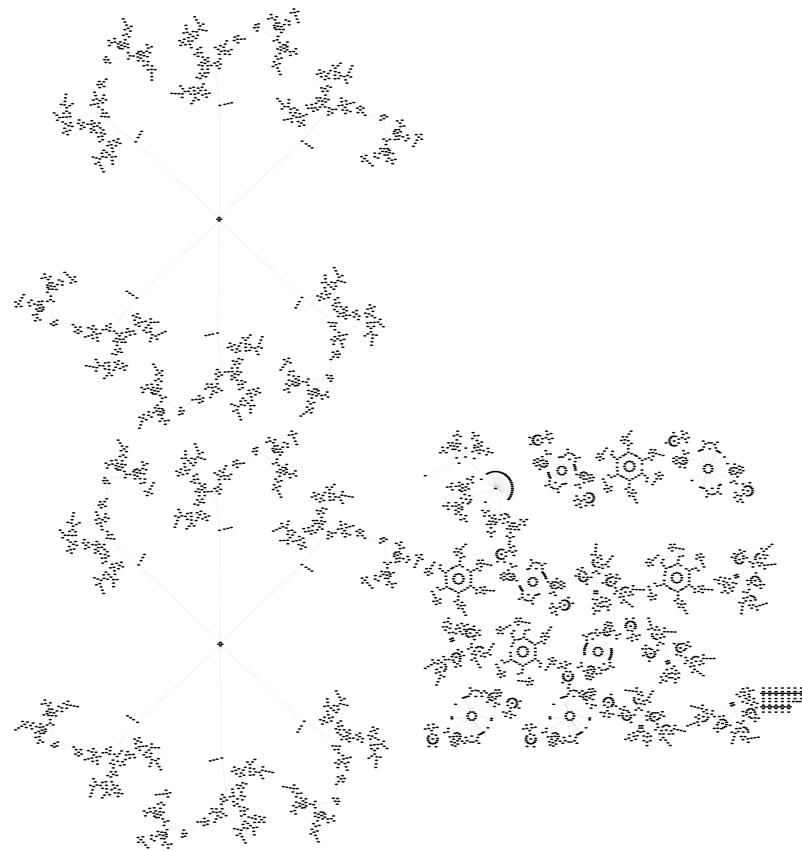


Figura 94: Atractor de tamaño 12.

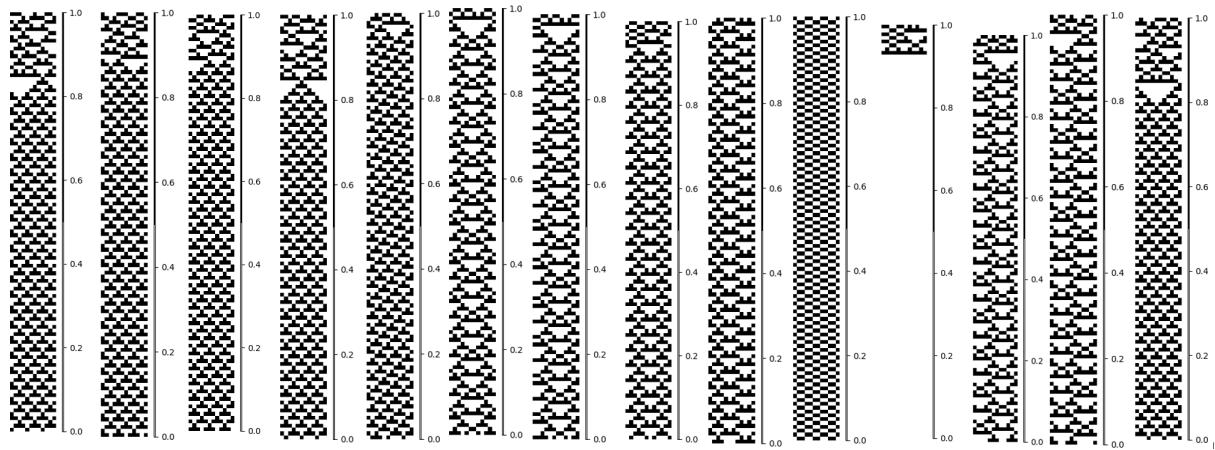


Figura 95: Evoluciones de los atractores.

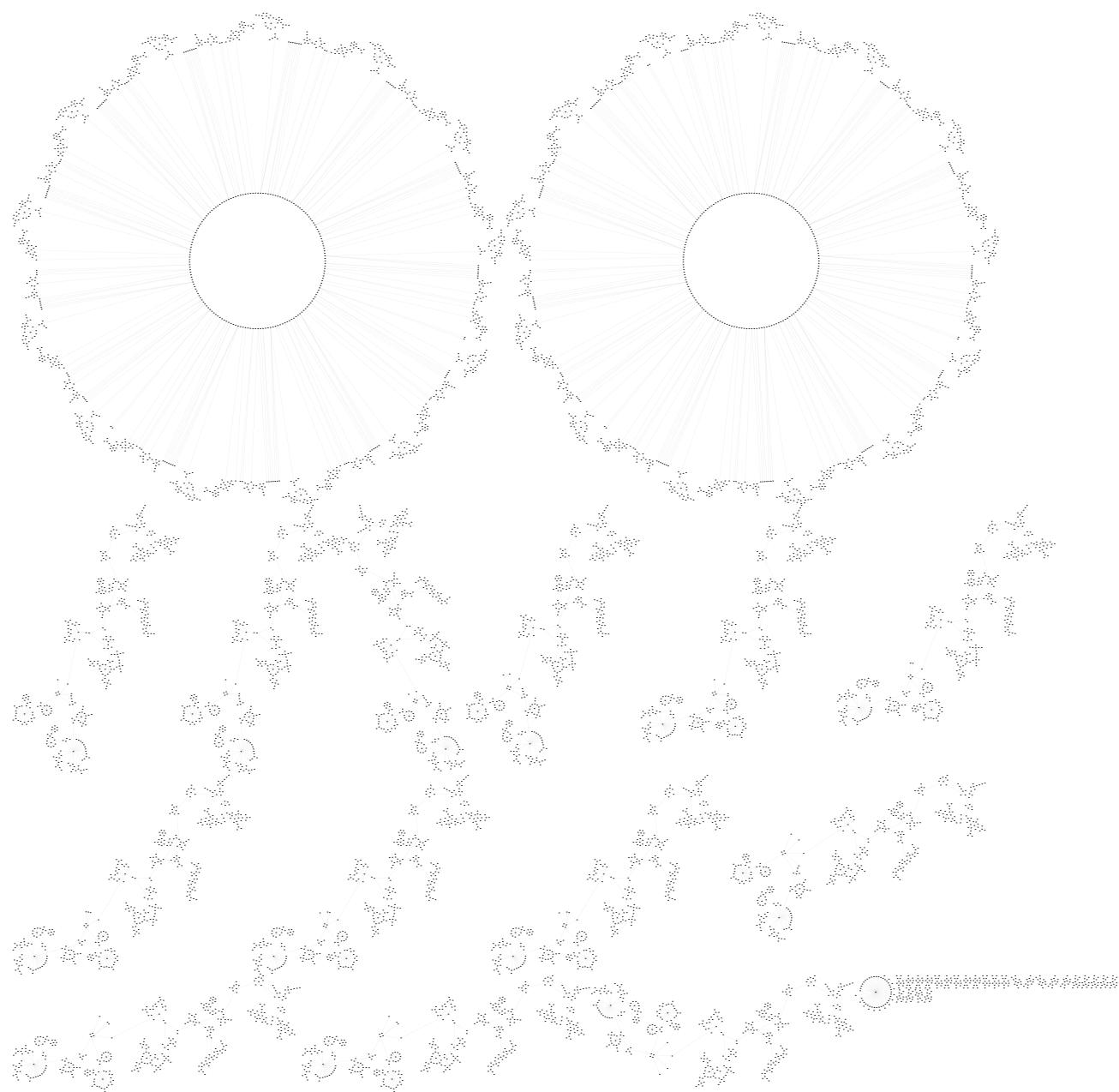
3.2.12. Tamaño 13

Figura 96: Atractor de tamaño 13.

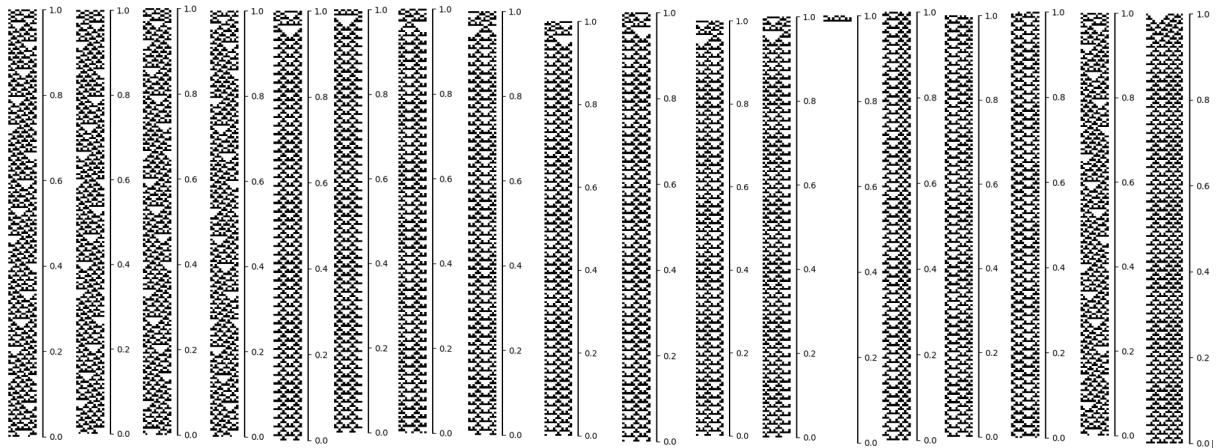


Figura 97: Evoluciones de los atractores.

3.2.13. Tamaño 14

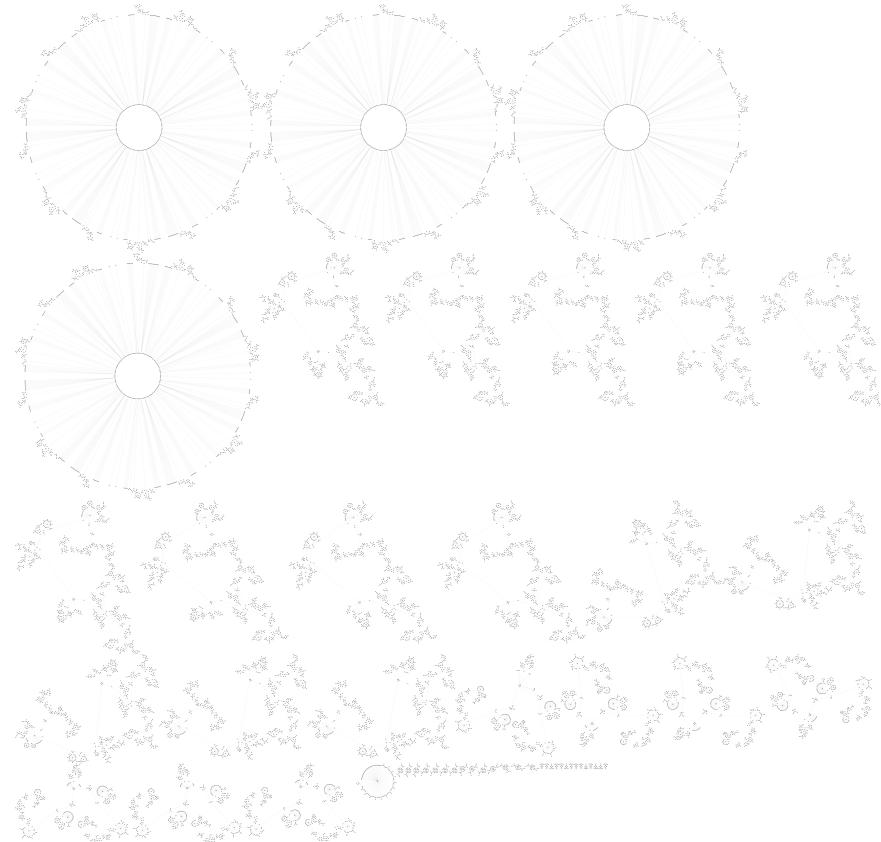


Figura 98: Atractor de tamaño 14.

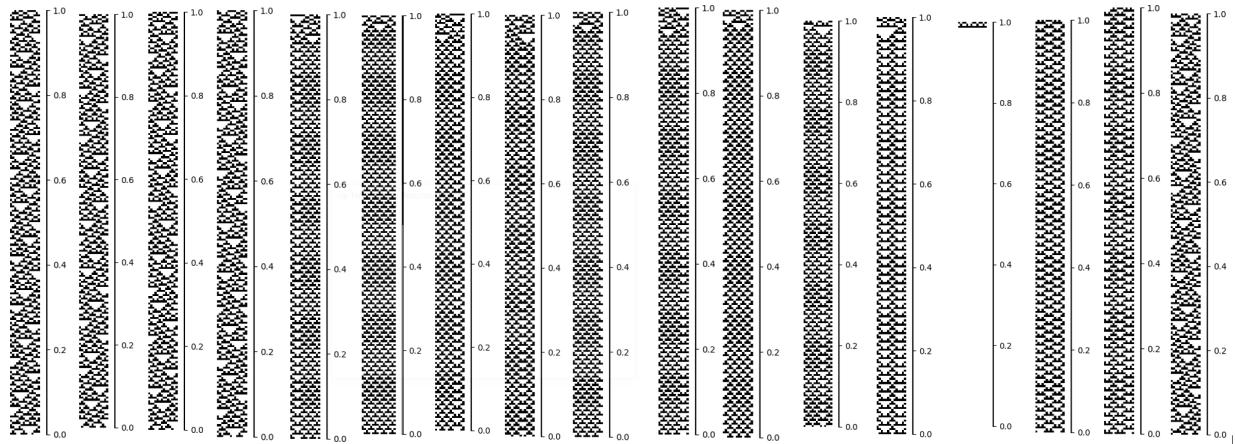


Figura 99: Evoluciones de los atractores.

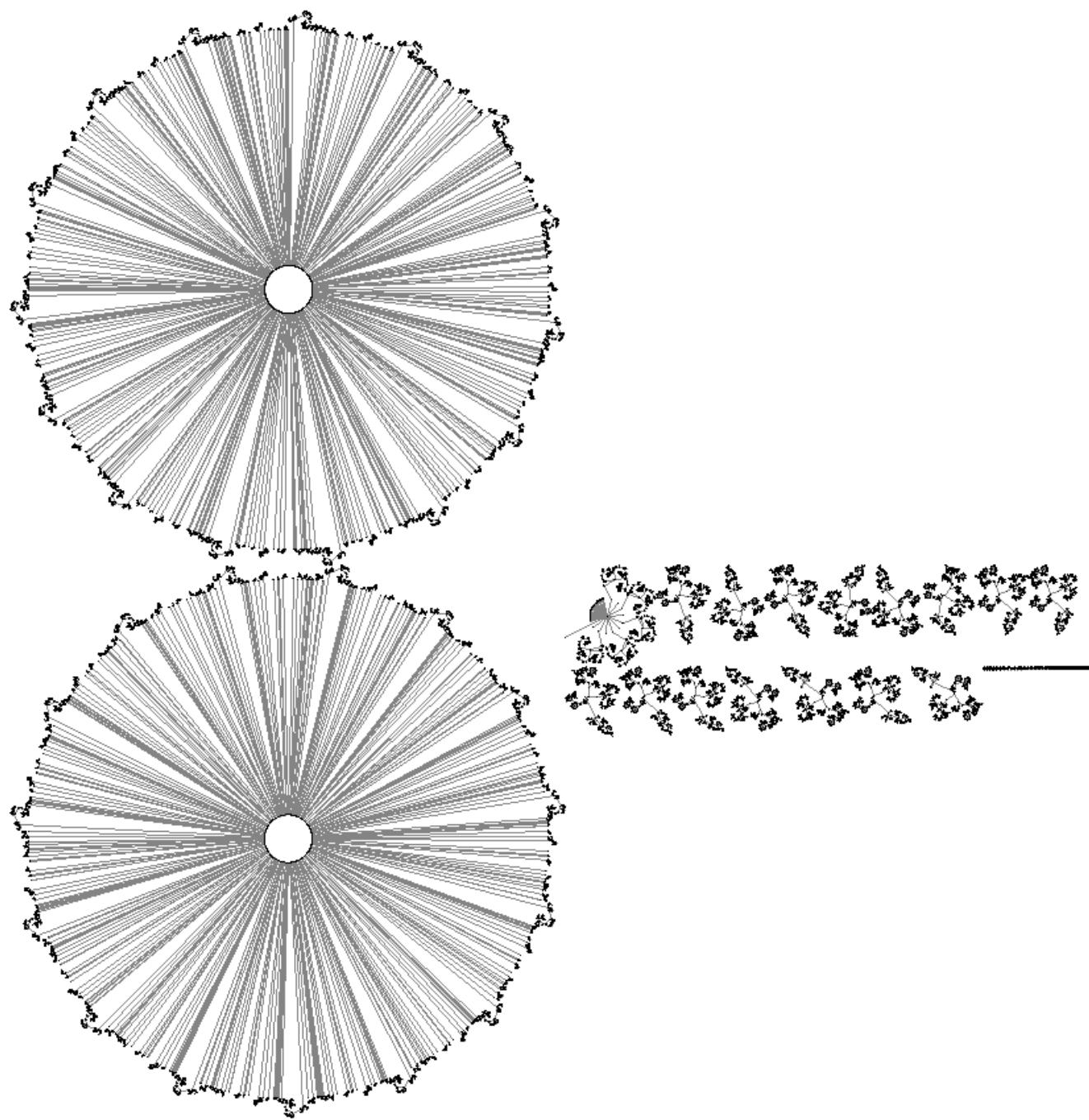
3.2.14. Tamaño 15

Figura 100: Atractor de tamaño 15.

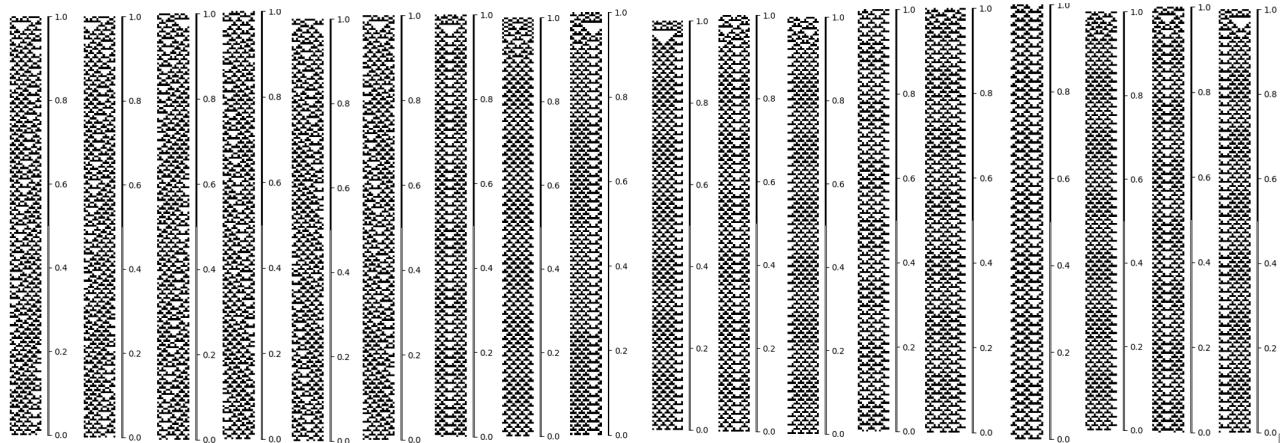


Figura 101: Evoluciones de los atractores.

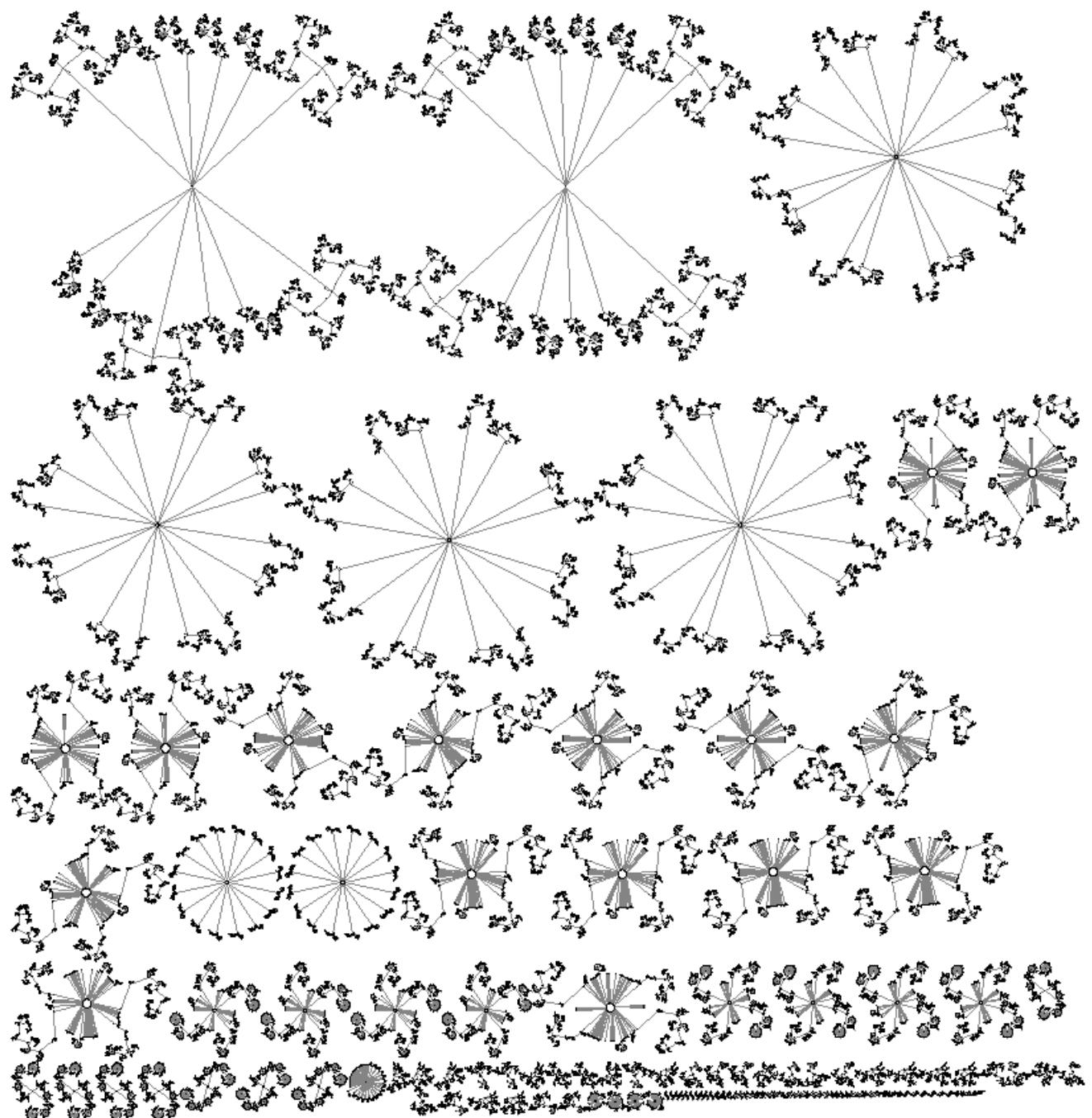
3.2.15. Tamaño 16

Figura 102: Atractor de tamaño 16.

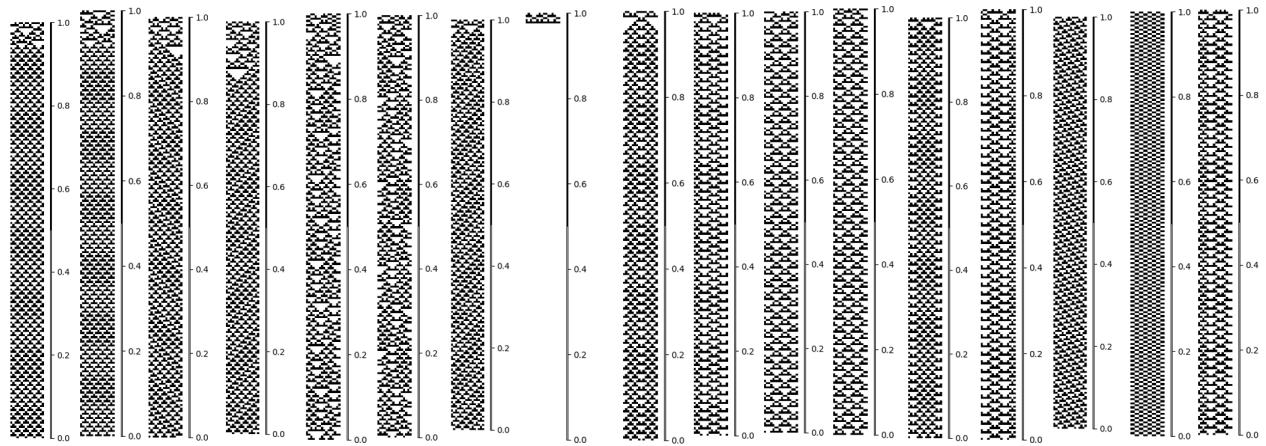


Figura 103: Evoluciones de los atractores.

3.2.16. Tamaño 17

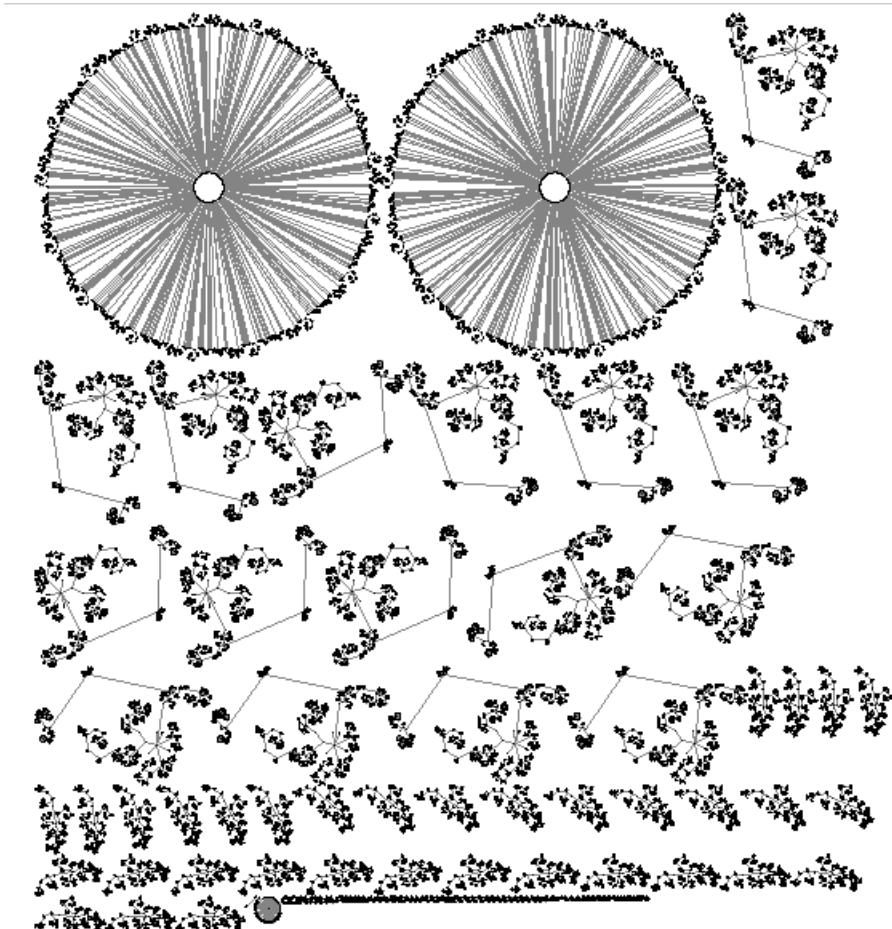


Figura 104: Atractor de tamaño 17.

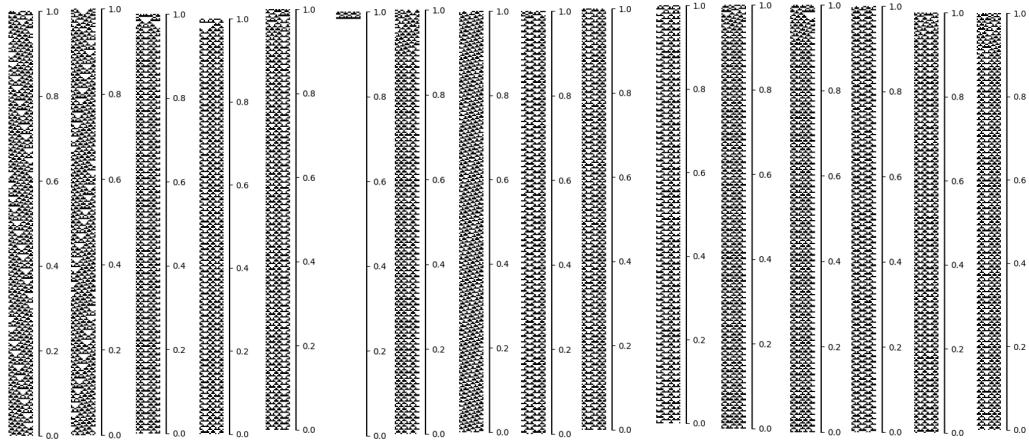


Figura 105: Evoluciones de los atractores.

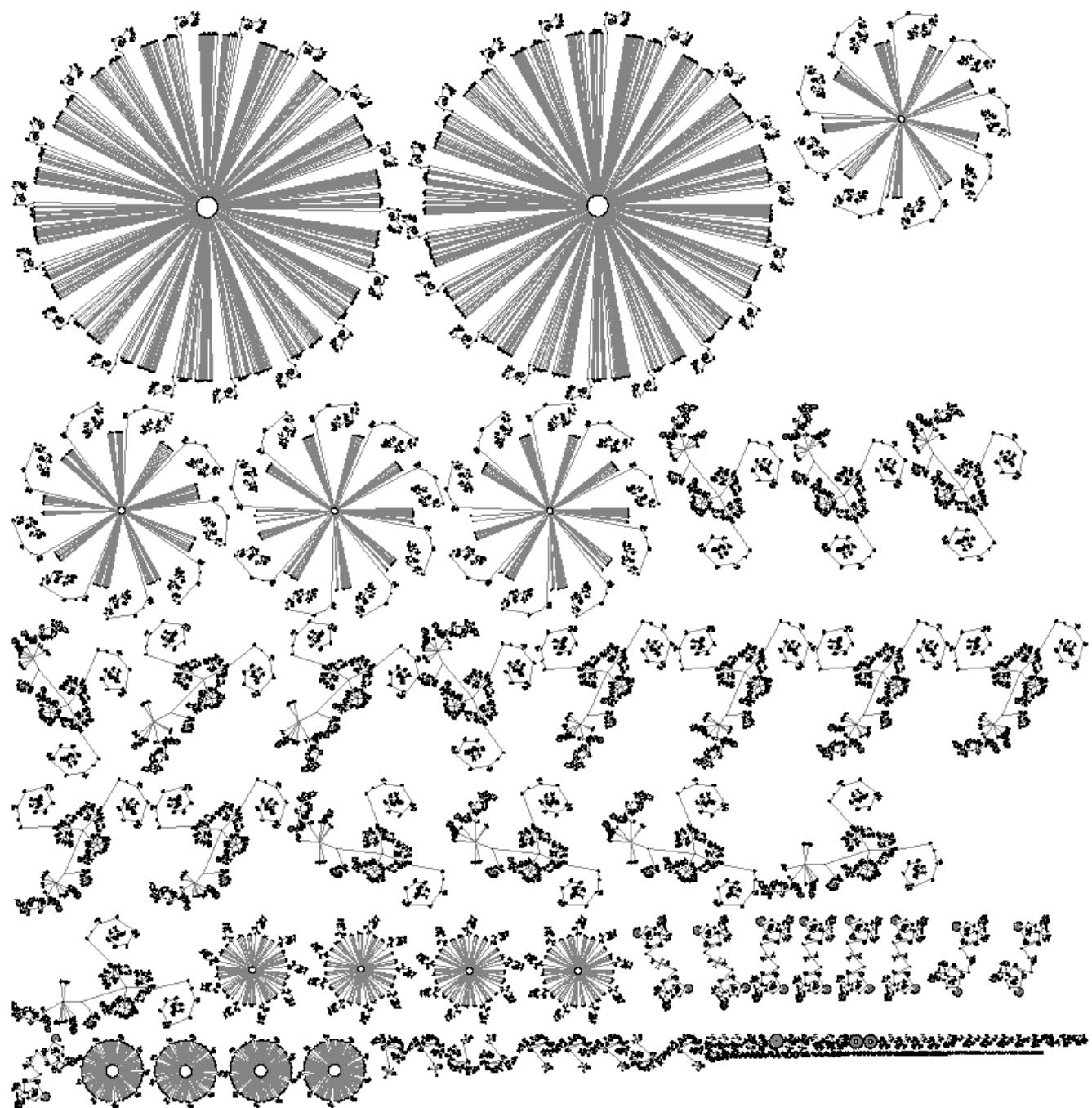
3.2.17. Tamaño 18

Figura 106: Atractor de tamaño 18.

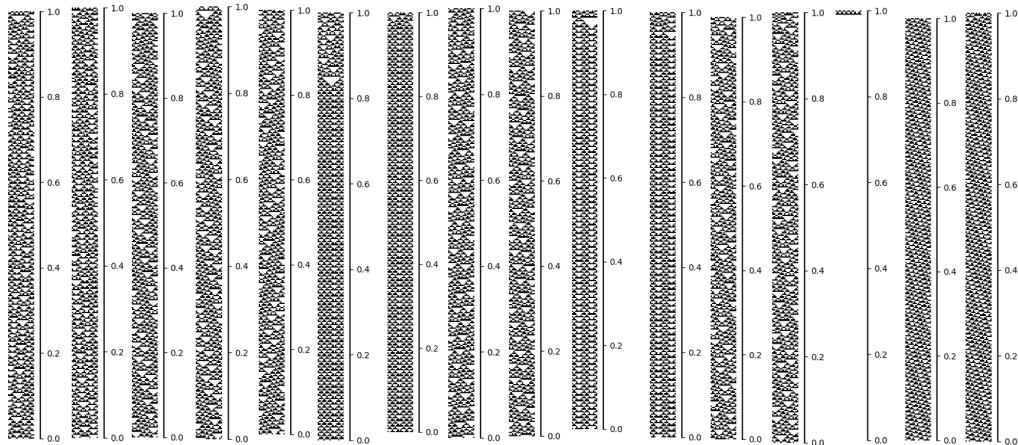


Figura 107: Evoluciones de los atractores.

Este fue el ultimo atractor que fue capaz de construir nuestro programa, sin embargo, mediante el uso de JetBrains Datalore, el cual nos sirve como una maquina externa que nos da mas poder de computo pudimos calcular hasta el tamaño 25, esto solo es guardado como archivos .txt que contienen los nodos y sus atractor correspondiente.

3.2.18. Tamaño 19

Para estos tamaños en donde solamente calculamos los atractores pero no calculamos la gráfica debido a que el incremento de nodos y por lo tanto del archivo es demasiado grande, solo mencionar que el archivo mas grande obtenido pesa medio Gb. Solo dos capturas de pantalla, el principio y el fin y algunas evoluciones con valores tomados aleatoriamente.

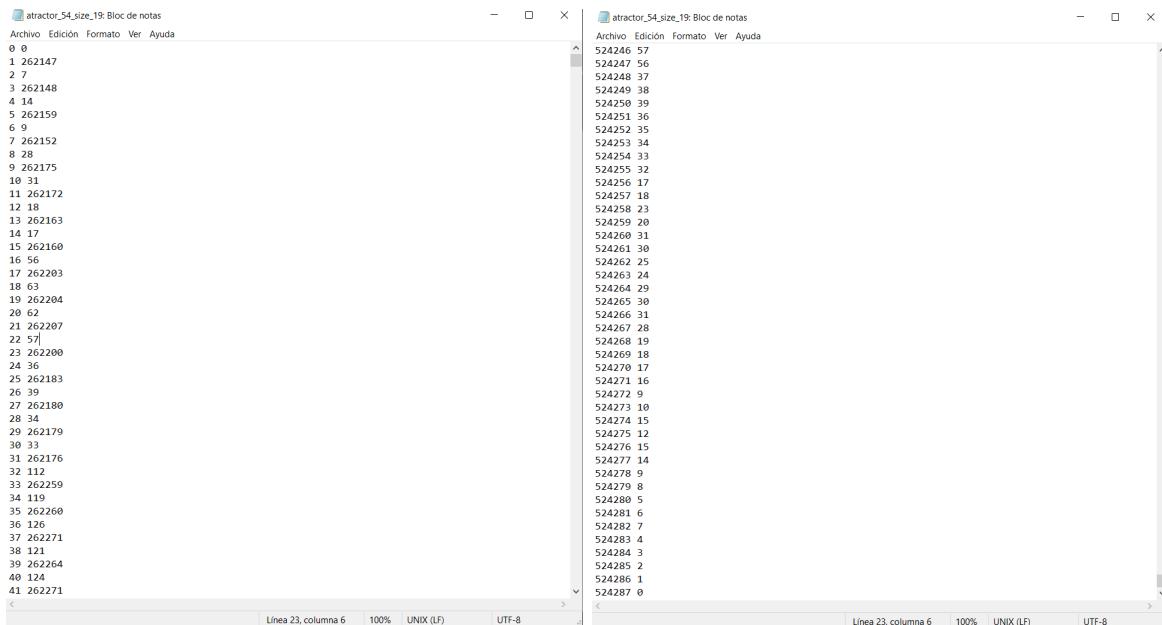


Figura 108: Atractor de tamaño 19.

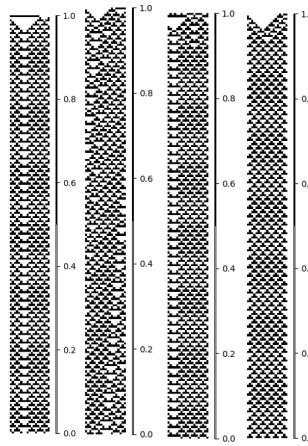


Figura 109: Evoluciones de los atractores.

3.2.19. Tamaño 20

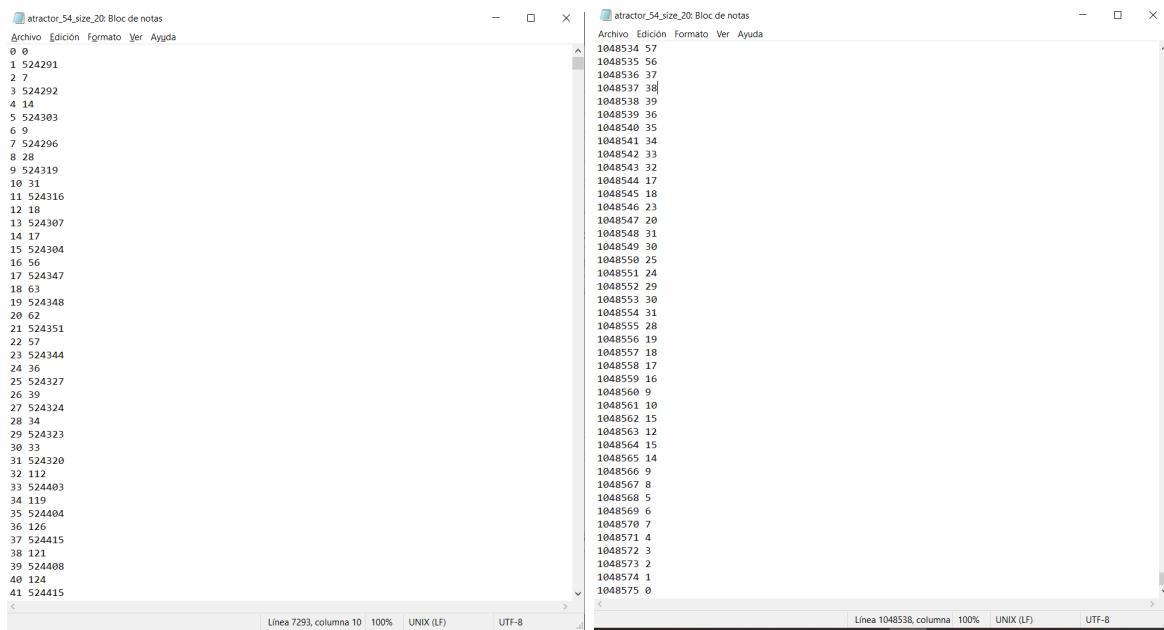


Figura 110: Atractor de tamaño 20.

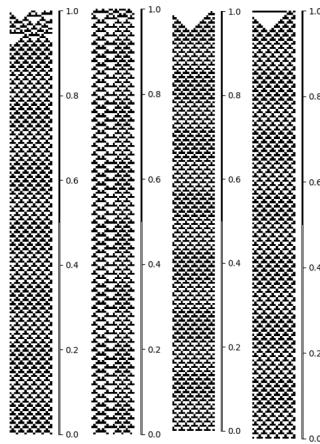


Figura 111: Evoluciones de los atractores.

3.2.20. Tamaño 21

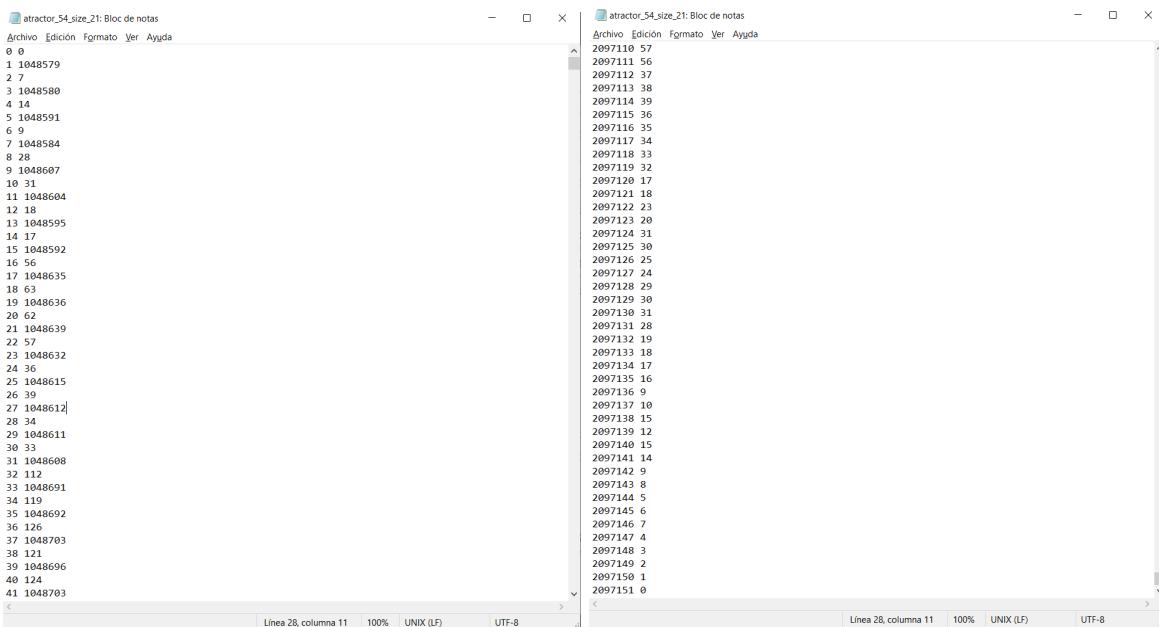


Figura 112: Atractor de tamaño 21.

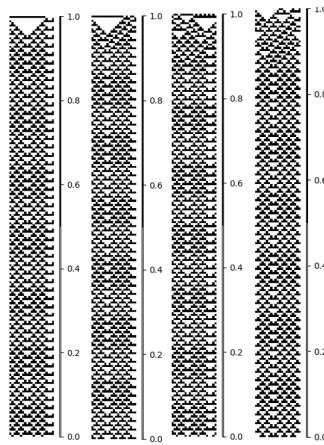


Figura 113: Evoluciones de los atractores.

3.2.21. Tamaño 22

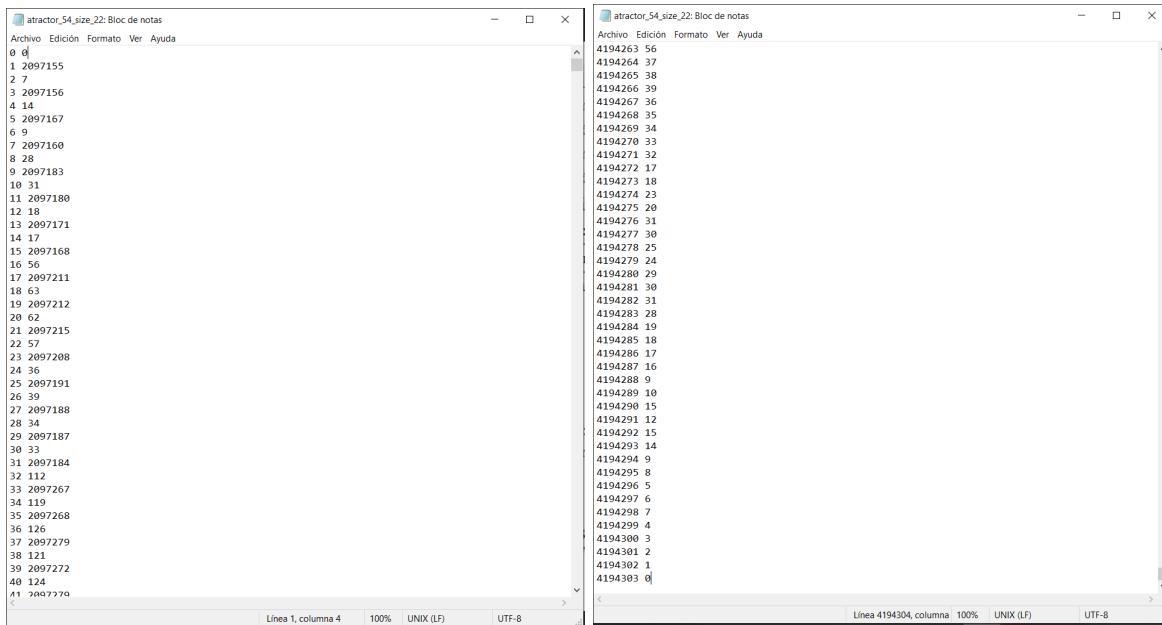


Figura 114: Atractor de tamaño 22.

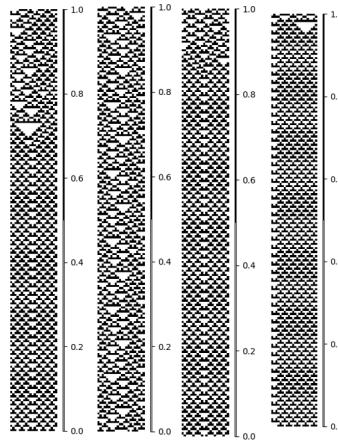


Figura 115: Evoluciones de los atractores.

3.2.22. Tamaño 23

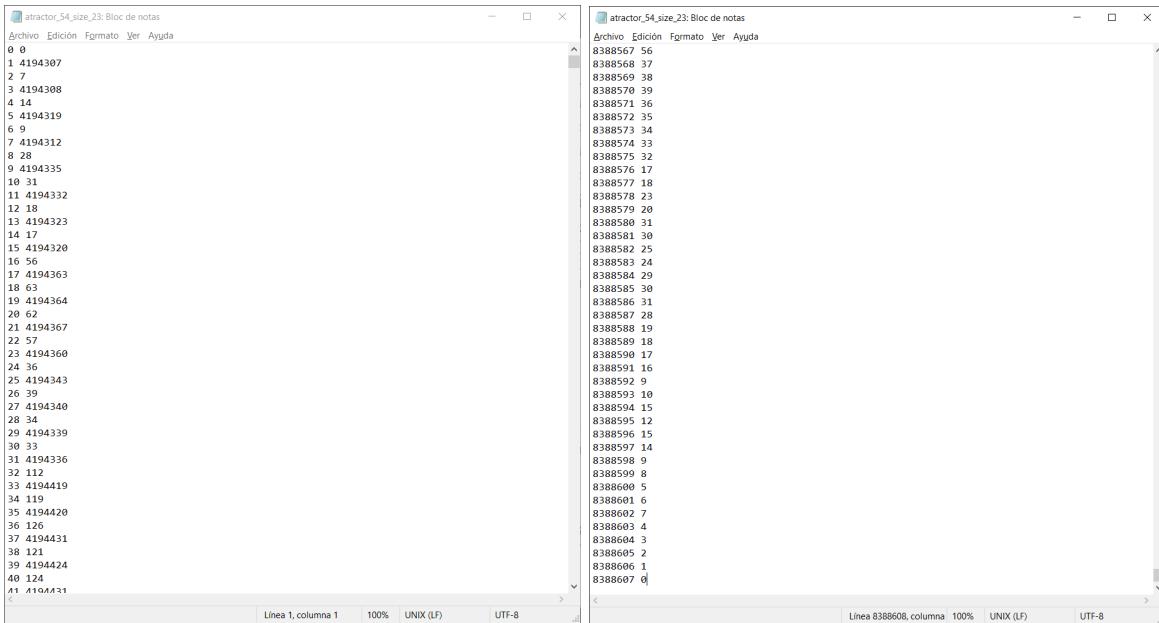


Figura 116: Atractor de tamaño 23.

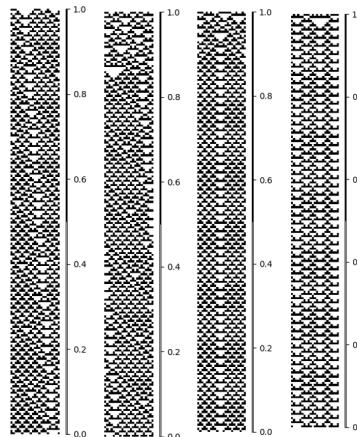


Figura 117: Evoluciones de los atractores.

3.2.23. Tamaño 24

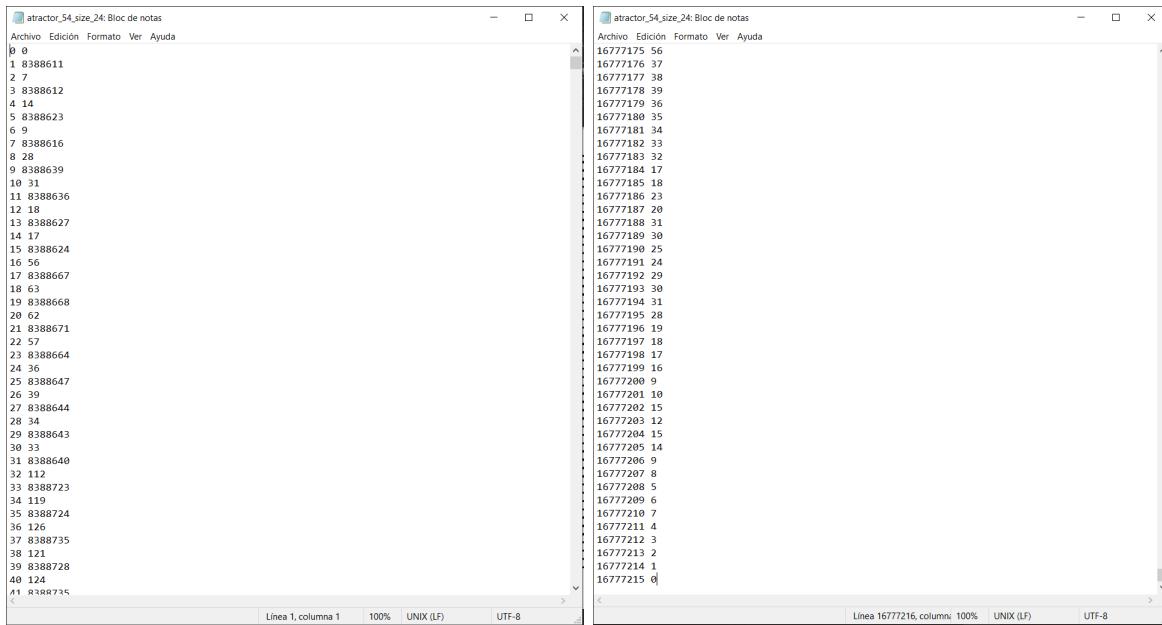


Figura 118: Atractor de tamaño 24.

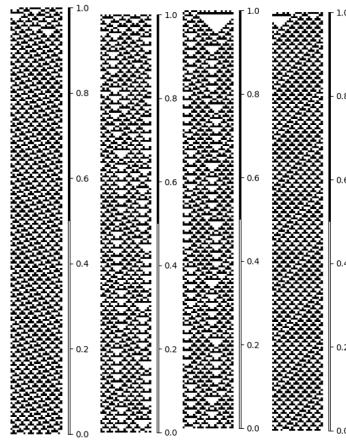


Figura 119: Evoluciones de los atractores.

3.2.24. Tamaño 25

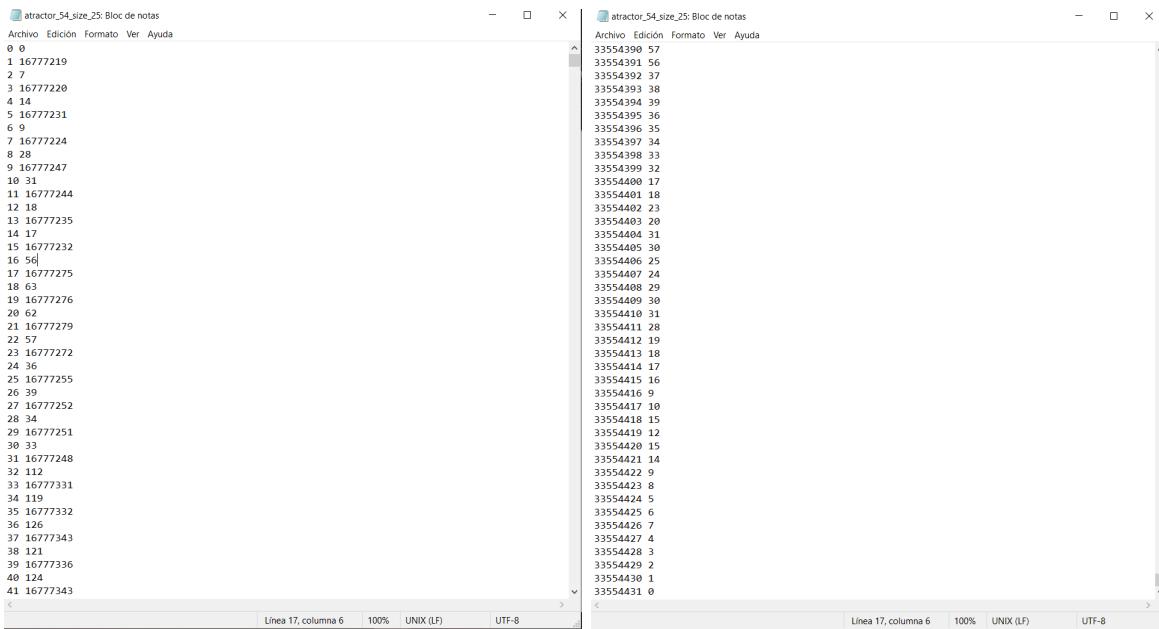


Figura 120: Atractor de tamaño 25.

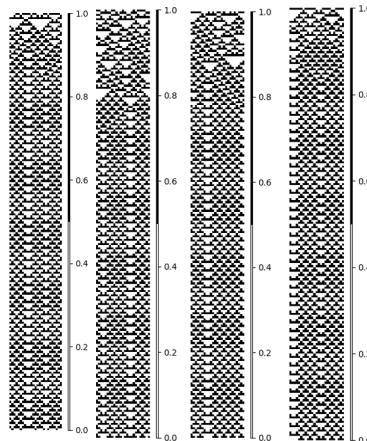


Figura 121: Evoluciones de los atractores.

4. Código

A continuación se anexa el código creado para la parte de la generación de expresiones regulares de forma aleatoria dada una longitud de cadena.

```

1  import random
2  import numpy as np
3
4  regex_22 = "(0+1(01)*00)(0+0(01)*00)*"
5  regex_54 = "(0+1(11)*10)(0+0(11)*10)*"
6
7
8  def fun_klee(num_veces, regex_klee):
9      cadena_klee = ""
10     for i in range(0, num_veces):
11         cadena_klee += regex_klee
12     return cadena_klee
13
14
15 def logica(longitud, regla):
16     cont_long = 0
17     cadena = ""
18     klee_r = bool(random.getrandbits(1))
19     lado_1_izq = bool(random.getrandbits(1))
20     regex_rule = regex_22 if regla == 22 else regex_54
21     if lado_1_izq:
22         cadena += regex_rule[1]
23         cont_long += 1
24     else:
25         klee = bool(random.getrandbits(1))
26         cadena += regex_rule[3]
27         cont_long += 1
28         if klee:
29             valor_klee = int(random.randint(0, longitud-cont_long)/2)

```

```

30         cadena += fun_klee(valor_klee, regex_rule[5:7])
31         cont_long += valor_klee
32     if not klee_r and cont_long+1 < longitud:
33         valor_klee = int((longitud-cont_long+1)/2)
34         cadena += fun_klee(valor_klee, regex_rule[5:7])
35         cont_long += valor_klee
36         cadena += regex_rule[9:11]
37         cont_long += 2
38     if klee_r or lado_1_izq:
39         while cont_long < longitud:
40             lado_1_izq = bool(random.getrandbits(1))
41             if lado_1_izq:
42                 cadena += regex_rule[13]
43                 cont_long += 1
44             else:
45                 klee = bool(random.getrandbits(1))
46                 cadena += regex_rule[15]
47                 cont_long += 1
48                 if klee:
49                     valor_klee = int(random.randint(0, longitud-
50                                     cont_long)/2)
51                     cadena += fun_klee(valor_klee, regex_rule[17:19])
52                     cont_long += valor_klee
53                     cadena += regex_rule[21:23]
54                     cont_long += 2
55
56
57 def start_RegEx(longitud, regla):
58     cadena_generada = logica(longitud, regla)
59     arreglo_RegEx = np.array(list(cadena_generada)).astype(np.int8)
60     return arreglo_RegEx

```

Tenemos un código actualizado para la lógica del programa el cual nos permite poder crear autómatas celulares de un tamaño de 30 mil células por 30 mil pasos en 1 minuto, esto sin la opción de animación, la cual es una característica nueva también, así como la lógica para el calculo de la entropia de Shannon y la densidad de la columna central.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.axes_grid1 import make_axes_locatable
4 import matplotlib.colors as colors
5 from matplotlib.animation import FuncAnimation
6
7 global unosxPaso
8 global probabilidadUnoxPaso
9 global anim_automata
10 global anim_densidad
11 global anim_log10
12 global anim_media
13 global anim_varianza

```

```

14 global anim_densidad_central
15
16
17 # Momento 2 = Sum(x^2*f(x)) - E[x]^2=> Momento 2 = 0*f(x) + 1*f(x) -
18 (1*f(x=1))^2 = f(x=1) - f(x=1)^2
19 def GraficaVarianza():
20     global probabilidadUnoxPaso
21     plot4 = plt.figure(3)
22     plt.title("Variance or second moment")
23     plt.xlabel("Steps")
24     plt.ylabel("Variance")
25     plt.plot(probabilidadUnoxPaso -
26               np.power(probabilidadUnoxPaso, 2), color="purple")
27
28 # E[x] = Sum(x*f(x))=> E[x] = 0*f(x) + 1*f(x) = f(x=1) = Probabilidad
29 # de X=1, osea Probabilidad de uno
30 def GraficaMedia():
31     global probabilidadUnoxPaso
32     plot3 = plt.figure(2)
33     plt.title("Expected value or average")
34     plt.xlabel("Steps")
35     plt.ylabel("Expected value")
36     plt.plot(probabilidadUnoxPaso, color="green")
37
38 def GraficaDensidad():
39     global unosxPaso
40     plot1 = plt.figure(0)
41     plt.title("Numbers of 1's for each step")
42     plt.xlabel("Steps")
43     plt.ylabel("Numbers of 1's")
44     plt.plot(unosxPaso, color="red")
45
46
47 def GraficaLog10():
48     global unosxPaso
49     plot2 = plt.figure(1)
50     plt.title("Log10(Numbers of 1's for each step)")
51     plt.xlabel("Steps")
52     plt.ylabel("Log10(Numbers of 1's)")
53     plt.plot(np.log(unosxPaso), color="blue")
54
55
56 def GraficaDensidadColCentral():
57     global automataCelular
58     indexColCentral = int(len(automataCelular[1])/2-1)
59     datosColCentral = automataCelular[:, indexColCentral]
60     finalDatos = np.zeros((len(datosColCentral), 1))
61     finalDatos[0] = datosColCentral[0]

```

```

62     for pasos in range(1, len(datosColCentral)):
63         finalDatos[pasos] = finalDatos[pasos-1] + datosColCentral [
64             pasos]
65     plot1 = plt.figure(4)
66     plt.title("Central column density")
67     plt.xlabel("Steps")
68     plt.ylabel("Value")
69     plt.plot(finalDatos, color="black")
70
71 #  $H(x) = - \sum_{i=1}^n p(x_i) \log_b p(x_i)$  donde  $p(x_i)$  es la
72 # probabilidad de un evento y  $b = 2 \{0,1\}$ 
73 def GraficaEntropiaShannon():
74     global probabilidadUnoxPaso
75     probabilidadCeroxPaso = 1 - probabilidadUnoxPaso
76     h_x = np.negative((probabilidadUnoxPaso*np.log2(
77         probabilidadUnoxPaso)) +
78                     (probabilidadCeroxPaso*np.log2(
79                         probabilidadCeroxPaso)))
80
81     plot1 = plt.figure(5)
82     plt.title("Shannon entropy")
83     plt.xlabel("Steps")
84     plt.ylabel("Value")
85     plt.plot(h_x, color="orange")
86
87
88
89 def calculoPaso(x, regla):
90     combinaciones3Celulas = np.vstack((np.roll(x, 1), x,
91                                         np.roll(x, -1))).astype(np.
92                                         int8)
93     # Representacion de las 3 celulas, int8 array
94     z = np.sum(combinaciones3Celulas * u, axis=0).astype(np.int8)
95     return regla[7 - z] # Valor de z en la regla
96
97 def Animation_Automata(i):
98     if(i != 0):
99         # Toma las 3 columnas, para llamar la funcion indiceRegla
100        automataCelular[i, :] = calculoPaso(
101            automataCelular[i-1, :], regla).astype(np.int8)
102        cax1.set_array(automataCelular)
103        return cax1,
```

```
104
105
106 def Animation_Densidad(i):
107     global unosxPaso
108     unosxPaso = np.count_nonzero(automataCelular == 1, axis=1)
109     valor_y_densidad.append(unosxPaso[i])
110     valor_x.append(i)
111     ax1.plot(valor_x, valor_y_densidad, color="red")
112     return ax1,
113
114
115 def Animation_Log10(i):
116     global unosxPaso
117     valor_y_log10.append(np.log10(unosxPaso[i]))
118     ax2.plot(valor_x, valor_y_log10, color="blue")
119     return ax2,
120
121
122 def Animation_Varianza(i):
123     global unosxPaso
124     global probabilidadUnoxPaso
125     valor_y_varianza.append(
126         probabilidadUnoxPaso[i]-np.power(probabilidadUnoxPaso[i], 2))
127     ax3.plot(valor_x, valor_y_varianza, color="purple")
128     return ax3,
129
130
131 def Animation_Media(i, len_arreglo):
132     global unosxPaso
133     global probabilidadUnoxPaso
134     probabilidadUnoxPaso = unosxPaso / len_arreglo
135     valor_y_media.append(probabilidadUnoxPaso[i])
136     ax4.plot(valor_x, valor_y_media, color="green")
137     return ax4,
138
139
140 def Animation_DensidadColCentral(i, indexColCentral):
141     global automataCelular
142     datosColCentral = automataCelular[:, indexColCentral]
143     if (i == 0 and len(valor_y_densidad_central) != 1):
144         valor_x_densidad_central.append(i)
145         valor_y_densidad_central.append(datosColCentral[0])
146     if (i > 0):
147         valor_x_densidad_central.append(i)
148         valor_y_densidad_central.append(
149             valor_y_densidad_central[i-1] + datosColCentral[i])
150     ax5.plot(valor_x_densidad_central, valor_y_densidad_central,
151             color="black")
152     return ax5,
```

```
153
154 def Animation_Entropia(i):
155     global probabilidadUnoxPaso
156     probabilidadCeroxPaso = 1 - probabilidadUnoxPaso
157     h_x = np.negative((probabilidadUnoxPaso[i]*np.log2(
158         probabilidadUnoxPaso[i])) +
159             (probabilidadCeroxPaso[i]*np.log2(
160                 probabilidadCeroxPaso[i])))
161     valor_y_entropia.append(h_x)
162     ax6.plot(valor_x, valor_y_entropia, color="orange")
163     return ax6,
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
```

```

200      # -----
201      # Representamos la regla ahora en base 2 como una cadena (8 bits)
202      # es un numero
203      reglaString = np.binary_repr(reglaNumero, width=8)
204      regla = np.array([int(bit) for bit in reglaString]).astype(np.
205          int8)
206      automataCelular = np.zeros((nPasos, len(arreglo))).astype(np.int8
207          )
208      probabilidadUnoxPaso = np.empty(nPasos, dtype=float).astype(np.
209          int8)
210      automataCelular[0, :] = arreglo # Fila 1
211      if(isAnimado == 1):
212          #--Grafica automata--#
213          fig, ax = plt.subplots(figsize=(16, 9))
214          cax1 = ax.matshow(automataCelular, cmap=cmap)
215          ax.axis(False)
216          divider = make_axes_locatable(ax)
217          cax = divider.append_axes("right", size="3%", pad=0.1)
218          plt.colorbar(cax1, cax=cax)
219          #--Grafica densidad--#
220          plot1 = plt.figure(0)
221          ax1 = plt.subplot()
222          plot1.suptitle("Numbers of 1's for each step")
223          plt.xlabel("Steps")
224          plt.ylabel("Numbers of 1's")
225          #--Grafica log10--#
226          plot2 = plt.figure(2)
227          ax2 = plt.subplot()
228          plt.title("Log10(Numbers of 1's for each step)")
229          plt.xlabel("Steps")
230          plt.ylabel("Log10(Numbers of 1's)")
231          #--Grafica Media--#
232          plot4 = plt.figure(3)
233          ax4 = plt.subplot()
234          plt.title("Expected value or average")
235          plt.xlabel("Steps")
236          plt.ylabel("Expected value")
237          #--Grafica Varianza--#
238          plot3 = plt.figure(4)
239          ax3 = plt.subplot()
240          plt.title("Variance or second moment")
241          plt.xlabel("Steps")
242          plt.ylabel("Variance")
243          #--Grafica Densidad central--#
244          plot5 = plt.figure(5)
245          ax5 = plt.subplot()
246          plt.title("Central column density")
247          plt.xlabel("Steps")
248          plt.ylabel("Value")
249          indexColCentral = int(len(automataCelular[0])/2-1)

```

```

246     #--Grafica Entropia Shannon--#
247     plot6 = plt.figure(6)
248     ax6 = plt.subplot()
249     plt.title("Shannon entropy")
250     plt.xlabel("Steps")
251     plt.ylabel("Value")
252     #--Animaciones--#
253     anim_automata = FuncAnimation(
254         fig, Animation_Automata, frames=nPasos, interval=8,
255         repeat=False)
255     anim_densidad = FuncAnimation(
256         plot1, Animation_Densidad, frames=nPasos, interval=8,
257         repeat=False)
257     anim_log10 = FuncAnimation(
258         plot2, Animation_Log10, frames=nPasos, interval=8, repeat
259         =False)
259     anim_media = FuncAnimation(plot4, Animation_Media, fargs=(
260         len(arreglo),), frames=nPasos, interval=8, repeat=False)
261     anim_varianza = FuncAnimation(
262         plot3, Animation_Varianza, frames=nPasos, interval=8,
263         repeat=False)
263     anim_densidad_central = FuncAnimation(plot5,
264         Animation_DensidadColCentral, fargs=(
265             indexColCentral,), frames=nPasos, interval=8, repeat=
266             False)
265     anim_entropia = FuncAnimation(
266         plot6, Animation_Entropia, frames=nPasos, interval=8,
267         repeat=False)
267     plt.show()
268 else:
269     for pasos in range(1, nPasos):
270         # Toma las 3 columnas, para llamar la funcion indiceRegla
271         automataCelular[pasos, :] = calculoPaso(
272             automataCelular[pasos-1, :], regla)
273         # despues empareja cada valor de nuestros indices con el
274         # valor en la regla 30, guardamos en el arreglo final
275         unosxPaso = np.count_nonzero(automataCelular == 1, axis=1)
276         probabilidadUnoxPaso = unosxPaso / len(arreglo)
277
278 def IniciarArchivo(Color0, Color1, Pasos, Regla, CelulasIniciales,
279     isAnimado):
280     CelulasIniciales = CelulasIniciales.astype(np.int8)
281     cmap = colors.ListedColormap([Color0, Color1]) # 0,1
282     GenerarAutomataCelular(CelulasIniciales, Pasos, Regla, cmap,
283     isAnimado)
284     if(isAnimado == 0):
285         GraficaDensidad()
286         GraficaLog10()
287         GraficaMedia()

```

```

286     GraficaVarianza()
287     GraficaDensidadColCentral()
288     GraficaEntropiaShannon()
289     cmap = colors.ListedColormap([Color0, Color1]) # 0,1
290     fig, ax = plt.subplots(figsize=(16, 9))
291     im = ax.matshow(automataCelular, cmap=cmap)
292     ax.axis(False)
293     divider = make_axes_locatable(ax)
294     cax = divider.append_axes("right", size="3%", pad=0.1)
295     plt.colorbar(im, cax=cax)
296     plt.show()
297
298
299 def IniciarRandom(Color0, Color1, Pasos, Regla, TamanioAutomata,
300    isAnimado):
301     CelulasIniciales = np.random.randint(
302         2, size=TamanioAutomata).astype(np.int8)
303     cmap = colors.ListedColormap([Color0, Color1]) # 0,1
304     GenerarAutomataCelular(CelulasIniciales, Pasos, Regla, cmap,
305        isAnimado)
306     if(isAnimado == 0):
307         GraficaDensidad()
308         GraficaLog10()
309         GraficaMedia()
310         GraficaVarianza()
311         GraficaDensidadColCentral()
312         GraficaEntropiaShannon()
313         cmap = colors.ListedColormap([Color0, Color1]) # 0,1
314         fig, ax = plt.subplots(figsize=(16, 9))
315         im = ax.matshow(automataCelular, cmap=cmap)
316         ax.axis(False)
317         divider = make_axes_locatable(ax)
318         cax = divider.append_axes("right", size="3%", pad=0.1)
319         plt.colorbar(im, cax=cax)
320         plt.show()
321
322 def IniciarEspecifico(Color0, Color1, Pasos, Regla, TamanioAutomata,
323    LugaresUnos, isAnimado):
324     CelulasIniciales = np.zeros((TamanioAutomata,), dtype=int)
325     for i in LugaresUnos:
326         CelulasIniciales[i] = 1
327     cmap = colors.ListedColormap([Color0, Color1]) # 0,1
328     GenerarAutomataCelular(CelulasIniciales, Pasos, Regla, cmap,
329        isAnimado)
330     if(isAnimado == 0):
331         GraficaDensidad()
332         GraficaLog10()
333         GraficaMedia()
334         GraficaVarianza()

```

```

332     GraficaDensidadColCentral()
333     GraficaEntropiaShannon()
334     cmap = colors.ListedColormap([Color0, Color1]) # 0,1
335     fig, ax = plt.subplots(figsize=(16, 9))
336     im = ax.matshow(automataCelular, cmap=cmap)
337     ax.axis(False)
338     divider = make_axes_locatable(ax)
339     cax = divider.append_axes("right", size="3%", pad=0.1)
340     plt.colorbar(im, cax=cax)
341     plt.show()
342
343
344 def IniciarProbabilidad(Color0, Color1, Pasos, Regla, TamanioAutomata,
345 , LugaresUnos, isAnimado):
346     CelulasIniciales = np.random.choice([0, 1], TamanioAutomata, p=[
347                                     1-LugaresUnos, LugaresUnos]).
348                                     astype(np.int8)
349     cmap = colors.ListedColormap([Color0, Color1]) # 0,1
350     GenerarAutomataCelular(CelulasIniciales, Pasos, Regla, cmap,
351     isAnimado)
352     if(isAnimado == 0):
353         GraficaDensidad()
354         GraficaLog10()
355         GraficaMedia()
356         GraficaVarianza()
357         GraficaDensidadColCentral()
358         GraficaEntropiaShannon()
359         cmap = colors.ListedColormap([Color0, Color1]) # 0,1
360         fig, ax = plt.subplots(figsize=(16, 9))
361         im = ax.matshow(automataCelular, cmap=cmap)
362         ax.axis(False)
363         divider = make_axes_locatable(ax)
364         cax = divider.append_axes("right", size="3%", pad=0.1)
365         plt.colorbar(im, cax=cax)
366         plt.show()

```

También la parte del código de la interfaz sufrió un cambio y es que ahora tenemos la opción de animar el autómata y detener dicha animación cuando nos plazca, aunado a eso tenemos la opción de hacer uso de la expresión regular para la regla 22 y 54.

```

1 #IMPORTAMOS LIBRERIAS NECESARIAS.
2 import tkinter
3 from tkinter import colorchooser
4 from tkinter.filedialog import askopenfilename
5 import numpy as np
6 import Logica as cellularAutomaton
7 import matplotlib.pyplot as plt
8 import RegExGenerator as RegEx
9 #-----CREAR VENTANA
10 root = tkinter.Tk()

```

```
11 root.wm_title("Elementary Cellular Automaton")
12 frame1 = tkinter.Frame(root)
13 frameColor0 = tkinter.Frame(root)
14 frameColor1 = tkinter.Frame(root)
15 frameCondicion = tkinter.Frame(root)
16 frameMainInformacion = tkinter.Frame(root)
17 frameStart = tkinter.Frame(root)
18 #-----FUNCIONES
19 -----
20 global color0
21 global color1
22 global arregloInicial
23 global detener
24 esBinario = tkinter.IntVar()
25 reglaValor = tkinter.StringVar()
26 tamEspacio = tkinter.StringVar()
27 numIteraciones = tkinter.StringVar()
28 esAleatorio = tkinter.IntVar()
29 esEspecifico = tkinter.IntVar()
30 posicionesUnos = tkinter.StringVar()
31 esAnimado = tkinter.IntVar()
32 useRegex = tkinter.IntVar()
33
34 color0 = "#ffffff"
35 color1 = "#000000"
36 arregloInicial = []
37 detener = False
38
39 def chageLabel():
40     if(esEspecifico.get() == 0):
41         labelEspecifico.grid_remove()
42         las.grid_remove()
43     else:
44         las.grid()
45         labelEspecifico.grid()
46
47 def colorto0():
48     global color0
49     color0 = colorchooser.askcolor()[1]
50     showColor0.config(bg=color0)
51
52 def colorto1():
53     global color1
54     color1 = colorchooser.askcolor()[1]
55     showColor1.config(bg=color1)
56
57 def saveCondic():
58     celulasIniciales = cellularAutomaton.automataCelular[0,:]
59     with open('celulasIniciales.txt', 'wb') as f:
60         np.savetxt(f, np.column_stack(celulasIniciales), fmt='%.1f')
```

```
60     tkinter.messagebox.showinfo("Save condition", "Successfully saved  
       ")  
61  
62 def fileselectD():  
63     global arregloInicial  
64     try:  
65         filename = askopenfilename()  
66         if (filename.find(".txt") != -1):  
67             labelLoadCondicion.config(text = filename)  
68             arregloInicial = np.loadtxt(filename)  
69         else:  
70             arregloInicial = []  
71             labelLoadCondicion.config(text = "Choose a file .txt")  
72     except:  
73         arregloInicial = []  
74         labelLoadCondicion.config(text = "File Error")  
75  
76 def messageAnim():  
77     if(esAnimado.get() == 1):  
78         tkinter.messagebox.showwarning(title="Warning", message="If  
           you have a lot of iterations and you need the results  
           quickly, I don't recommend enable this option")  
79  
80 def stopcontAnim():  
81     global detener  
82     detener ^= True  
83     if detener:  
84         cellularAutomaton.anim_automata.event_source.stop()  
85         cellularAutomaton.anim_densidad.event_source.stop()  
86         cellularAutomaton.anim_log10.event_source.stop()  
87         cellularAutomaton.anim_media.event_source.stop()  
88         cellularAutomaton.anim_varianza.event_source.stop()  
89         cellularAutomaton.anim_densidad_central.event_source.stop()  
90     else:  
91         cellularAutomaton.anim_automata.event_source.start()  
92         cellularAutomaton.anim_densidad.event_source.start()  
93         cellularAutomaton.anim_log10.event_source.start()  
94         cellularAutomaton.anim_media.event_source.start()  
95         cellularAutomaton.anim_varianza.event_source.start()  
96         cellularAutomaton.anim_densidad_central.event_source.start()  
97  
98 def deletePlt():  
99     global arregloInicial  
100    plt.close("all")  
101    arregloInicial = []  
102  
103 def iniciar():  
104     global arregloInicial  
105     plt.close("all")  
106     numPasos = int(numIteraciones.get())
```

```

107     ruleEntero = int(reglaValor.get())
108     animacion = int(esAnimado.get())
109     usarexpre = int(useRegex.get())
110     if(esBinario.get() == 1):
111         ruleString = reglaValor.get()
112         ruleEntero = int(ruleString,2)
113     if(len(arregloInicial) != 0 or usarexpre==1):
114         if(usarexpre==1):
115             tamEsp = int(tamEspacio.get())
116             arregloInicial = RegEx.start_RegEx(tamEsp,ruleEntero)
117             cellularAutomaton.IniciarArchivo(color0,color1,numPasos,
118                 ruleEntero,arregloInicial,animacion)#color 0, color 1,
119                 pasos,regla,
120             else:
121                 tamEsp = int(tamEspacio.get())
122                 if(esAleatorio.get()==1):
123                     cellularAutomaton.IniciarRandom(color0,color1,numPasos,
124                         ruleEntero,tamEsp,animacion)
125                 else:
126                     if '%' in posicionesUnos.get():
127                         unosEntero = int(posicionesUnos.get().replace('%',''))
128                             /100
129                         cellularAutomaton.IniciarProbabilidad(color0,color1,
130                             numPasos,ruleEntero,tamEsp,unosEntero,animacion)
131             else:
132                 unosString = posicionesUnos.get().split(',')
133                 unosArray = list(map(int, unosString))
134                 cellularAutomaton.IniciarEspecifico(color0,color1,
135                     numPasos,ruleEntero,tamEsp,unosArray,animacion)

136 #-----CREAR INTERFAZ
137 -----
138 titulo = tkinter.Label(framet, text="Elementary Cellular Automaton.",
139     font=("times new roman", 24))
140 titulo.pack(side="top")

141 #-----COLOR
142 0-----
143 color0Label = tkinter.Label(frameColor0, text="Choose color to
144     represent 0 in the cellular automaton:",font=("times new roman",
145     14))
146 color0Label.pack(side="top")
147 buttonColor0 = tkinter.Button(frameColor0, text = "Choose a color",
148     command=colorto0,font=("times new roman", 14))
149 buttonColor0.pack(side="left")
150 showColor0 = tkinter.Label(frameColor0, text="This is the color to
151     use in 0 value",font=("times new roman", 14))
152 showColor0.pack(side="left")

```

```

144
145 #-----COLOR
146   1-----
147 color1Label = tkinter.Label(frameColor1, text="Choose color to
148   represent 1 in the cellular automaton:", font=("times new roman",
149   14))
150 color1Label.pack(side="top")
151 buttonColor1 = tkinter.Button(frameColor1, text = "Choose a color",
152   command=colorto1 ,font=("times new roman", 14))
153 buttonColor1.pack(side="left")
154 showColor1 = tkinter.Label(frameColor1, text="This is the color to
155   use in 1 value",font=("times new roman", 14))
156 showColor1.pack(side="left")
157
158 #-----CONDICIONES ARCHIVO
159 -----
160
161 tittleCondicion = tkinter.Label(frameCondicion, text = "Initial
162   condition with files" ,font=("times new roman", 18))
163 tittleCondicion.pack(side="top")
164 saveCondicion = tkinter.Button(frameCondicion, text = "Save condition
165   ", command=saveCondic ,font=("times new roman", 14))
166 saveCondicion.pack(side="top")
167 loadCondicion = tkinter.Button(frameCondicion, text = "Load condition
168   ", command=fileselectD ,font=("times new roman", 14))
169 loadCondicion.pack(side="left")
170 labelLoadCondicion = tkinter.Label(frameCondicion, text = "Choose File
171   .txt",font=("Times New Roman",14))
172 labelLoadCondicion.pack(side="left")
173
174 #-----CONDICION INICIAL USUARIO
175 -----
176 tittleCondicionUsuario = tkinter.Label(frameMainInformacion, text = "Initial
177   condition by user" ,font=("times new roman", 18))
178 tittleCondicionUsuario.grid(row = 0, column = 0,columnspan=2)
179
180 tamanoEspacio = tkinter.Label(frameMainInformacion, text = "Space
181   size: " , font=("times new roman", 14))
182 tamanoEspacio.grid(row=1, column=0)
183 entradaEspacio = tkinter.Entry(frameMainInformacion,font=("times new
184   roman", 14), textvariable = tamEspacio)
185 entradaEspacio.grid(row=1, column=1)
186
187 numeroIteracion = tkinter.Label(frameMainInformacion, text = "Number
188   of iterations: " ,font=("times new roman", 14))
189 numeroIteracion.grid(row=2, column=0)
190 entradaIteracion = tkinter.Entry(frameMainInformacion,font=("times
191   new roman", 14), textvariable = numIteraciones)
192 entradaIteracion.grid(row=2, column=1)

```

```

178
179 reglaUsar = tkinter.Label(frameMainInformacion, text = "Rule to use:
    " ,font=("times new roman", 14))
180 reglaUsar.grid(row=3, column=0)
181 entradaRegla = tkinter.Entry(frameMainInformacion,font=("times new
    roman", 14), textvariable = reglaValor)
182 entradaRegla.grid(row=3, column=1)
183 binary = tkinter.Checkbutton(frameMainInformacion, text="Is binary?", 
    variable=esBinario, font=("times new roman", 13))
184 binary.grid(row=4, column=1)
185
186
187 labelCondicion = tkinter.Label(frameMainInformacion, text = "Type of
    inicial condition: " , font=("times new roman", 14))
188 labelCondicion.grid(row=5, column = 0)
189 checkAleatoria = tkinter.Checkbutton(frameMainInformacion, text="Random",
    variable=esAleatorio, font=("times new roman", 14))
190 checkAleatoria.grid(row=5, column=1)
191 tipoCondicion = tkinter.Checkbutton(frameMainInformacion, text = "Specific",
    variable=esEspecifico, command=chageLabel,font=("times
    new roman", 14))
192 tipoCondicion.grid(row=5, column = 3)
193 expresionRegular = tkinter.Checkbutton(frameMainInformacion, text = "Use regular expression (rule 22 & 54 only)", variable=useRegex,
    font=("times new roman", 14))
194 expresionRegular.grid(row=7, column = 0, columnspan=2)
195 labelEspecifico = tkinter.Label(frameMainInformacion, text = "Indices
    to complete with 1 or use % \nat the end to assign probability to
    1's:" , font=("times new roman", 14))
196 labelEspecifico.grid(row=6, column = 0)
197 las = tkinter.Entry(frameMainInformacion,font=("times new roman", 14),
    , textvariable = posicionesUnos)
198 las.grid(row=6, column = 1)
199 labelEspecifico.grid_remove()
200 las.grid_remove()
201
202 #-----ANIMACION -----
203 checkAnimado = tkinter.Checkbutton(frameMainInformacion, text="Enable
    Animation", variable=esAnimado, command=messageAnim, font=("times
    new roman", 14))
204 checkAnimado.grid(row=8, column=0)
205
206 buttonStop = tkinter.Button(frameMainInformacion, text="Stop/Continue
    animation", command=stopcontAnim, font=("times new roman", 14))
207 buttonStop.grid(row=8, column=1)
208
209 buttonDeletePlt = tkinter.Button(frameMainInformacion, text="Delete
    all plots", command=deletePlt, font=("times new roman", 14))
210 buttonDeletePlt.grid(row=9, column=0)
211

```

```

212 #-----BOTON MAESTRO
213 -----
214 buttonStart = tkinter.Button(frameStart, text = "Start", command=
215     iniciar ,font=("times new roman", 14))
216 buttonStart.pack(side="top")
217 framet.grid(row=0, column=0, columnspan=1)
218 frameColor0.grid(row=1, column=0)
219 frameColor1.grid(row=2, column=0)
220 frameCondicion.grid(row=3, column=0)
221 frameMainInformacion.grid(row=4, column=0)
222 frameStart.grid(row=5, column=0)
223
224 tkinter.mainloop()

```

Código para el calculo de atractores, modificado talque la longitud máxima menos 1 para poder visualizar la información en formato de imagen, ademas de ordenarla por medio de carpetas y con su tamaño correspondiente.

```

1 import numpy as np
2 import time
3 import networkx as nx
4 import os
5
6 # Hacemos el calculo por el paso correspondiente, juntamos la parte
7 # de obtener las 3 combinaciones posibles y la fun Indice regla
8 # esto provoca optimizar el codigo, pasamos de 3.3 s para el calculo
9 # de un ECA de 1kx1k a 0.07s
10 u = np.array([[4], [2], [1]]) # Arreglo para convertir binario a
11 decimal
12
13 def calculoPaso(x, regla):
14     combinaciones3Celulas = np.vstack((np.roll(x, 1), x,
15                                         np.roll(x, -1))).astype(np.
16                                         int8)
17     # Representacion de las 3 celulas, int8 array
18     z = np.sum(combinaciones3Celulas * u, axis=0).astype(np.int8)
19     return regla[7 - z] # Valor de z en la regla
20
21 def AutomataCelular(arreglo, regla):
22     pasada = arreglo
23     final = []
24     condicion = True
25     while condicion:
26         # Toma las 3 columnas, para llamar la funcion indiceRegla
27         actual = calculoPaso(pasada, regla)
28         pasada = actual
29         final = actual

```

```
28     if ((actual == pasada).all()):
29         condicion = False
30     return [arreglo.tolist(), final.tolist()]
31
32
33 def CalculadoraCombinaciones(longitud):
34     # Producto cartesiano de 0,1 de con longitud n.
35     return np.array([0, 1])[np.rollaxis(
36         np.indices((len([0, 1]),) * longitud), 0, longitud + 1)
37         .reshape(-1, longitud)]
38
39
40 def regla_representacion(reglaNumero):
41     reglaString = np.binary_repr(reglaNumero, width=8)
42     return np.array([int(bit) for bit in reglaString]).astype(np.int8)
43
44
45 def ListaBinariatoNumero(lista):
46     return int("".join(str(i) for i in lista), 2)
47
48
49 def crear_carpetas(dirName):
50     try:
51         os.makedirs(dirName)
52         print("Directory ", dirName, " Created ")
53     except FileExistsError:
54         print("Directory ", dirName, " already exists")
55
56
57 def main():
58     carpeta_22 = "Atractores_regla_22"
59     carpeta_54 = "Atractores_regla_54"
60     crear_carpetas(carpeta_22)
61     crear_carpetas(carpeta_54)
62     longitud_max = 19
63     regla_22 = regla_representacion(22)
64     regla_54 = regla_representacion(54)
65     t0 = time.time()
66     for longitud in range(2, longitud_max):
67         combinaciones = CalculadoraCombinaciones(longitud)
68         atractoresList_22 = []
69         atractoresList_54 = []
70         for inicial in combinaciones:
71             atractores_22 = AutomataCelular(inicial, regla_22)
72             atractoresList_22.append((ListaBinariatoNumero(
73                 atractores_22[-1]), ListaBinariatoNumero(
74                     atractores_22[0])))
75             atractores_54 = AutomataCelular(inicial, regla_54)
76             atractoresList_54.append((ListaBinariatoNumero(
```

```

76         atractores_54[-1]), ListaBinariatoNumero(
77             atractores_54[0]))
78     atractor_22_nombre = os.path.join(
79         carpeta_22, 'atractor_22_size_'+str(longitud)+'.graphml')
80     atractor_54_nombre = os.path.join(
81         carpeta_54, 'atractor_54_size_'+str(longitud)+'.graphml')
82     g_22 = nx.MultiGraph()
83     g_22.add_edges_from(atractoresList_22)
84     nx.write_graphml(g_22, atractor_22_nombre)
85     g_54 = nx.MultiGraph()
86     g_54.add_edges_from(atractoresList_54)
87     nx.write_graphml(g_54, atractor_54_nombre)
88     t1 = time.time()
89     total = t1-t0
90     print(total)
91
92 if __name__ == "__main__":
93     main()

```

Código para el calculo de atractores mediante el uso de Datalore y guardando en un archivo .txt con el nodo inicial del lado izquierdo y el nodo final del lado derecho.

```

1 import numpy as np
2 import time
3 import os
4
5 # Hacemos el calculo por el paso correspondiente, juntamos la parte
6 # de obtener las 3 combinaciones posibles y la fun Indice regla
7 # esto provoca optimizar el codigo, pasamos de 3.3 s para el calculo
8 # de un ECA de 1kx1k a 0.07s
9 u = np.array([[4], [2], [1]]) # Arreglo para convertir binario a
10 decimal
11
12 def calculoPaso(x, regla):
13     combinaciones3Celulas = np.vstack((np.roll(x, 1), x,
14                                         np.roll(x, -1))).astype(np.
15                                         int8)
16     # Representacion de las 3 celulas, int8 array
17     z = np.sum(combinaciones3Celulas * u, axis=0).astype(np.int8)
18     return regla[7 - z] # Valor de z en la regla
19
20 def AutomataCelular(arreglo, regla):
21     pasada = arreglo
22     final = []
23     condicion = True
24     while condicion:
25         # Toma las 3 columnas, para llamar la funcion indiceRegla
26         actual = calculoPaso(pasada, regla)

```

```
25     pasada = actual
26     final = actual
27     if ((actual == pasada).all()):
28         condicion = False
29     return [arreglo.tolist(), final.tolist()]
30
31
32 def CalculadoraCombinaciones(longitud):
33     # Producto cartesiano de 0,1 de con longitud n.
34     return np.array([0, 1])[np.rollaxis(
35         np.indices((len([0, 1]),) * longitud), 0, longitud + 1)
36         .reshape(-1, longitud)]
37
38
39 def regla_representacion(reglaNumero):
40     reglaString = np.binary_repr(reglaNumero, width=8)
41     return np.array([int(bit) for bit in reglaString]).astype(np.int8)
42
43
44 def ListaBinariatoNumero(lista):
45     return int("".join(str(i) for i in lista), 2)
46
47
48 def crear_carpetas(dirName):
49     try:
50         os.makedirs(dirName)
51         print("Directory ", dirName, " Created ")
52     except FileExistsError:
53         print("Directory ", dirName, " already exists")
54
55
56 def main():
57     carpeta_22 = "Atractores_regla_22"
58     carpeta_54 = "Atractores_regla_54"
59     crear_carpetas(carpeta_22)
60     crear_carpetas(carpeta_54)
61     longitud_max = 32
62     regla_22 = regla_representacion(22)
63     regla_54 = regla_representacion(54)
64     t0 = time.time()
65     for longitud in range(26, longitud_max):
66         combinaciones = CalculadoraCombinaciones(longitud)
67         atractoresList_22 = []
68         atractoresList_54 = []
69         for inicial in combinaciones:
70             atractores_22 = AutomataCelular(inicial, regla_22)
71             atractoresList_22.append((ListaBinariatoNumero(
72                 atractores_22[0]), ListaBinariatoNumero(atractores_22
73                 [-1])))
```

```

73         atractores_54 = AutomataCelular(inicial, regla_54)
74         atractoresList_54.append((ListaBinariatoNumero(
75             atractores_54[0]), ListaBinariatoNumero(atractores_54
76             [-1])))
77
77     atractor_22_nombre = os.path.join(
78         carpeta_22, 'atractor_22_size_'+str(longitud)+'.txt')
79     atractor_54_nombre = os.path.join(
80         carpeta_54, 'atractor_54_size_'+str(longitud)+'.txt')
81
82     with open(atractor_22_nombre, 'w') as fp:
83         fp.write('\n'.join('{} {}'.format(x[0],x[1]) for x in
84             atractoresList_22))
84     with open(atractor_54_nombre, 'w') as fp:
85         fp.write('\n'.join('{} {}'.format(x[0],x[1]) for x in
86             atractoresList_54))
86     t1 = time.time()
87     total = t1-t0
88     print(total)
89
90
91 if __name__ == "__main__":
92     main()

```

5. Conclusiones

Sin duda una practica sumamente interesante, ya que no solo vimos como controlar de una manera a nuestros autómatas con determinadas reglas mediante las expresiones regulares y como estas nos generan patrones compartidos entre autómatas, pero sin duda alguna lo mas interesante y complicado fue el calculo de los atractores ya que estamos hablando de estos creces exponencialmente, tan solo para el tamaño 18 el numero total de nodos es de 262144 y ni hablar de tamaños superiores, en lo personal fue complicado ya que mi equipo no es el mejor ni el mas nuevo y eso se ve reflejado en el tamaño que puede manejar, pero mediante la ayuda de servicios como Datalore nos aumenta la potencia de computo, pero aun así me enfrenté al problema de gastar toda la RAM inclusive en la versión de paga, que por cierto no es barata, llegue a calcular el tamaño 25 en el cual estamos hablando de 33554432 nodos, generando un archivo de medio Gb, algo realmente impresionante y sin duda interesante, en especial me pude percatar que para la regla 22 el atractor principal era el nodo 0 mientras que para la regla 54 no contaba con uno principal, tenia varios, esto nos ocasiona observar atractores de diferentes formas.

Referencias

- The Global Dynamics of Cellular Automata; An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata. Addison-Wesley, 1992.
- Simple networks on complex cellular automata: From de Bruijn diagrams to jump-graphs. In: Swarm Dynamics as a Complex Network, Springer, (I. Zelinka and G. Chen Eds.), chapter 12, pages 241-264, 2018.