UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"ALGORITMOS PARA LA DETECCION DE PATRONES CLASE IV EN JUEGO DE LA VIDA Y JUEGOS POBLACIONALES".

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

P R E S E N T A :

MARIANO DOMINGUEZ MOLINA

DIRECTOR DE TESIS: DR. PEDRO MIRAMONTES VIDAL.

FACULTAD DE CIENCIAS
SECCION ESCOLAR

UNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

MAT. MARGARITA ELVIRA CHÁVEZ CANO
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo de Tesis:

Algoritmos para detección de patrones clase IV en juego de la vida y juegos poblacionales.

realizado por Mariano Domínguez Molina

con número de cuenta 8917736-1 , pasante de la carrera de Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Dr. Pedro Miramontes Vidal
Propietario	Dr. Ricardo Mansilla
Propietario	M. en C. Guadalupe Ibargüengoitia González
Suplente	M en C. Mónica Leñero Padierna
Suplente	M. en C. José Antonio Gómez Ortega

Consejo Departamental de Matemáticas

M. en C. Alejandro Bravo Mojica

Agradecimientos

A mi madre, por tantos sacrificios a lo largo de tantos años.

A mi pareja Ligia , por todo su trabajo, amor y entusiasmo.

A mi hermana Dení. No sé por qué, pero gracias.

Al maestrísimo Dr. Pedro Miramontes Vidal,
por su infinita paciencia, su asesoría y ejemplo.

A la UNAM, a la Facultad de Ciencias,
y a todos aquellos maestros que la
llevan día con día por puro amor al arte.

A mi querida Potenciana Torrija Thyssen-Bornemisza

Junio del 2001

Índice General

Capítulo 1

Introducción.

El modelo de Autómata Celular (AC) fue planteado por Von Neumann en 1943 [18] , como un modelo en el que se buscaba obtener un comportamiento de auto-reproducción en una máquina. A partir de éste se produjeron una serie de resultados y desarrollos posteriores que pusieron el estudio de los AC dentro de las ramas de la matemática que siguen arrojando resultados a la fecha: la sencillez de definición y construcción en un AC contrasta con la complejidad de su comportamiento.

La teoría de los AC, y en general, la teoría de los Autómatas y Lenguajes Formales, obtuvo un gran impulso con el desarrollo posterior de las computadoras; puesto que actualmente se pueden realizar simulaciones y observaciones anteriormente limitadas por la velocidad de las máquinas. Las Ciencias de la Computación han proporcionado herramientas que permiten esclarecer un poco el carácter de los AC; sobre todo en los temas sobre computabilidad y universalidad.

El interés público en los AC, se puede atribuir al trabajo de John H. Conway enfocado encontrar una especificación más simple que la de Von Neumann y explorar sus capacidades. Inspirado en conceptos biológicos muy simples, Conway presentó en 1970 un juego ecológico llamado *Juego de la Vida (o vida)*, que a partir de su aparición en la revista Scientific American [19], inspiró una investigación intensa, tanto de aficionados que buscaban patrones como de teóricos que intentaban investigar la inesperada complejidad observada. Muchos resultados interesantes fueron encontrados por el grupo del Laboratorio de Inteligencia Artificial en el MIT, corriendo simulaciones en una computadora PDP-6.

Posteriormente, el trabajo de Stephen Wolfram [15] se encaminó a estudiar las propiedades generales de los AC (al principio restringiéndose a autómatas unidimensionales), con lo que obtuvo nuevos resultados. Auxiliado por la mecánica estadística y la teoría de sistemas dinámicos contínuos, le dió un enfoque distinto al que se había desatado con *vida*. Más que buscar patrones "interesantes", se pensó en la conducta a largo plazo de un número de células con estados iniciales fijos bajo las reglas de evolución. Una de las aportaciones de este enfoque es la clase IV de AC, a la que pertenece *vida*, ésta se caracteriza por tener patrones localizados de conducta compleja; las ciencias de la computación describen la clase IV como patrones cuya conducta rompe con la regularidad estadística de la distribución.

En la búsqueda del comportamiento de clase IV, principalmente se trabajaba mediante búsquedas exhaustivas, al hacerse esto imposible con el crecimiento del tamaño de los

patrones; se trabajó con otros enfoques, como la unión de patrones conocidos, colisiones de patrones, etc..

El obtener configuraciones complejas a partir de otras más simples arrojó resultados como los que se describen en el trabajo de Dave Buckingham, [11], sin embargo no hay un método acerca de cómo se debe realizar esta construcción, y la mayoría de ellas se hacen por ensayo y error.

El presente trabajo tiene como objetivo presentar un enfoque de la búsqueda de configuraciones hacia métodos usados por la teoría del corrimiento de registros *(shift register theory)*. Las ideas fundamentales de esta teoría se exponen en el trabajo de Golomb [20], Los diagramas de De Bruijn son una forma natural de modelar sucesiones de elementos donde existe un traslape, El artículo de McIntosh [7], dá un método que encuentra los patrones con un comportamiento dado, sin embargo se requiere una restricción importante, pues se trabaja en porciones finitas del espacio base.

Existen muchos problemas que se siguen trabajando en AC, uno de ellos es la búsqueda de patrones que no pueden ser producidos por las reglas evolutivas del AC y sólo pueden ser configuraciones iniciales (estas configuraciones reciben el nombre de jardines del edén).

McIntosh hace un tratamiento de los autómatas celulares lineales [6], que se enfoca en los AC unidimensionales y presenta el algoritmo usado. En la búsqueda de patrones estables y periódicos en *vida*, existen dos trabajos posteriores del mismo autor ([8] y [9]), sin embargo, en la búsqueda de jardines del edén, aunque se presenta el diagrama de subconjuntos, no existe un trabajo aplicado a *vida*, por lo que se extenderá este trabajo

en la presentación del algoritmo con el fin de hacer una implementación posterior.

En el trabajo de José Negrete, [2] se presentan otros juegos ecológicos enfocados a la dinámica de poblaciones, y como tales, pocos de ellos hacen suficiente énfasis en la configuración espacial de los individuos y el principal interés es en fluctuaciones de población. Como una extensión de ello, se presentan dos juegos ecológicos inspirados en el juego de la vida, en los cuales se observan comportamientos que pudieran presentarse en modelos mas completos posteriores. La aplicación de los algoritmos de detección de patrones presentados se podrá realizar en estos juegos sin extrema dificultad, una vez que se haya visto el caso de *vida*, aunque el crecimiento de la complejidad computacional limitará el análisis.

Se implementará asimismo un sistema de simulación de propósito general para AC bidimensionales, con el cual se podrán llevar a cabo las simulaciones presentadas y se pretende que sirva como base para una posterior investigación en aplicaciones de AC.

En el capítulo 2 se hace una aproximación informal a los conceptos fundamentales de los AC, con el fin de poder formular una definición mas precisa, en la segunda parte de la sección, se formalizan los conceptos tratados mediante el autómata elemental. Se elaboran las definiciones del espacio base del autómata, la célula que lo compone, el estado que dicha célula tenga y la configuración que componen los estados de todas las células en un instante de tiempo dado.

En el mismo capítulo se tratarán brevemente las propiedades más importantes de los AC, que son su conducta a largo plazo, expuesta mediante la clasificación general de

Wolfram y la reversibilidad, que es un tema completo de estudio por sí mismo, por lo que se mencionan los resultados más importantes.

El capítulo 3 desglosa los distintos algoritmos para detección de patrones con comportamiento especial :

El estudio de las gráficas asociadas a AC cuando se restringe el espacio base a un toro, que permite detectar los ciclos límte, jardines del edén, patrones estables, y muchos otros comportamientos, pero no existe una manera inmediata de generalizar los patrones a AC no restringidos. Los diagramas de De Bruijn de distintos niveles dan un método para construir patrones en "tiras" de ancho fijo y longitud arbitraria; algunas de las configuraciones generadas son extendibles a AC no restringidos.

El capítulo 4 trata sobre las particularidades del juego de la vida J. H. Conway. uno de los AC más conocidos y trabajados, se describen de manera breve los patrones con comportamiento característico como un ejemplo de la clase de resultados que se desearía obtener mediante algoritmos..

En el capítulo 5 se trata la aplicación de los algoritmos de detección mencionados en el capítulo 3 al juego de la vida, con énfasis en los diagramas de De Bruijn. se establecen las particularidades de una implementación, mencionando los resultados obtenidos en casos de complejidad pequeña.

Capítulo 2

Autómatas Celulares.

Al elaborar modelos tradicionales, principalmente para fenómenos biológicos y físicos, normalmente se encuentra el problema de obtener ecuaciones que no pueden ser resueltas analíticamente. entonces se trata de aproximar numéricamente las soluciones y en muchos de los casos se toma como solución una aproximacion que arroja características cualitativas. En casos como estos, la simulación directa puede proveer más datos sobre fenómenos con un margen de error igual o menor que otros modelos que involucran demasiadas variables o implican la falta de una solución explícita (como en los sistemas de ecuaciones diferenciales acoplados).

Aunque los AC son modelos discretos, en muchos casos, este tipo de simulación aproxima lo suficiente las soluciones de los sistemas con variables continuas, un ejemplo es el de la simulación de gases o reacciones químicas.

Un autómata celular se piensa como un modelo que extrae características original-

mente observadas en seres vivos (tejidos, microorganismos, colonias, etc.) como son el estar constituídos de elementos mínimos idénticos (células), que influyen entre sí, cambiando de comportamiento (estado), dichas células evolucionan en el tiempo de manera determinística.

Para poder elaborar una definición precisa, es conveniente exponer brevemente el modelo mas sencillo de autómata celular, el autómata unidimensional más simple:

Pensemos en una sucesión (una "fila" infinita en ambas direcciones) de "células", elementos que pueden estar en uno de dos estados ("vivo y muerto", "apagado y encendido", "0 y 1", etc.) una porción de ella se vería como en la figura 2-1:

Figura 2-1: Segmento de un AC bidimensional binario.

En el siguiente instante de tiempo, el autómata evolucionará un paso a la vez: el estado de cada célula cambia con respecto al estado de la misma en el tiempo anterior, y además depende del estado de sus dos vecinas mas cercanas (la *vecindad* de una célula) de acuerdo a una regla determinada (figura 2-2).

Un autómata celular es un sistema dinámico discreto: El espacio, el tiempo y los estados posibles de cada célula son conjuntos discretos, el estado de todas las células en un tiempo dado es una *configuración.*

En este caso, la regla que determina la evolución del autómata depende, -en cada paso del tiempo-, del estado presente de la célula y del estado de sus dos vecinas inmediatas;

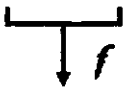


Figura 2-2: Regla local de evolución

estos individuos contiguos conforman la *vecindad* de una célula. En el caso de este ejemplo, el hecho que la vecindad sea de una célula a cada lado se denota como $\{-1, 1\}$. Se define el *radio de conectividad* como la mayor longitud de los elementos de la vecindad.

La regla o función local se determina dando el estado que tomará la célula central con respecto a la vecindad. una de las 256 reglas posibles para este autómata sería:

$$f(0,0,0) = 0$$

$$f(0,0,1) = 1$$

$$f(0,1,0) = 0$$

$$f(0,1,1) = 1$$

$$f(1,0,0) = 0$$

$$f(1,0,1) = 1$$

$$f(1,1,0) = 0$$

$$f(1,1,1) = 1$$

La nomenclatura para las reglas locales del autómata unidimensional binario es in-

mediata: se toma el numero binario formado por la última columna de la tabla anterior, tomando como dígito menos significativo el primer renglón de la tabla anterior. La regla queda únicamente caracterizada por el decimal correspondiente. La regla anterior será en esta notación, la regla 10101010 en binario o 170 decimal , tambien escrito como f_{170}.

Se puede ver en la figura 2-3 que el efecto de esta regla es un corrimiento hacia la izquierda de cada configuración.

Una *regla totalista* es aquella que sólo depende de la cantidad de células que se encuentran en determinado estado dentro de la vecindad, y no de la disposición de las células.

Podemos ver el comportamiento de un AC de una manera global, teniendo una función que asigne a cada configuración, en un tiempo dado, la configuración resultante del autómata según la regla local en cada célula (Figura 2-4):

O más brevemente si s_t es la configuración del autómata en el tiempo t:

$$s_{t+1} = F(s_t)$$

Aunque esta función es más difícil de especificar, puesto que el número posible de configuraciones es infinita (numerable). sólo depende de la definición de nuestra regla local, que se especifica mediante una tabla finita.

2.1 Definiciones.

Dicho lo anterior, se puede elaborar una definición para el autómata celular de dimensión d, vecindad N y cuyas células pueden tomar cualquier estado de un conjunto S:

Sea un conjunto finito S .

El *espacio base* es el conjunto de funciones de S en $\mathbf{Z}^d$ (es decir, la lattice $\mathbf{S}^{Z^d}$).Cada elemento del conjunto S es un *estado* del Autómata celular.

Una *configuración* del autómata es un elemento del espacio base $\mathbf{S}^{Z^d}$.

La *regla evolutiva global* es una función:

$$F : \mathbf{S}^{Z^d} \rightarrow \mathbf{S}^{Z^d}$$

La regla evolutiva global F determina el comportamiento del sistema , si s_t es la configuración del autómata en el tiempo t:

$$s_{t+1} = F(s_t)$$

Una condición para F es estar "determinada localmente" por una función (llamada *regla local* o de vecindad) f . Es decir, que debe existir una función

$$f : \mathrm{S}^{2r+1} \rightarrow \mathrm{S}$$

Definida en una región finita $N \subset \mathrm{S}^{Z^d}$ (que se define como la *vecindad*), de tal manera que el valor de F depende de la aplicación de f en dicha región.

Los valores de F están determinados por f, si la restricción de F a cada célula c es:

$$F_f(c)_v = f(c_{v+N})$$

Donde $v \in \mathbf{Z}^d$ y $v + N$ es el conjunto de todas las traslaciones de v por elementos de N.

Una propiedad fundamental en el estudio de los AC es que, aunque el espacio y la función F sean conjuntos infinitos, la definición de f es una tabla finita.

Un AC está determinado entonces por la tetrada (S, d, f, N). y se define como *AC inyectivo* si F es inyectiva y *AC suprayectivo* si F es suprayectiva.

2.2 Propiedades.

2.2.1 La clasificación de Wolfram.

Los AC fueron propuestos como modelos simples en los cuales estudiar procesos biológicos como la auto-reproduccción . Cualquier sistema de variables discretas bajo interacciones locales determinísticas puede ser aproximado con un AC. Los AC pueden ser considerados como procesadores paralelos, en los que las condiciones iniciales determinan el programa y datos de entrada, el estado al que se converja será la salida codificada.

Un AC suficientemente complejo (como *vida*) es "computacionalmente universal" [3], es decir, se convierte en una computadora de propósito general. Es posible codificar cualquier algoritmo o sistema formal de primer orden en él, comportándose como una máquina de Turing universal[1].

Al realizar un cierto trabajo de experimentación, y observar una gran cantidad de resultados de simulaciones, se busca un esquema de clasificación de la conducta a corto plazo del AC en cuestión. Se buscan reglas que exhiban una conducta regular; haciendo variar la definición de la función local y observar, dada una configuración, su evolución posterior.

Si partimos de un estado desordenado podemos encontrar conductas clasificables, presentándose cuatro grandes tipos de comportamiento:

- Clase 1: Patrones que evolucionan a estados homogéneos
- Clase 2: Evolucionan a estructuras períodicas separadas

- Clase 3: Exhiben patrones caóticos.

- Clase 4: Presentan comportamiento caótico en pequeñas zonas localizadas.

La clasificación es a todas luces cualitativa; podemos ver ejemplos de estos comportamientos en la figura 2-5. Sobre todo en lo referente a la clase 4 no hay un parámetro que nos permita decidir si una configuración pertenece o no a esta clase .Se trabaja en formas de hacer la clasificación cuantitativa, y formular definiciones precisas para las clases.

Para algunas reglas, uno esperaría que la pertencencia a cada clase fuera fácilmente determinable, pero existen casos en la frontera donde la definición no es clara.

Los comportamientos observados en las simulaciones de Autómatas Celulares tienen su contraparte en sistemas dinámicos contínuos: la clase 1 muestra puntos límite, mientras la clase 2 puede ser considerada como evolución hacia un ciclo límite.

La clase 3 de AC exhibe una conducta parecida a la encontrada en sistemas con atractores extraños. La clase 4 de AC no tiene un equivalente en sistemas dinámicos tradicionales (lo que se da en llamar comportamiento parecido a un solitón). La teoría de los sistemas dinámicos discretos nos da una aproximación preeliminar a la caracterización de los AC, así también se utilizan herramientas derivadas de la mecánica estadística [15].

Con el fin de medir la transición entre clases, se han definido variables asociadas a un AC. Cada una de ellas cuenta el número de sucesiones posibles de configuraciones correspondiente a alguna región del espacio base. Por ejemplo, la entropía espacial mide la dimensión del conjunto de configuraciones que pueden ser generadas en un tiempo

finito de evolución, empezando de todos los posibles estados iniciales. Hay, en general $N(X) \leq |S|^X$ sucesiones posibles de valores para un bloque de X sitios en este espacio de configuraciones. Otra magnitud con la que se observan fases de transición es la entropía de topología espacial, que se define como $\lim_{X\to\infty}(1/X)\log_K N(X)$, etc.

Sin embargo no es aún claro como se comportan las transiciones entre las clases con respecto a estas variables, se ha afirmado de la clase IV que el surgimiento de autómatas computacionalmente universales rompe la regularidad estadística que mide la entropía [15].

2.2.2 Reversibilidad.

Un Autómata Celular es reversible cuando su función global F es biyectiva y su inversa F^{-1} es en sí misma, una función global para algún otro autómata y por tanto tiene una descripción local. Cuando un AC es reversible, cada configuración tiene el menos un ancestro y en este caso su evolución e historia de una configuración dada pueden ser determinadas sin ambigüedades.

Proposición 1: Un AC es suprayectivo si, y sólo si es inyectivo.

La demostración es un trabajo de Moore, y se puede encontrar en :[12]

Proposición 2: Toda función suprayectiva que conmute con una traslación y que sea contínua para la topología producto es una función global para algún AC.

La demostración de Hedlund se encuentra en [3]

Como consecuencia inmediata , tenemos que es suficiente ser inyectivo para ser reversible, por desgracia, existen cuestiones de decidibilidad para esto:

Proposición 3: Es decidible la reversibilidad del autómata unidimensional (d = 1).

(Demostrado por Amoroso en [14])

Proposición 4: Es indecidible la reversibilidad del autómata celular de dimensión 2 o mayor.

(La demostración de Kari se encuentra en [3])

El resultado anterior es uno de los mas importantes, y para demostrarlo se utiliza la indecidibilidad demostrada de otro problema y se prueba la equivalencia; sin embargo. sólo el bosquejo con la idea básica es un trabajo completo. Dada su importancia, se espera que pronto se encuentre una prueba más sintética.

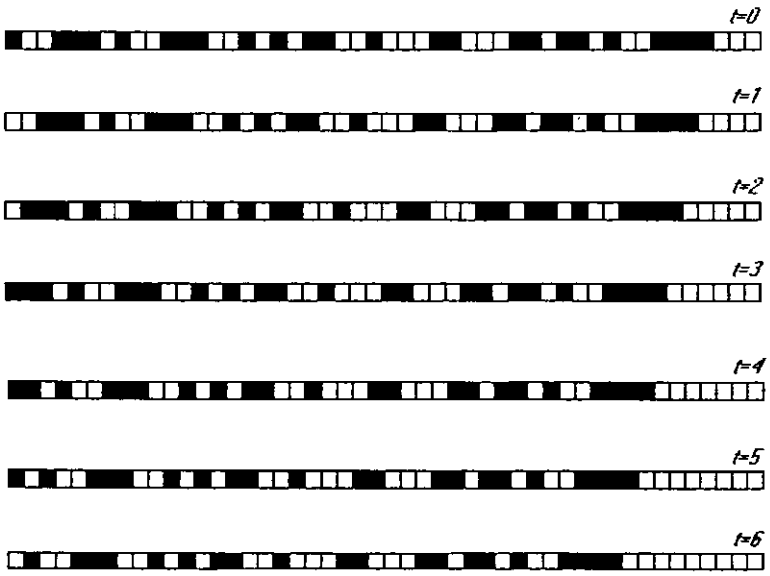


Figura 2-3: 6 generaciones de la regla 170 en el autómata elemental, mostrando un corrimiento hacia la izquierda

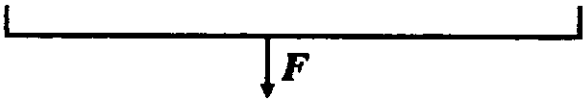


Figura 2-4: Regla evolutiva global

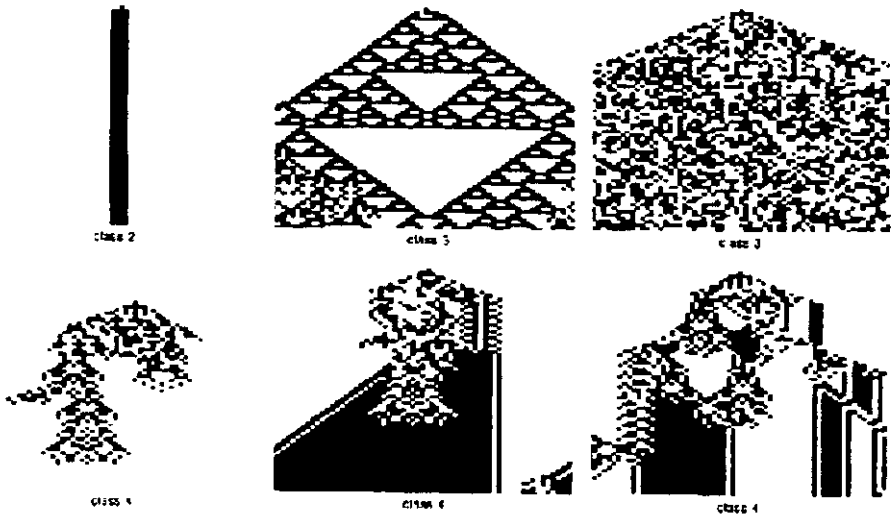


Figura 2-5: Diagramas de tiempo para el autómata elemental, donde se muestran comportamientos que ilustran las clases de Wolfram, tomado de [16]

Capítulo 3

Detección de patrones en Autómatas Celulares.

3.1 Diagrama de transiciones.

Cuando se restringe el espacio del AC, es decir, se toma como espacio base una latiz cúbica finita de dimensión d (indexada por los enteros módulo n), el espacio base se reduce al conjunto $S^{Z_n^d}$, asimismo, se puede restingir a una latiz $d-$rectangular , con lo que el espacio quedará como $S^{(Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_d})}$, a dicho conjunto se le llama *anillo* (sin relación con sus propiedades algebráicas). Puesto que se mantiene la homogeneidad de las vecindades el espacio resultante es equivalente a una latiz sobre un d-toro (ver figuras 3-1 y 3-2):

Un AC restringido a tal espacio es analizable. Tanto el número posible de configura-

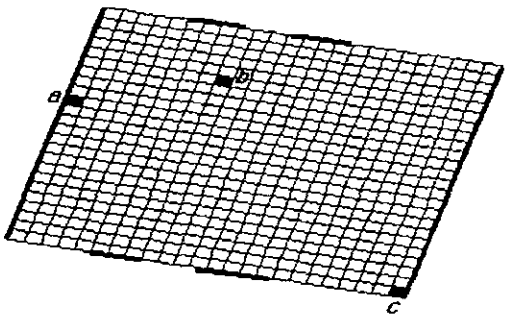


Figura 3-1: Ejemplo en dos dimensiones de un espacio base restringido, a, b y c son tres células con sus vecindades sombreadas (el AC tiene vecindad de Moore en este caso)

ciones como la función global F se convierten en conjuntos finitos; lo que hace posible la determinación de configuraciones cíclicas, jardines del edén, configuraciones estables, etc.

Para cada AC restringido, se puede construir una gráfica dirigida que representa su evolución, llamada diagrama de transiciones. Cada nodo de dicha gráfica corresponde a un estado del AC. Los nodos se unen con un arco del nodo i al nodo j si el estado correspondiente al nodo i evoluciona al estado del nodo j bajo la función global (ejemplos de estos diagramas en anillos del AC elemental son los 3-3, 3-4 y 3-5).

Una configuración estable será la que corresponda a un nodo que sólo se una con él mismo, los jardines del edén serán los nodos que no tengan arcos que lleguen a ellos.

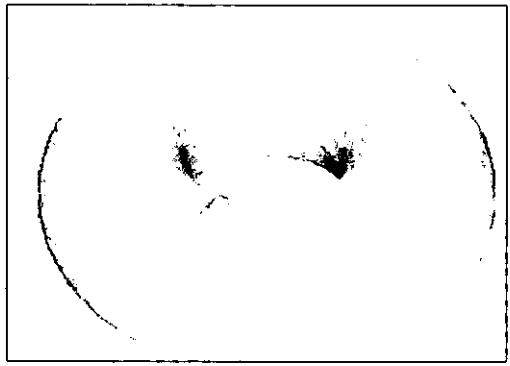


Figura 3-2: En dos dimensiones, al identificar los bordes opuestos del espacio base restringido, el espacio base es equivalente a la latiz sobre el toro

Puesto que el número de transiciones posibles es finito, eventualmente, deberá existir algún patrón que se repita, por lo que en AC restringidos siempre habrá por lo menos un ciclo límite.

Una vez que se ha construído el diagrama de transiciones, se puede representar mediante la matriz M, que se define:

$$m_{ij} = \begin{cases} 1 \text{ si } F(s_i) = s_j \\ 0 \text{ en otro caso} \end{cases}$$

dicha matriz es denominada *matriz de evolución.*

Donde s_i y s_j son los estados del AC asociados a los nodos i, j. Para algunas reglas en particular, esta matriz puede ser reducida a clases de equivalencia, por ejemplo, si f (la regla local) es invariante bajo translaciones, los estados se agruparán en clases de

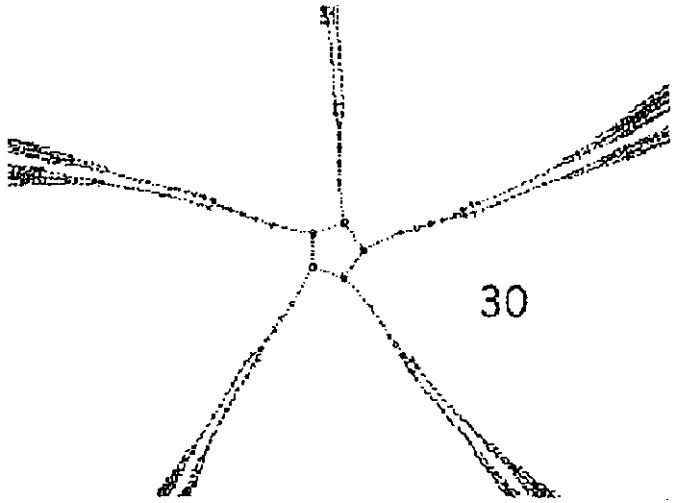


Figura 3-3: Diagrama de transición para la regla 30, anillo de long. 20

simetría translacional.

Puesto que las sucesiones de estados generados admiten una representación con gráficas finitas, se pueden asociar a una expresión regular, y por tanto, hay un autómata finito que acepta tal lenguaje [23], con ello se puede ver que, para un valor fijo finito del tamaño n del anillo, el AC es equivalente a un Autómata o máquina de estados finitos, lo que es computacionalmente inferior a una máquina de Turing.

Sin embargo, conforme aumenta n la complejidad del diagrama de transiciones también aumenta, y a consecuencia, el conjunto límite, - también equivalente a un AC -, no es un lenguaje regular. Se piensa que una clase de lenguajes más amplia, los lenguajes libres de contexto (LLC) podrían proveer descripciones de los conjuntos límite, sin embargo no

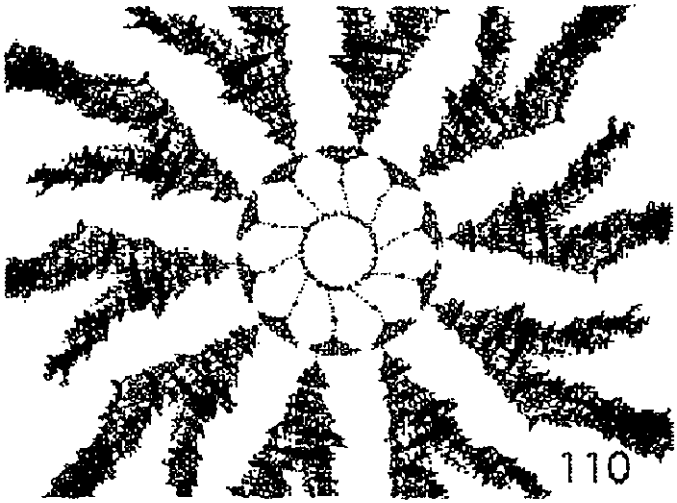


Figura 3-4: Diagrama de transiciones para la regla 110 en un anillo de longitud 20

se conoce ningún ejemplo no trivial de un AC descrito por un LLC [17].

La inspección exhaustiva del diagrama de transiciones es un problema de complejidad no polinomial (NP), puesto que el número total de nodos en la gráfica es $|S|^{n^d}$,(es decir, el número de estados posibles en un anillo de tamaño n en d dimensiones) lo que la hace imposible para valores de n moderadamente grandes.

3.2 Diagramas de De Bruijn

Considerando el problema desde otro punto de vista, una de las dificultades para poder predecir la conducta de un AC es el hecho que las vecindades se traslapan, lo que hace difícil la obtención de patrones por construcción. Una aproximación sería: dada una

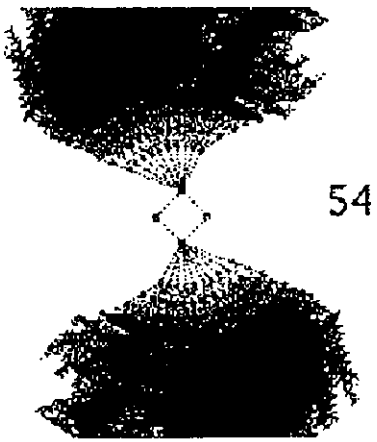


Figura 3-5: Diagrama de transiciones para la regla 54

célula inicial, fijar su vecindad de modo que se comportara de la forma deseada; después, repetir el mismo proceso con la unión de las vecindades de cada célula de la vecindad del paso anterior, y así sucesivamente. Si se alcanzan estados inconsistentes; se regresaría a un paso anterior. El algoritmo mencionado, aunque eficaz para construir, por ejemplo, patrones estables, tiene una complejidad enorme; ya que en cada paso el número de celdas a considerar aumenta como $2^{|S|^n n^2}$.

La teoría del corrimiento de registros provee un modelo para vecindades traslapadas, considerando estas como parte de un patrón mayor, del que existe una "ventana" que sólo deja ver una parte de la configuración a la vez.

La representación gráfica de una serie de cadenas traslapadas es el diagrama de De

Bruijn. La aplicación para los AC se hace tomando una gráfica dirigida. Asociando un nodo a cada vecindad N, y cada arco al resultado de un paso de la evolución en dicha vecindad $F(N)$. Incluyendo o excluyendo arcos del diagrama de acuerdo a propiedades que nos interese conservar bajo F nos resulta una cantidad de subdiagramas de los que se pueden derivar métodos para la construcción de configuraciones de diverso comportamiento.

Para construir el diagrama de De Bruijn, tomemos una gráfica dirigida con k^s nodos, cada nodo representa una posible palabra de longitud s de un alfabeto de k símbolos. Si se piensa en los símbolos como enteros módulo k, los nodos son los enteros de s dígitos escritos en base k.

Un arco va de un nodo a otro si el primero termina con los mismos $s-1$ dígitos con que el segundo empieza; el diagrama resultante describe cómo se traslapan las distintas vecindades.

Aunque distintos grados de traslape pueden ser considerados, el más simple es el que se da con la adición o la eliminación de un símbolo al final de la sucesión: por ejemplo, la palabra binaria 1001 se traslapa con 0010 perdiendo el primer símbolo, mientras la segunda se traslapa perdiendo el último símbolo. El arco que une los nodos correspondientes se etiqueta con la sucesión completa 10010.

Los símbolos se toman como números enteros, entonces es posible expresar las conexiones en forma algebraica o aritmética. Debido a que el número de nodos es grande aun para anillos muy pequeños, es conveniente representar los diagramas con sus matrices de

adyacencia.

La notación usada para la matriz de conectividad de un diagrama de k símbolos y nivel n es $B_{k,n}$. Si b_{ij} es el elemento del renglón i y la columna j, entonces $B_{k,n}$se puede escribir de forma aritmética como:

$$b_{ij} = \begin{cases} 1 \text{ si } j = \begin{cases} ki \\ ki+1 \\ \cdots \\ ki+k-1 \end{cases} \pmod{k^{2r}} \\ = 0 \text{ en otro caso} \end{cases}$$

El diagrama de la figura 3-6 corresponde a la matriz:

\	00	01	10	11
00	1	1	0	0
01	0	0	1	1
10	1	1	0	0
11	0	0	1	1

De acuerdo a los distinos traslapes posibles, se puede evaluar cualquier función booleana b de los nodos. Eliminando los arcos que no la preservan, podemos construir para cualquier recorrido, una configuración que preserve b bajo F (la función evolutiva), ésto

es particularmente útil para encontrar patrones estables.

Existen varias maneras de construir el diagrama de De Bruijn asociado a un autómata (S, d, f, N), el principal problema consiste en encontrar subvecindades cuyos traslapes se puedan representar de manera sencilla, en el caso de $d = 1$ o $d = 2$ (con vecindad de Moore), ésto se puede hacer de manera directa.

Los arcos de un diagrama de De Bruijn se asocian con las vecindades de manera natural si se usan los mismos símbolos para los estados, entonces el diagrama estará naturalmente asociado a los AC $(\{0,1\}, 1, f, \{-1, 1\})$ para toda regla f. Cada nodo representa la vecindad parcial de una célula, mientras el arco representa la vecindad completa resultante del traslape de las parciales.

Para una regla dada, se pueden obtener todos los patrones estables removiendo aquellos arcos en la gráfica que no mantengan el estado de su célula central, así para la regla 90 (adición binaria):

vecindad	000	001	010	011	100	101	110	111
imágen	0	1	0	1	1	0	1	0
b = "preserva célula central"	sí	no	no	sí	no	sí	sí	no

Con el criterio anterior, vemos que se remueven los arcos 001,010,100 y 111, resultando el diagrama 3-7:

Entonces, en el AC $(\{0,1\}, 1, f_{90}, \{-1, 1\})$ la sucesión de recorridos del subdiagrama

será el conjunto de patrones estables, así $(011)^*$ serán patrones estables.

En la detección de los Jardines del Edén, el cálculo es mucho más laborioso, puesto que se utiliza una construcción distinta: a cada arco del diagrama de De Bruijn se le reetiqueta con el estado al que evolucionó la vecindad, -que correspondío al nombre anterior del arco -.

Una vez obtenido el diagrama 3-8. Cada recorrido de la gráfica genera una sucesión de células, con la particularidad de haber evolucionado de alguna configuración; para encontrar la sucesión, se usa la unión de las vecindades que describen los nodos de la gráfica.

En el diagrama anterior (3-8) generemos el recorrido $(00, 01, 11, 11, 10, 00)$; tomando los estados asociados a los nodos tenemos la secuencia 11011 (etiquetas inferiores). Apreciamos que la configuración 11011 procede de la configuración 0011100 aplicando la regla de evolución. Pero esta última secuencia (la configuración padre) se obtuvo escribiendo el traslape de los nodos del mismo recorrido (etiquetas superiores). Obviamente, otro recorrido distinto que produzca la misma secuencia dará un ancestro distinto. De esta manera se pueden detectar todos los ancestros de una configuración dada. Podría pasar también que no hubiera un recorrido en el diagrama que corresponda a ésta. En ese caso se trataría de un Jardín del Edén [9].

3.3 Diagrama de subconjuntos

Podríamos ser más sistemáticos, buscando en lugar de un recorrido en particular, todos los recorridos que evolucionan a una configuración dada. Tomamos la primera célula de nuestra configuración, anotamos todos los estados que producen ésa célula, desde cada uno de dichos estados, buscamos el conjunto de nodos que produzcan la segunda célula, y así sucesivamente. Entonces estaremos obteniendo recorrido en una gráfica donde cada nodo corresponde a un subconjunto de nodos del diagrama original.

En esta nueva gráfica, al llegar al nodo correspondiente al conjunto vacío encontraremos una palabra excluída, ninguna configuración que contenga ésta puede ser producida por evolución [9].

Construyamos el diagrama de subconjuntos asociado a la regla 90:

Por simplicidad etiquetamos los nodos:

00	A
01	B
10	C
11	D

Entonces podemos construir el recorrido en cada caso, cada nodo tendrá dos arcos de salida :

Nodo	0 lleva a	1 lleva a
A	A	B
B	C	D
C	B	A
D	D	C

El diagrama de subconjuntos completo corresponderá a tomar como nodo cada elemento de $\wp(\{A, B, C, D\})$ y según la tabla anterior, construir cada uno de los arcos entre los nodos. Así el nodo $\{ABC\}$ estará unido al nodo $\{BDA\}$ por el arco 1.

En este caso no se encontrarán palabras excluídas, pues cada subconjunto de nodos va a exactamente dos subconjuntos (tomando la unión). Cada nodo tiene dos arcos que salen de él etiquetados con 0 y 1.

En otras reglas, este equilibrio se rompe, habiendo nodos con dos arcos 1 o dos arcos 0. Para encontrar un ejemplo de regla no balanceada, tomemos la regla 126 (todos los estados van a 1 excepto 000 y 111):

vecindad	000	001	010	011	100	101	110	111
imágen	0	1	1	1	1	1	1	0

Nombrando los nodos del diagrama de De Bruijn de la misma manera que el anterior,

construimos la tabla para los nodos individuales:

Nodo	0 lleva a	1 lleva a
A	A	B
B	ϕ	C, D
C	ϕ	A, B
D	D	C

El diagrama de subconjuntos completo son 2^4 nodos para los 16 conjuntos, pero para la detección de los jardines del edén nos interesa únicamente la parte conexa del nodo universal $\{ABCD\}$ pues, al ir construyendo una palabra, el recorrido nos irá generando los ancestros de dicha palabra y si se encuentra el conjunto vacío quiere decir que se trata de una palabra excluída.

De esta manera construímos la parte que nos interesa del diagrama de subconjuntos:

Subconjunto	0 lleva a	1 lleva a
$\{ABCD\}$	$\{AD\}$	$\{ABCD\}$
$\{AD\}$	$\{AD\}$	$\{BC\}$
$\{BC\}$	ϕ	$\{ABCD\}$
ϕ	ϕ	ϕ

Vemos que la palabra 010 nos da el recorrido $\{ABCD\}$,$\{AD\}$,$\{BD\}$,$\{\}$. por lo que es una palabra excluída, y toda configuración que la contenga será un jardín del edén.

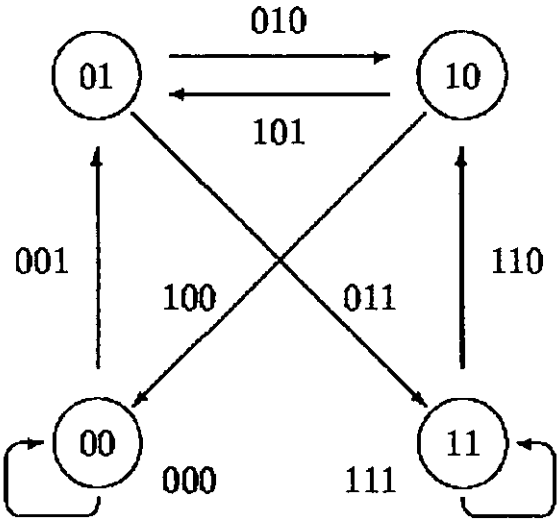


Figura 3-6: Diagrama de De Bruijn genérico para vecindad {-1,1}

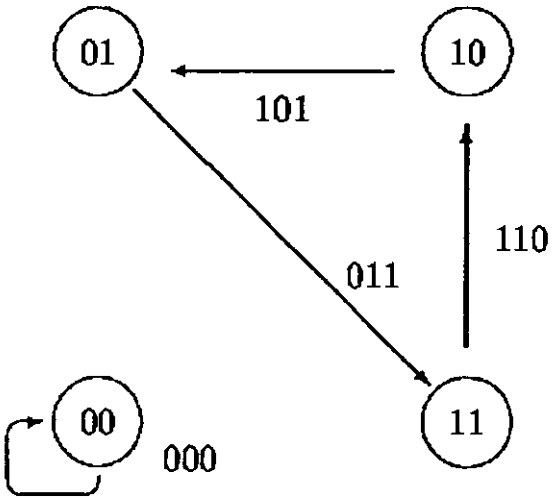


Figura 3-7: Diagrama de De Bruijn correspondiente a la regla 90

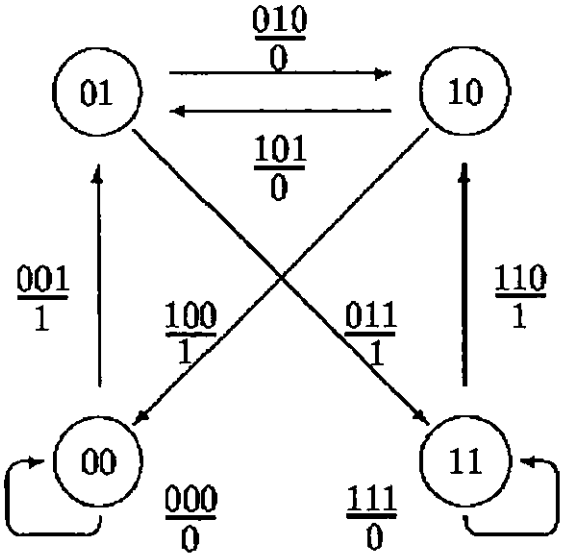


Figura 3-8: Diagrama con los estados asociados a la regla 90

Capítulo 4

El Juego de la vida (J. H. Conway).

4.1 Generalidades

Uno de los AC más conocidos y difundidos es el "juego de la vida" presentado en 1970 por J. H. Conway; el cual se puede especificar de la siguiente manera:

$$S = \{0, 1\}$$

$$d = 2$$

$$N = \{(0,0), (-1,1), (-1,0), (-1,-1), (0,-1), (1,-1), (1,0), (1,1), (0,1)\}$$

(vecindad de Moore)

$$f(c+N) = \begin{cases} 1 \text{ si } \begin{cases} c = 1 \text{ y } 2 \leq \sum_{i=1}^{8} c + n_i \leq 3 \\ c = 0 \text{ y } \sum_{i=1}^{8} c + n_i = 3 \end{cases} \\ 0 \text{ en cualquier otro caso} \end{cases}$$

La regla se lee en otros términos como:

- Un individuo que tenga menos de 2 vecinos, morirá en la siguiente generación.
- Un individuo con más de 3 vecinos, muere por sobrepoblación.
- En los espacios vacíos que tengan 3 vecinos, nacerá un nuevo individuo.

Vida es un AC bidimensional binario con vecindad de Moore,esto es, las vecindades se construyen con la célula central y sus 8 vecinas inmediatas.

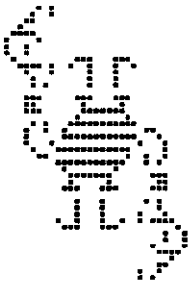
La función evolutiva local es una *regla semitotalista*, es decir, se puede escribir como la composición de funciones entre una regla totalista y una regla que depende sólo del estado de la célula central.

Este autómata es notable por el hecho que una población inicial de células vivas generadas al azar eventualmente se estabiliza en una colección de objetos visiblemente

separados que pasan por cortos ciclos evolutivos. El ciclo más común, de período 1 es llamado un "estable", pero hay "osciladores" y "alternadores" usualmente de período 2. Después de una larga evolución, son comunes pequeños objetos residuales de larga vida, en contraste con otros AC que tienden a evolucionar hacia campos caóticos uniformes de densidad fija, o a separarse en zonas estables. Las reglas similares a las de *vida* no son comunes, es difícil encontrar otra regla que exhiba la conducta organizada encontrada en él.

Este AC se motiva en una analogía con organismos en una superficie plana (idealizada como una rejilla cuadrada infinita), cada individuo tiene ocho vecinos, y para la supervivencia de la colonia se siguen las reglas descritas anteriormente. A pesar de su simplicidad, *vida* ha exhibido uno de los comportamientos más complejos observados, y ha arrojado resultados sorprendentes, a partir de la observación de cómo evolucionan ciertos patrones, se han encontrado patrones estables, cíclicos, etc.

Conway había conjeturado que ningún patrón finito podría crecer indefinidamente, pero posteriormente se descubrió un patrón que produce una sucesión de configuraciones crecientes, el siguiente en particular, consta de una parte cíclica que al desplazarse en cada dirección, llenará el espacio con la proporción de 1/2 entre células vivas y muertas (*densidad*); también conjeturada como la densidad máxima posible.

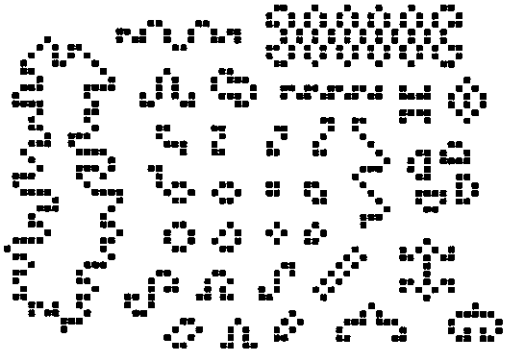


Patron de crecimiento infinito

4.2 Patrones.

4.2.1 Estables.

Los Patrones más comunes en el juego de la vida, son aquellos que se mantienen sin ningún cambio, (es decir $F(c) = c$) se conocen varios ejemplos:



Algunos patrones estables

algunos de ellos aparecen espontáneamente como resultado de la evolución de otros

patrones.

4.2.2 Osciladores.

Hay, asimismo, patrones que al cabo de n pasos, vuelven a su estado original y se conocen como patrones de período n o patrones pn. Los patrones estables son $p1$.

Un AC es omnioscilador si tiene osciladores de todos los períodos. no se sabe si vida es omnioscilador, pero es muy probable que lo sea, el trabajo de Dave Buckingham [11] en ciclos de herschels redujo el número de casos no resueltos a un número finito. Ahora, para los únicos períodos que no se conocen osciladores son 19, 23, 27, 31, 37, 38, 41, 43, 49 y 53. Aunque para algunos casos los osciladores conocidos no son muy satisfactorios; p. ej. para formar un $p34$ se usó un $p17$ y , de manera que no interfiriera, un $p2$. el patrón resultante se toma como uno solo, aunque se noten dos osciladores separados.

4.2.3 Naves.

Asimismo, un patrón que se observa de manera natural en muchas configuraciones es el trineo (glider), que se caracteriza por desplazarse recuperando su forma original, cada dos latidos, con la diferencia de la posición, un trineo tarda cuatro generaciones en avanzar una casilla en diagonal.

Su pequeño tamaño hace a los trineos (Figura 4-2) un intermediario frecuente en configuraciones durante su evolución, es tambien comprensible que las colisiones de trineos estuvieran entre los primeros procesos en ser estudiados. Se ha encontrado que colisiones

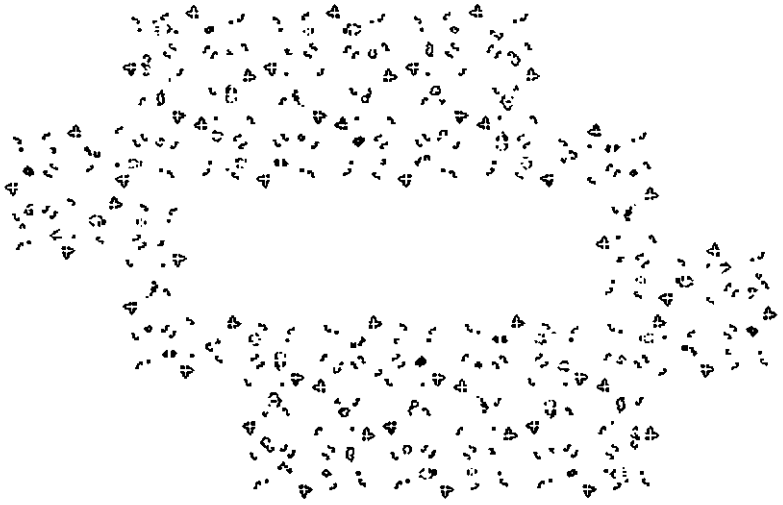


Figura 4-1: Oscilador de período 61 por Dave Buckingham

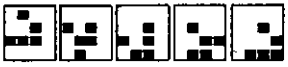


Figura 4-2: "trineo" una de las naves que aparecen com mayor frecuencia

de trineos de variada complejidad podrían producir todo tipo de objetos estables como un producto final.

La velocidad máxima a la que una nave puede desplazarse (sin perder su forma) es una casilla en cada paso, a esto se le llama c o la velocidad de la luz. Si la nave se desplaza d lugares cada que recupera su forma y completa este ciclo en n pasos, entonces se dice que es una nave de velocidad $\frac{cd}{n}$. Un trineo tiene velocidad $\frac{c}{4}$, y se han encontrado naves de velocidades variadas:

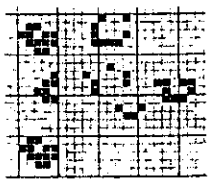


Figura 4-3: Nave de período 24, velocidad $c/2$

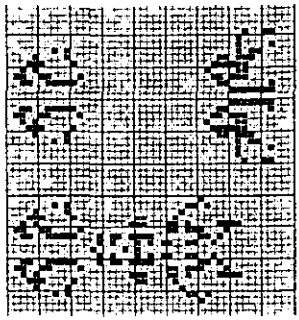


Figura 4-4: Naves de velocidad $2c/5$

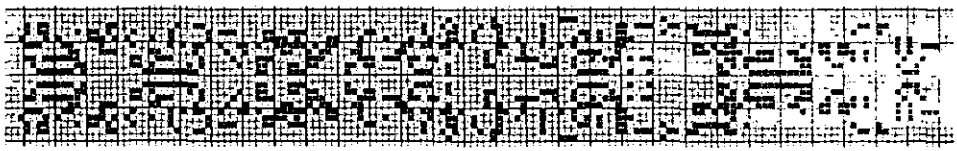


Figura 4-5: Nave de velocidad $c/5$

4.2.4 Jardínes del edén.

Un resultado, encontrado por Moore [12] establece que si en un AC hay una configuración con más de un ancestro, entonces habrá configuraciones sin ancestros, es decir, que no pueden ser generadas por evolución y solo pueden ser estados iniciales. La función global de vida no es inyectiva, esto se puede apreciar cuando se restringe el espacio a un cuadrado de $n \times n$, se han encontrado un par de patrones que no son producibles por las reglas locales a partir de ninguna configuración (fig 4-6 y 4-7):

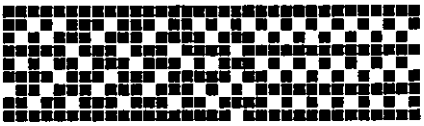


Figura 4-6: Jardin del edén

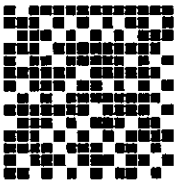


Figura 4-7: Jardín del edén, el más pequeño conocido en juego de la vida

Por lo general, probar que una configuración es un jardín del edén es un problema difícil, normalmente se hacen búsquedas exahustivas a través de todos los ancestros posibles. Otro problema relacionado y que sigue abierto es encontrar, si existe, un patrón que ten-

Capítulo 5

Análisis de un juego de la vida restringido a un espacio finito (toro).

Como se vió anteriormente, se pueden encontrar patrones de AC restringidos, dada la dificultad de búsqueda en el juego de la vida, se restringirá a espacios de $n \times m$ (anillos), en las cuales se pueden encontrar los patrones fijos, periódicos, naves y jardines del edén, sin embargo, el diagrama de transición de un juego de la vida restringido no necesariamente es una subgráfica del diagrama de transición de vida (de hécho el diagrama de transición de vida no es una gráfica, puesto que el número de nodos es infinto).

La existencia de un patrón con un comportamiento en el espacio restringido $S^{(Z_n \times Z_m)}$ no implica necesariamente que lo tenga en el espacio base S^{Z^2}, sin embargo, bajo ciertas condiciones algunos patrones pueden ser transportados al espacio base, por ejemplo, la existencia de un borde quiescente en el anillo garantiza que un patrón estable en el espacio

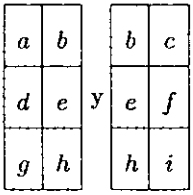
restringido, sea estable en cualquier anillo que lo contenga, para los patrones periódicos, el borde debe mantenerse para todo el ciclo. En el caso de los jardines del edén, las palabras excluídas del AC no restringido estarán contenidas en las de los anillos.

Para encontrar las configuraciones, se aplica el algoritmo descrito en [7] mediante diagramas de De Bruijn. La construcción del diagrama para dos dimensiones es un poco más complicada, puesto que en el ejemplo anterior se tenía una vecindad de 3 células en una dimensión. De esta manera al nodo 00 se le descartaba el primer símbolo, agregando un 1 al final, lo que daba el nodo 01, el arco, entonces tenía la vecindad completa 001.

En el juego de la vida, se tienen vecindades de 3×3 en dos dimensiones, pero asimismo nos interesa el traslape de las vecindades parciales; es decir, la vecindad representada por la matriz:

a	b	c
d	e	f
g	h	i

correspondiente a la célula e, será el resultado del traslape de las vecindades:



Este traslape se puede dar de ocho modos; hay $2^9 = 512$ posibles estados para cada vecindad, y cada nodo está contituído por una "media vecindad" de 6 células, por lo que el primer nivel del diagrama de De Bruijn tendrá 64 nodos y 512 arcos. Esta construcción nos arrojará en anillos patrones de altura 3 y longitud arbitraria.

Para construir anillos de mayor altura se requiere hacer niveles mayores del diagrama, tarea que requiere demasiado tiempo de cómputo. sin embargo, algunos de los patrones generados tienen bordes quiescentes o son periódicos, por lo que pueden ser repetidos arbitrariamente a lo alto.

El primer nivel del diagrama de De Bruijn muestra qué renglones de células pueden ser construídos de tal manera que cumplan el comportamiento deseado. La manera como se unen estos renglones está determinada por un segundo nivel del diagrama: se escoge una anchura fija pero arbitraria que dará todos los patrones de dicha longitud. Se puede atenuar el crecimiento del número de nodos tomando en cuenta sólo la parte conexa del nodo $(0,0)$ de la matriz de conectividad. El diagrama completo de una banda de longitud N deberá tener 2^{2N} nodos y 2^{3N} arcos. Por ejemplo, para una banda de ancho 5, dos renglones consecutivos se verán:

$$\begin{bmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \\ p & q & r & s & t \end{bmatrix}$$

Las células que forman el patrón a considerar (sobre el que se aplica la función para nombrar los arcos) serán los renglones $fghij$ y $klmno$, las vecindades parciales se verán:

célula	vecindad	vecindad superior	vecindad inferior
$fghij$	$a\ b\ c\ d\ e$ $f\ g\ h\ i\ j$ $k\ l\ m\ n\ o$	$a\ b\ c\ d\ e$ $f\ g\ h\ i\ j$	$f\ g\ h\ i\ j$ $k\ l\ m\ n\ o$
$klmno$	$f\ g\ h\ i\ j$ $k\ l\ m\ n\ o$ $p\ q\ r\ s\ t$	$f\ g\ h\ i\ j$ $k\ l\ m\ n\ o$	$k\ l\ m\ n\ o$ $p\ q\ r\ s\ t$

El nuevo diagrama de De Bruijn tendrá como nodos a las vecindades superiores e inferiores, y como arcos a las vecindades.

En el artículo citado [8] se aplicaron técnicas de supercómputo para mostrar el resultado del algoritmo para bandas de anchura de 1 hasta 6.

El diagrama de subconjuntos completo del diagrama consta de 2^{64} nodos, sin embargo la parte conexa del nodo total es la porción que nos interesa para poder encontrar palabras

excluídas,eso da una primera simplificación, el hecho que la búsqueda se haga hasta encontrar la primera palabra excluída podría reducir el número de nodos a cantidades manejable, pero como de todas maneras se debe empezar con rastrear el nodo total. El tiempo de respuesta, aunque finito, puede ser enorme.

5.1 Estimaciones sobre la complejidad del algoritmo de análisis

Un algoritmo eficiente es aquel cuyo "tiempo de respuesta", crece como un polinomio en función del "número de entradas", es decir, su complejidad es de orden polinomial. No se han encontrado algoritmos eficientes para detección de patrones en vida, puesto que todos los cálculos que hemos realizado muestran ordenes exponenciales, en general no polinomiales.

Aunque la complejidad de la búsqueda exhaustiva hace imposible en términos prácticos (encontrar los patrones en un cuadrado de n celdas es un problema de orden $o(2^{n^2})$ aunque se reduce mediante clases de simetría). Los diagramas de De Bruijn reducen la complejidad del algoritmo a la búsqueda de recorridos hamiltonianos en una gráfica, que aunque menor, es también de complejidad no polinomial (NP).

Un problema de complejidad NP, establece un límite a la cantidad de soluciones que se pueden encontrar con el método dado, puesto que las funciones exponenciales tienden a rebasar, con valores muy pequeños en el dominio a cualquier otra función creciente

polinomial, como lo es la evolución en la capacidad de proceso de las máquinas.

Al reducir los estados a clases de simetría, el número resultante de dichas clases es un resultado de la teoría de grupos, y en particular de g-conjuntos, hay

$$K = \frac{1}{o(G)} \sum_{g \in G} I(g)$$

clases de simetría. En esta expresión el conjunto g de operaciones de simetría sobre el grupo G de orden $o(G)$ tiene $I(g)$ puntos fijos. Cuando la simetría es rotacional los puntos fijos son sucesiones que se repiten antes de alcanzar la longitud del anillo; para reflexiones, son palindromas. El número de clases, de cualquier modo, crece exponencialmente, aunque para anillos largos, este número es menor que el número de configuraciones en una proporción de aproximadamente $2N$, el tamaño del grupo diédrico de rotaciones y reflexiones.

5.2 Resultados en algunos anillos.

El Diagrama de De Bruijn se puede especificar de manera más simple si usamos notación octal para etiquetar a los nodos, si tomamos una vecindad:

a	b	c
d	e	f
g	h	i

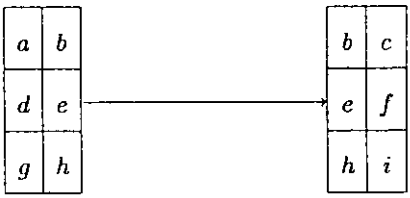
Y definimos

$$\alpha = 4a + 2d + g$$

$$\beta = 4b + 2e + h$$

$$\gamma = 4c + 2f + i$$

donde $a, b, c, d, e, f, g, h, i \in \{0, 1\}$, como en la gráfica tenemos los dos nodos correspondientes a las vecindades parciales traslapadas:



Podemos escribir esto como el arco $\alpha\beta\gamma$ y los nodos $\alpha\beta$ y $\beta\gamma$. Si usamos el operador $\mathrm{mod}(a, b)$ (función residuo de la división entera de a y b); la matriz de adyacencia de la gráfica se puede especificar (en notación decimal) como:

$$
M_{i,j} = \begin{cases} 1 \text{ si} \begin{cases} j = \mathrm{mod}(8 * i, 64) \\ j = \mathrm{mod}(8 * i + 1, 64) \\ j = \mathrm{mod}(8 * i + 2, 64) \\ j = \mathrm{mod}(8 * i + 3, 64) \\ j = \mathrm{mod}(8 * i + 4, 64) \\ j = \mathrm{mod}(8 * i + 5, 64) \\ j = \mathrm{mod}(8 * i + 6, 64) \\ j = \mathrm{mod}(8 * i + 7, 64) \end{cases} \\ 0 \text{ en otro caso} \end{cases}
$$

$$
i \in \{0, 1, ..., 63\}
$$

$$
j \in \{0, 1, ..., 63\}
$$

El arco que conecta al nodo i con el j está dado por:

$$
V(i, j) = 8(i - \mathrm{mod}(i, 8)) + j
$$

Por ejemplo, los nodos $10 = 12oct$ y $19 = 23oct$ estarán unidos por el arco $33 = 123oct$,pues $V(10, 19) = 33$. La matriz A de la vecindad de *vida* correspondiente al nodo $\alpha\beta\gamma$ está dado por

$$A_{i,j} = \mod\left(\frac{V(x,y)}{2^{3(2-j)+(2-i)}}, 2\right)$$
$$i,j \in \{0,1,2\}$$

El número de células vivas en dicha vecindad es:

$$N1(A) = \left(\sum_{i=0}^{2}\sum_{j=1}^{2} A_{i,j}\right) - A_{1,1}$$

si definimos

$$S(x,y,A) = \begin{cases} 1 \text{ si } A_{1,1} = f((x,y)+N) \\ 0 \text{ en otro caso} \end{cases}$$

donde f es la regla local de transición para *vida*, entonces la matriz de De Bruijn para patrones fijos es

$$B_{x,y} = \begin{cases} S(x,y,A) \text{ si } M_{x,y} = 1 \\ 0 \text{ en otro caso} \end{cases}$$

Ver anexos 1 y 2 para una vista de ambas matrices; cualquier recorrido en la matriz B

generará renglones de células que forman patrones estables, sin embargo, pocos de ellos pueden sobrevivir sin conectarse con otros renglones, o en configuraciones quiescentes al infinito, por lo que se tendrá que buscar aquellos patrones que pertenezcan a la parte conexa del nodo 0.

si (x, y, t) representa un patrón que cada célula corresponde a la de x lugares a la derecha y y lugares arriba después de t generaciones, la vecindad de 3×3 es lo suficientemente grande para detectar patrones $(0,0,0)$, $(0,1,1)$, $(1,0,1)$ y $(1,1,1)$, Para otro tipo de patrones se necesitarían diagramas de mayor orden, que involucran demasiados nodos.

Para la obtención del diagrama de subconjuntos con el fin de encontrar palabras excluídas, se requirió de un diagrama de ancho 14, que representa un cálculo demasiado complejo para los recursos computacionales que se disponen. Un posterior desarrollo a este trabajo corresponderá en aplicar técnicas de supercómputo para desarrollar los algoritmos aquí presentados.

Capítulo 6

Algunos juegos como modelos poblacionales.

Algunos fenómenos en modelos complejos pueden ser observados en juegos que se plantean como modelos, aunque un poco mas simples y desconectados de la simulación de procesos físicos, como tal se plantean algunos juegos que pueden mostrar comportamiento interesante.

Al tratarse de AC bidimensionales con vecindad de Moore, los algoritmos descritos anteriormente pueden aplicarse para encontrar patrones oscilantes, estables y jardines del edén. La diferencia en el número de estados posibles aumenta el número de nodos en la gráfica, con lo que el trabajo de cálculo será mayor.

6.1 Juego de las bestias.

6.1.1 Definiciones.

Este juego es una variante del juego de la vida en la que conviven dos especies, evolucionando cada una de manera independiente, pero en vecindades donde se encuentren ambas, la especie que esté en desventaja numérica morirá, este comportamiento se especifica mediante el AC:

$$\mathbf{S} = \{0,1,2\}\ N = \text{vecindad de Moore}$$

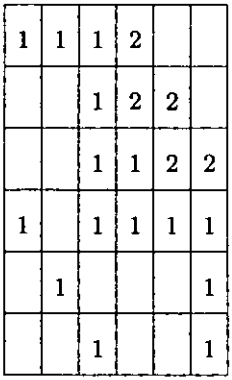
$$f(c+N) = \begin{cases} 1 \text{ si} \begin{cases} c = 1 \text{ y } 2 \leq N_1(c+N) \leq 3, N_2 = 0 \\ c = 0 \text{ y } N_1(c+N) = 3, N_2 = 0 \\ c = 2 \text{ y } {}_1(c+N) > N_2(c+N) \end{cases} \\ 2 \text{ si} \begin{cases} c = 2 \text{ y } 2 \leq N_2(c+N) \leq 3, N_1 = 0 \\ c = 0 \text{ y } N_2(c+N) = 3, N_1 = 0 \\ c = 1 \text{ y } N_2(c+N) > N_1(c+N) \end{cases} \\ 0 \text{ en cualquier otro caso} \end{cases}$$

Donde $N_1(c+N)$ es el número de células en estado 1 en $c+N$

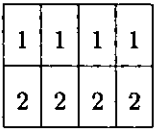
y $N_2(c+N)$ es el número de células en estado 2 en $c+N$

6.1.2 Algunos patrones.

En esta variación, se ven algunos patrones:



nave $p16$ con desplazamiento diagonal



Patrón que llena el espacio con densidad 1.

6.1.3 Simulación.

Al observar la evolución de configuraciones al azar en un anillo de longitud 50, se observan comportamientos descritos por Gutowitz [1] como "membranas" y "macrocélulas", existe una tendencia a formar zonas donde sólo existe una de las dos especies. La inestabilidad

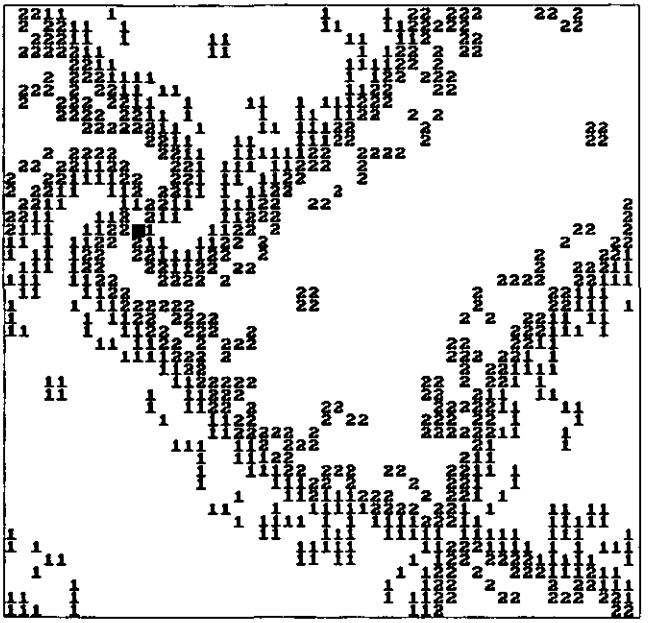


Figura 6-1: Patrón obtenido de la configuración simétrica 1

de las reglas de "vida" se observa en la evolución de los límites entre las zonas, en la mayoría de las configuraciones, las macrocélulas de una especie disminuyen de tamaño hasta que se extinguen. Otras configuraciones muestran condiciones de equilibrio.

partiendo de una configuración simétrica (1):

1	2	2
1	0	1
2	2	1

al cabo de algunas generaciones se observan comportamientos definidos:

Uno de los factores que generan situaciones de equilibrio inestable es el desbalanceo de

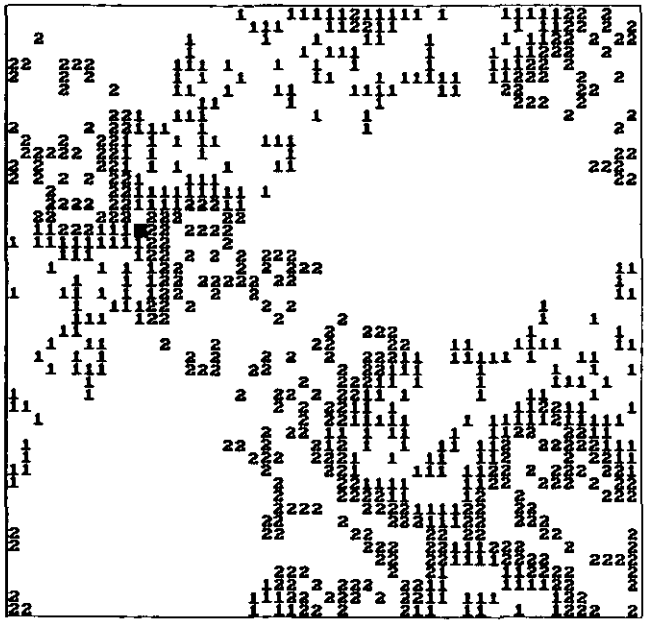


Figura 6-2: Surgimiento de patrones espirales en juego de las bestias.

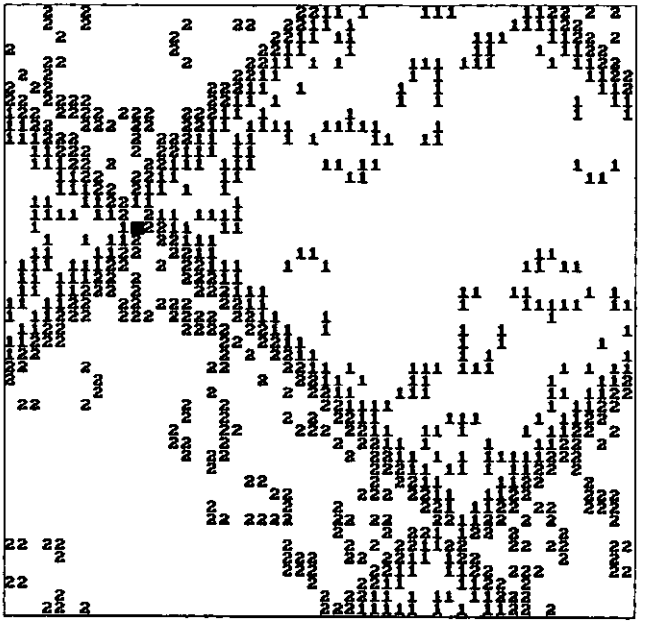


Figura 6-3: Formación de zonas estables

las reglas, en este modelo se aprecia, que la permanencia de una especie se ve favorecida por la presencia en sus cecanías de unos pocos individuos de la otra especie, pues en ese caso se evitan las reglas generadoras de estados quiescentes de *vida*, ésto genera comportamientos caóticos en las fronteras.

Por ser este juego una extensión (trivial en lo tocante a su dinámica) de *vida*, hereda las propiedades computacionales, y la pertenencia a la clase IV,

6.2 Juego depredador - presa.

6.2.1 Definiciones.

En este juego se plantean dos especies, teniendo un comportamiento asimétrico, una de ellas (la presa, correspondiente a las células en estado 1) se alimenta del medio y se reproduce con la condición que existan dos de su especie en la cercanía.

Un efecto de la reproductividad de la presa es que ésta tenderá a llenar el espacio disponible, las presas no pueden sobrevivir aisladas .

La otra especie es un depredador que se reproduce cuando existen presas cerca (se trata de obtener el comportamiento planteado por Negrete "el que come, se reproduce" [2]), además hay dos etapas del depredador que corresponde a un ciclo de vida más largo que el de la presa, un depredador joven (estado 2) sin presas puede sobrevivir un paso más en el tiempo, convirténdose en un depredador viejo (estado 3). Dicho estado se caracteriza por ajustarse a un modelo de depredador parco, puesto que una presa tiene

más tolerancia a una colonia de depredadores (estado 3).

La especificación de este AC es:

$$
\begin{aligned}
\mathrm{S} &= \{0,1,2,3\}, d = 2 \\
N &= \text{Vecindad de Moore} \\
f(c+N) &= \begin{cases}
1 \text{ si } \begin{cases} c=0, N_1=2, N_2, N_3=0 \\ c=1, N_1 \geq 2, N_2 \leq 2, N_3 \leq 3 \end{cases} \\
2 \text{ si } \begin{cases} c=0, N_1>0, N_2 \geq 2 \\ c=1, N_1<2, N_2 \geq 3 \\ c=2, N_1 > N_2 \end{cases} \\
3 \text{ si } \begin{cases} c=0, N_1>0, N_2=0, N_3>1 \\ c=2, N_1=0, N_1 \leq N_3 \\ c=3, N_1 \geq 2 \end{cases} \\
0 \text{ en cualquier otro caso}
\end{cases}
\end{aligned}
$$

6.2.2 Algunos patrones

Aunque la pertenencia a la clase IV, de este AC no está definida, se pueden observar algunos patrones estables, sin embargo el mayor número de estados haría la aplicación de los algoritmos de búsqueda y análisis demasiado complejo.

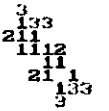


Figura 6-4: Patrón periódico (p2) en predador-presa

6.2.3 Simulación.

Al ejecutar las simulaciones se observan patrones de equilibrio inestable, dado que cada una de las colonias tiende a expandirse, las zonas estables no se forman tan fácilmente como en los modelos anteriores, un posible problema a resolver para este AC es comparar sus transiciones (con respecto λ de Langton o alguna otra magnitud, con otros AC)

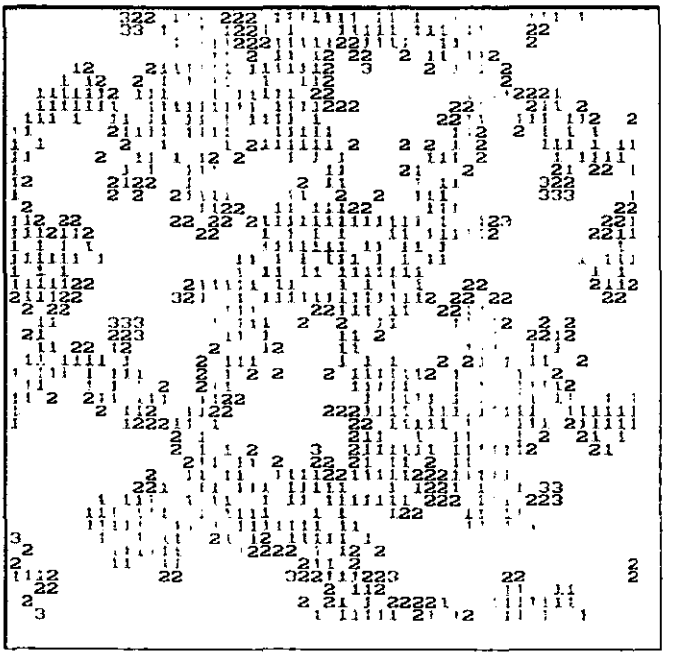


Figura 6-5: Expansión de 2 especies en todo el plano.

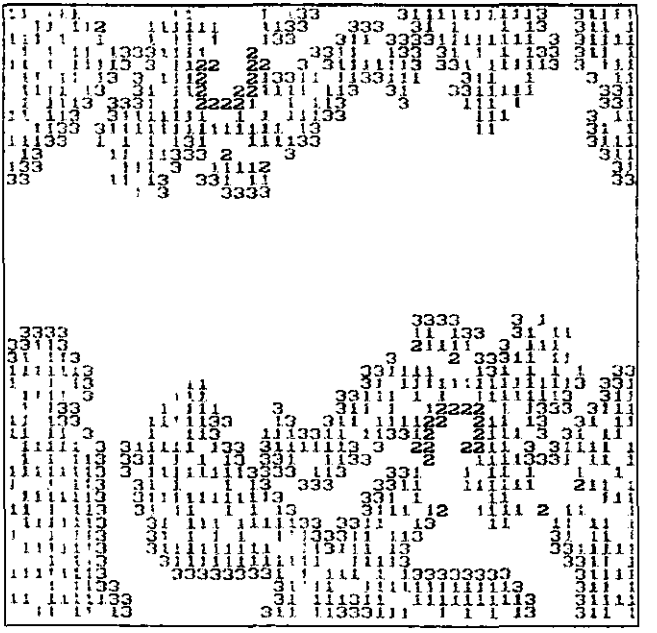


Figura 6-6: Las colonias de presas rodeadas tienden a colapsarse.

Capítulo 7

Conclusiones

En la investigación, uno de los primeros pasos para encontrar respuestas, es formular las preguntas correctas. El presente trabajo arroja varias ideas que pueden desarrollarse.

Como se puede ver en la construcción de los anillos de un AC restringido 3-1, las células cercanas al borde cambian sus vecindades radicalmente, lo que hace que el descendiente de una configuración se identifique con otra configuración que inclusive podría no ser alcanzable en el AC no restringido. Una mejor restricción a un AC sería, partiendo de una configuración inicial (que no es trivial elegir, puesto que muchos AC tienen diagramas de transición disconexos), tomando un número finito de configuraciones conexas, al llegar a un estado que se quiera excluir, se crearía un estado "pozo" que no creara nuevos arcos, este método tiene la ventaja que los diagramas de transición de AC restringido serán subgráficas del diagrama correspondiente en el AC no restringido, sin embargo, tal construcción sobre el diagrama, requeriría tener en cuenta las transformaciones que se

hacen al espacio base.

Por lo anterior, el diagrama de evolución de un AC restrigido, es muy distinto del diagrama del mismo AC restringido en otros anillos, pero deben existir parámetros que permitan encontrar una regularidad, ya sea en los lenguajes regulares asociados o en la gráfica misma, sería interesante tomar reglas de cada comportamiento y elaborar una clasificación preliminar, tal vez cualitativa en principio para estos diagramas.

Las construcciones presentadas dejan en claro cómo implementar búsquedas para patrones dados, sin embargo aún requieren una capacidad de almacenamiento muy grande, trabajar en la optimización de los algoritmos, para exhibir en lugar de los modelos básicos, patrones de tamaño mediano. En lo tocante a la búsqueda de jardines del edén, el diagrama de subconjuntos es mejor que las búsquedas exhaustivas que se realizan normalmente, elaborar un programa para hacer detecciones de nuevos jardines del edén es posible con los recursos suficientes.

Los juegos presentados muestran patrones que pueden ser detectados mediante los diagramas de subconjuntos o de De Bruijn, el mayor número de estados implica un mayor tiempo de proceso, pero una vez realizada la optimización para el *juego de la vida*, la extensión a los otros dos juegos es inmediata.

Elaborar autómatas para procesos reales permite obtener el comportamiento deseado (por ejemplo, la morfogénesis) de manera artificial, aunque no por ello se entiende la naturaleza intrínseca del fenómeno.

Bibliografía

[1] Gutowitz. Howard. Artificial life Online , http://alife.santafe.edu/alife/. Abril 1 , 1997

[2] Negrete. José, Yankelevitch et.. al. Juegos ecológicos y epidemiológicos, 1976, foccavi/conacyt.

[3] Gutowitz. Howard, ed. Cellular Automata: Theory and Experiment, 1989, NORTH-HOLLAND.

[4] Durand - Lose. Jérôme Reversible space-time simulation of cellular autómata, Laboratoire Bordelais de Recherche en Informatique (Report de recherche numéro 1177-97), 2 Oct 1997.

[5] Demongeot. J., Dynamical Systems and Cellular Automata, 1985, Academic Press.

[6] McIntosh. Harold V.,Linear Cellular Automata, Rev. 10 Ago 1989, Universidad Autónoma de Puebla.

[7] McIntosh. Harold V.,Linear Cellular Automata via de Bruijn Diagrams, Rev. 10 Ago 1991, Universidad Autónoma de Puebla.

[8] McIntosh. Harold V.,A Zoo of Life forms, Departamento de Aplicación de Microcomputadoras,Instituto de Ciencias, Universidad Autónoma de Puebla. Oct 16, 1988.

[9] McIntosh. Harold V., Still Lifes. Life's Still Lifes, Departamento de Aplicación de Microcomputadoras, Instituto de Ciencias, Universidad Autónoma de Puebla. Sep 10, 1988.

[10] McIntosh. Harold V., Reversible Cellular Automata, Departamento de Aplicación de Microcomputadoras, Instituto de Ciencias, Universidad Autónoma de Puebla. Sep 10, 1991.

[11] Buckingham. Dave, Callahan. Paul , Tight bounds on periodic cell configurations in Life, Aceptado en Experimental Mathematics, publicación pendiente.

[12] Moore, E. Machine Models in self reproduction, Proceedings of Simposyum on Applied Mathematics, Vol 14, 1962.

[13] Myhill, J. The converse of Moore's garden-of-eden theorem, Proceedings of Simposyum on Applied Mathematics, No. 14, 1963

[14] Amoroso, S., Patt, Y., Decision Procedure for surjectivity and injectivity of parallel maps for tesellation structure. Journal of Computer and System Sciences Vol 6; 1972.

[15] Wolfram, S., Statistical Mechanics of Cellular Automata, Reviews of Modern Physics, 55 (July 1983) 601-644.

[16] Wolfram S., Cellular Automata , Los Alamos Science (Fall 1983) 2-21.

[17] Wolfram S.,Cellular Automata as Simple Self-Organizing Systems,Caltech preprint CALT-68-938.

[18] J. von Neumann, "The General and Logical Theory of Automata", en J. von Neumann. "Collected Works" (ed. A.H. Taub), Vol. 5, p.288; J. von Neumann, "Theory of Self-Reproducing Automata", (ed. A.W. Burks), Univ. of Illinois Press (1966); ed. A.W. Burks, "Essays on Cellular Automata", Univ. of Illinois Press (1970).

[19] Gardner M., Mathematical Games - The fantastic combinations of John Conway's new solitaire Game "life", Scientific American, Oct 1970.

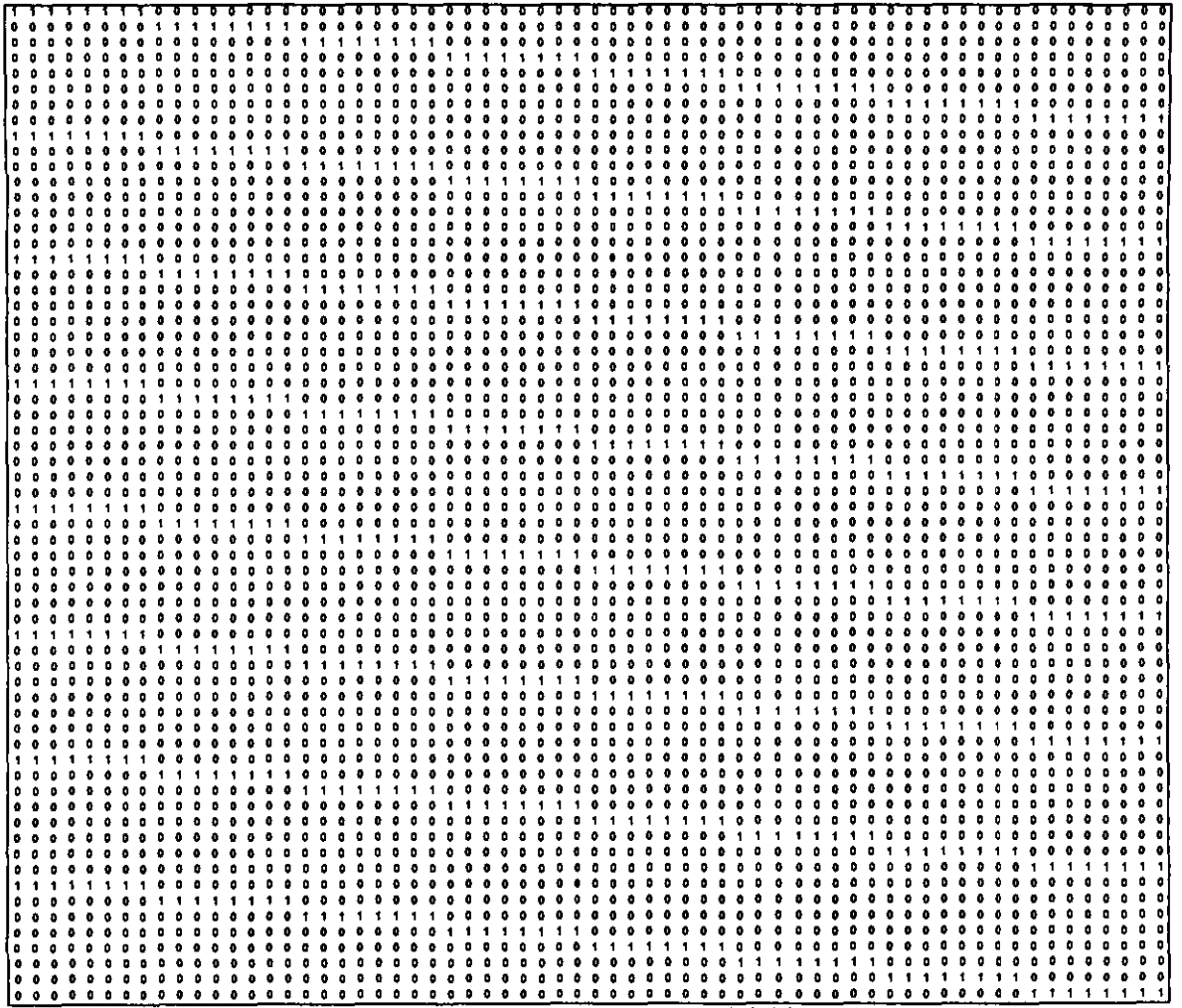
[20] Golomb, Solomon W, Shift registry Sequences, Holden-Day Inc.., San Francisco, 1967.

[21] Bak. P., Self-organized criticallity in the Game of Life, Nature 6251 V 342, Dic 14, 1989.

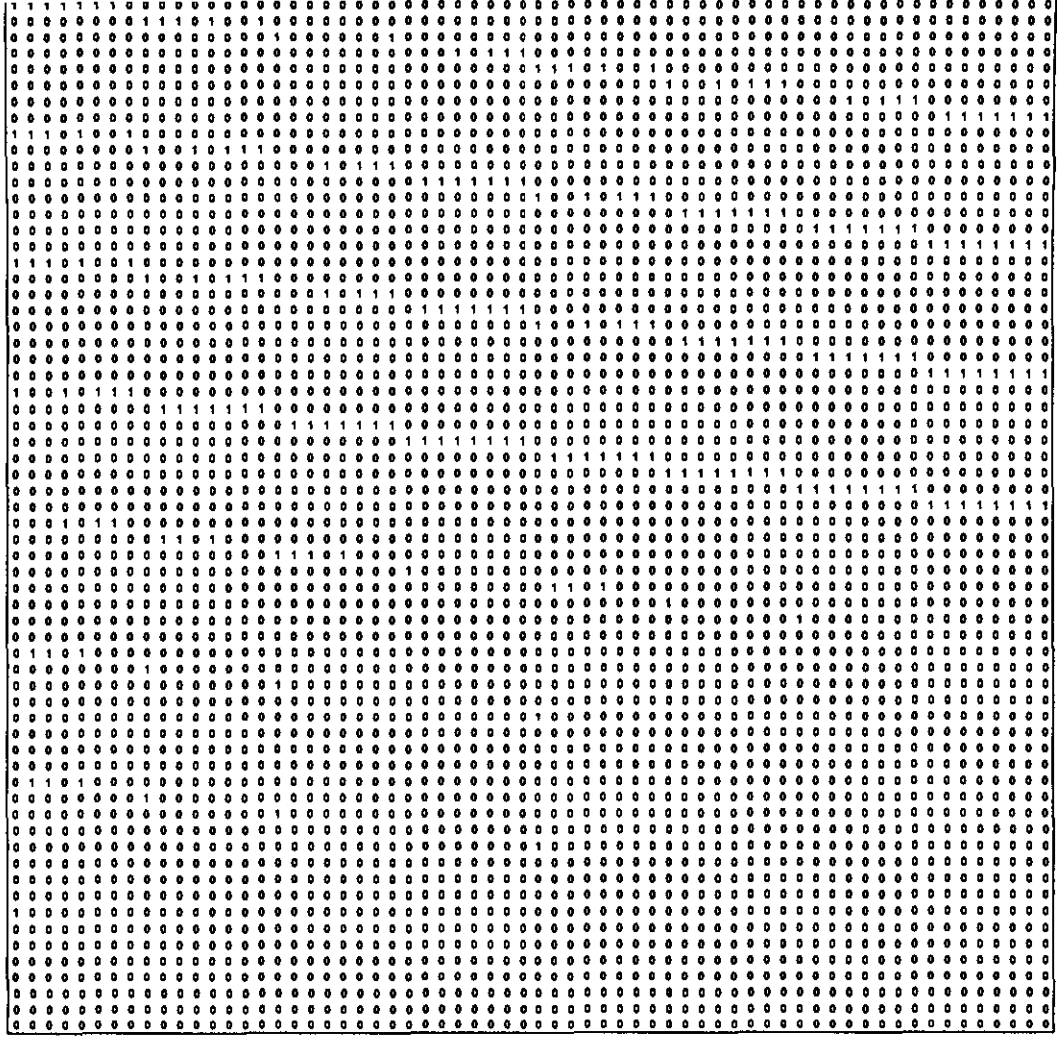
[22] Wensche. Andrew, Discrete Dynamics Lab, Programa disponible en: ftp://alife.santafe.edu/pub/SOFTWARE/ddlab.

[23] Linz. Peter, An introduction to formal languages and automata , Lexington, massachusetts : D.C. Heath, 1990.

Anexos



Anexo 1: Matriz de De Bruijn Correspondiente a un autómata binario con vecindad de Moore



Anexo 2: Matriz de De Bruijn para los patrones fijos en Juego de la vida

```
/*
Mariano Domínguez Molina.
Definición en lenguaje C de las funciones usadas en las simulaciones

*/

unsigned int fn(unsigned int V[3][3])
//Función generadora del juego predador-presa
{
int i,j,C1=0,C2=0,C3=0,c;
for(i=0;i<3;i++)
   for(j=0;j<3;j++)
   {
   if (V[i][j]==1 && !(i==1 && j==1)) C1++;
   if (V[i][j]==2 && !(i==1 && j==1)) C2++;
   if (V[i][j]==3 && !(i==1 && j==1)) C3++;
   }
c=V[1][1];
if (c==0 && C1>2 && C2==0 && C3==0) return 1;
if (c==1 && C1>0 && C2==0 && C3==0) return 1;

if (c==0 && C1>0 && C2>0) return 2;
if (c==2 && C1>0) return 2;

if (c==0 && C1>0 && C2==0 && C3>0) return 3;
if (c==2 && C1>0) return 3;
if (c==3 && C1>0) return 3;
return 0;
}

unsigned int fnBestias(unsigned int V[3][3])
//funci¢n generadora para el juego de las bestias
{
int i,j,C1=0,C2=0;
for(i=0;i<3;i++)
   for(j=0;j<3;j++)
   {
   if (V[i][j]==1 && !(i==1 && j==1)) C1++;
   if (V[i][j]==2 && !(i==1 && j==1)) C2++;
   }
if (C2==0)
{
  if(C1>3) return 0;
  if(C1<2) return 0;
  if(V[1][1]==0 && C1==3) return 1;
}
 else
    if (C1==0)
      {
      if(C2>3) return 0;
      if(C2<2) return 0;
      if(V[1][1]==0 && C2==3) return 2;
      }
    else
    {
    if(C2>C1) return 2;
    if(C1>C2)       return 1;
    }
return V[1][1];
}
```

```
unsigned int fnVida(unsigned int V[3][3])
//funci¢n generadora para el juego de la vida
{
int i,j,C1=0;
for(i=0;i<3;i++)
   for(j=0;j<3;j++)
   if (V[i][j]==1 && !(i==1 && j==1)) C1++;
if(C1>3) return 0;
if(C1<2) return 0;
if(V[1][1]==0 && C1==3) return 1;
return V[1][1];
}
```

```
/*

Mariano Domínguez Molina
Programa simulador de los autómatas usados en la tesis.

*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#define Max N 50

const int MaxN =50;
unsigned int nS,n,nVec;
int Nx[Max N];
int Ny[Max N];
unsigned int Espacio[Max N][Max N];

void Init()
{
int gdriver = DETECT, gmode, errorcode;
unsigned int i,j;
nS=0;
n=0;
for(i=0;i<=MaxN;i++)
      {
      Nx[i]=0;
      Ny[i]=0;
      for(j=0;j<=MaxN;j++)
         Espacio[i][j]=0;
      }

initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode != grOk)  /* an error occurred */
{
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);             /* return with error code */
}
setgraphmode(VGAHI);
setcolor(YELLOW);
}


unsigned int Intpow(unsigned int a,unsigned int b)
{
unsigned int i,res;
res=1;
if (b<=0) return 1;
for (i=0;i<b;i++)
  res=res*a;
return res;

}
```

```
int LeeConfig(char *s)
{
FILE *Archivo;
char buf[80];
char *cad=" ";
unsigned int x,y,i,j,st;
if ((Archivo= fopen(s,"rt")) == NULL)
   {
   fprintf(stderr,"No existe el archivo\n");
   return 1;
   }
cleardevice();
while (!feof(Archivo))
   {
   fgets(buf,80,Archivo);
   sscanf(buf,"%d,%d,%d\n",&x,&y,&st);
   Espacio[x][y]=st;
   }
 for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      {
      cad[0]=Espacio[i][j]+48;
      if(Espacio[i][j] == 1) setcolor(CYAN);
      if(Espacio[i][j] == 2) setcolor(YELLOW);
      if(Espacio[i][j]!=0) outtextxy(i*8,j*8,cad);
      }
return 0;
}

int EscribeConfig(char *s)
{
FILE *Archivo;
char buf[80];
int i,j;
if ((Archivo= fopen(s,"wt")) == NULL)
   {
   fprintf(stderr,"No puedo abrir el archivo\n");
   return 1;
   }
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      fprintf(Archivo,"%d,%d,%d\n",i,j,Espacio[i][j]);
return 0;
}

int Lee(char *s)
{
FILE *Archivo;
char buf[80],nombre[80];
unsigned int x,y,i=0,j=0;

if ((Archivo= fopen(s,"rt")) == NULL)
   {
   fprintf(stderr,"No existe el archivo\n");
   return 1;
   }
while (!feof(Archivo))
   {
     strcpy(buf,"");
     fgets(buf,80,Archivo);
```

```
      if(strcmp(buf,"#nestados\n")==0)
       {
       fgets(buf,80,Archivo);
       sscanf(buf,"%d\n",&nS);
       }
      if (strcmp(buf,"#lanillo\n")==0)
       {
       fgets(buf,80,Archivo);
       sscanf(buf,"%d\n",&n);
       }
      if (strcmp(buf,"#nombre aut\n")==0)
       fgets(nombre,80,Archivo);
      if (strcmp(buf,"#vecindad\n")==0)
       {
       fgets(buf,80,Archivo);
       i=0;
       while(strcmp(buf,"#fin\n"))
          {
          sscanf(buf,"%d,%d\n",&x,&y);
          Nx[i]=x;
          Ny[i]=y;
          fgets(buf,80,Archivo);
          i++;
          }
          nVec=i-1;
       }
    }
return 0;
}

unsigned int fn(unsigned int V[3][3])
{
int i,j,C1=0,C2=0,C3=0,c;
for(i=0;i<3;i++)
   for(j=0;j<3;j++)
   {
   if (V[i][j]==1 && !(i==1 && j==1)) C1++;
   if (V[i][j]==2 && !(i==1 && j==1)) C2++;
   if (V[i][j]==3 && !(i==1 && j==1)) C3++;
   }
c=V[1][1];
if (c==0 && C1==2 && C2==0 && C3==0) return 1;
if (c==1 && C1>=2 && C2<=2 && C3<=3) return 1;

if (c==0 && C1>0 && C2>=2) return 2;
if (c==1 && C1<2 && C2>=3) return 2;
if (c==2 && C1>C2) return 2;

if (c==0 && C1>0 && C2==0 && C3>1) return 3;
if (c==2 && C1==0 && C1<=C2) return 3;
if (c==3 && C1>=2) return 3;
return 0;
}

void Ejecuta()
{
unsigned int ns,ngen=0,k=0,S[Max N*Max N],
X[Max N*Max N],Y[Max N*Max N];
int i,j,l,m,x,y;
unsigned int Vec[3][3];
```

```
char *s=" ";
while(ngen<100)
     {
     k=0;
     ngen++;
     for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         {
         for(l=-1;l<=1;l++)
            for(m=-1;m<=1;m++)
             {
             x=i+l;
             if(x<0) x=n-1;
             if(x>=n) x=0;
             y=j+m;
             if(y<0) y=n-1;
             if(y>=n) y=0;
             Vec[l+1][m+1]=Espacio[x][y];
             }
         ns=fn(Vec);
         if (ns!=Espacio[i][j])
            {
            S[k]=ns;
            X[k]=i;
            Y[k]=j;
            k++;
            }
         }
     for(i=0;i<k;i++)
      {
      Espacio[X[i]][Y[i]]=S[i];
      s[0]=S[i]+48;
      if(S[i]!=0)
         {
         setcolor(BLACK);
         outtextxy(X[i]*8,Y[i]*8,"0");
         setcolor(YELLOW);
         if(Espacio[X[i]][Y[i]] == 1) setcolor(CYAN);
         if(Espacio[X[i]][Y[i]] == 2) setcolor(YELLOW);
         if(Espacio[X[i]][Y[i]] == 3) setcolor(MAGENTA);
         outtextxy(X[i]*8,Y[i]*8,s);
         }
      else
         {
         setcolor(BLACK);
         outtextxy(X[i]*8,Y[i]*8,"0");
         setcolor(YELLOW);
         }
      }
    }
}

void Editor()
{
char com;
char *s=" ";
int i,j,x,y;
cleardevice();
for (i=0;i<n;i++)
    for (j=0;j<n;j++)
```

```
      {
      s[0]=Espacio[i][j]+48;
      if(Espacio[x][y] == 1) setcolor(CYAN);
      if(Espacio[x][y] == 2) setcolor(YELLOW);
      if(Espacio[x][y] == 3) setcolor(MAGENTA);
      if(Espacio[i][j]!=0) outtextxy(i*8,j*8,s);
      }
x=0;
y=0;
com='a';
while(com!='q')
     {
     setcolor(YELLOW);
     outtextxy(x*8,y*8,"Û");
     com=getch();
     setcolor(BLACK);
     outtextxy(x*8,y*8,"Û");
     setcolor(YELLOW);
     s[0]=Espacio[x][y]+48;
     if(Espacio[x][y] == 1) setcolor(CYAN);
     if(Espacio[x][y] == 2) setcolor(YELLOW);
     if(Espacio[x][y] == 3) setcolor(MAGENTA);
     if(Espacio[x][y]!=0) outtextxy(x*8,y*8,s);
     else outtextxy(x*8,y*8," ");
     if (com=='8') y--;
     if (com=='2') y++;
     if (com=='4') x--;
     if (com=='6') x++;
     if(x<0) x=n-1;
     if(y<0) y=n-1;
     x=x%n;
     y=y%n;
     if (com=='5')
      {
      Espacio[x][y]=Espacio[x][y]+1;
      if (Espacio[x][y]>=nS) Espacio[x][y]=0;
      };
     if (com==' ') Ejecuta();
     if (com=='r') LeeConfig("config.dat");
     if (com=='w') EscribeConfig("config.dat");

     }
};

main()
{
char *s;
Init();
Lee("prueba.aut");
Editor();
return 0;
}
```