Automatas celulares

Juan Carlos Olivier Jasso

11 Abril 2016

Contents

1 Introduccion

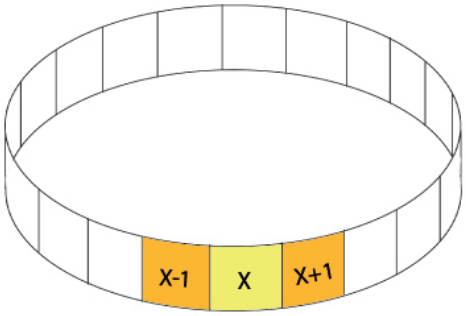
Modelo matematico para un sistema dinámico que evoluciona en pasos discretos. Es decir, es un modelo matemático para sistemas que soportan cambios; además, estos cambios se suceden cada tiempos constantes, razón por la cual, se usa una escala de enteros.

La idea de la creacion de los automatas celulares nacion en la decada de 1940 en donde John Von Neumann, intentaba la creacion de un sistema que pudiera autoreplicarse o extenderse de forma autonoma, mediante un modelo matematico, aplicados en una red rectangular. Contenian celulas las cuales asemejaban al proceso de vida en donde crecian, se reproducian y morian conforme pasaba el tiempo, Tienen diferentes valores y comportamientos entre si.

Se define Lattice,es una cuardicula de dimension N e infinitamente extendida. Dentro de cada una de las celdas en la cuadricula se encuentra una *"Celula"*. Cada celula toma un valor de un conjunto finito de estados k, el ciclo de vida de la celula y su evolucion no dependera solo de ella, si no, tambien de su entrono, los cuales determinaran sus condiciones de vida. Para determinar su evolucion, la funcion de reticula, se aplica a cada una de la celulas dentro de la *lattice*, toma todos los valores de la celda y sus alrededores-, asi podra determinar los nuevos valores de cada celula.

2 Automatas celulares unidimensionales

Consiste en una sola fila de células a los que se aplica un principio de vecindad básico de dos vecinos por célula y a los que igualmente se pueden aplicar las diversas condiciones de frontera que hemos nombrado anteriormente



Autómata celular unidimensional con un radio de vecindad r=1 y una condición de frontera periódico.

Como ejemplo, podemos tomar un autómata celular unidimensional con un radio de vecindad r=1, dos estados (0 y 1) y una condicion de frontera de tipo periódico, como el que se muestra en la imagen. Usaremos para este caso un tamaño de diez células y unas funciones de transición basadas en lo siguiente:

- Si ambos vecinos de la célula tienen el mismo estado, el estado de la célula a la que se aplica la funcion cambiará.
- Si ambos vecinos de la célula tienen distinto estado, el estado de la célula a la que se aplica se mantendrá igual.

Stephen Wolfram clasificó el comportamiento de los automatas celulares unidimensionales. Según Wolfram, todo automata celular pertenece a una de las siguientes clases:

- Clase I. La evolución lleva a una configuración estable y homogénea, es decir, todas las células terminan por llegar al mismo valor.
- Clase II. La evolución lleva a un conjunto de estructuras simples que son estables o periódicas.
- Clase III. La evolución lleva a un patrón caótico1.
- Clase IV. La evolución lleva a estructuras aisladas que muestran un comportamiento complejo (es decir, ni completamente caótico, ni completamente ordenado, sino en la línea entre uno y otro, este suele ser el tipo de comportamiento más interesante que un sistema dinámico puede presentar).

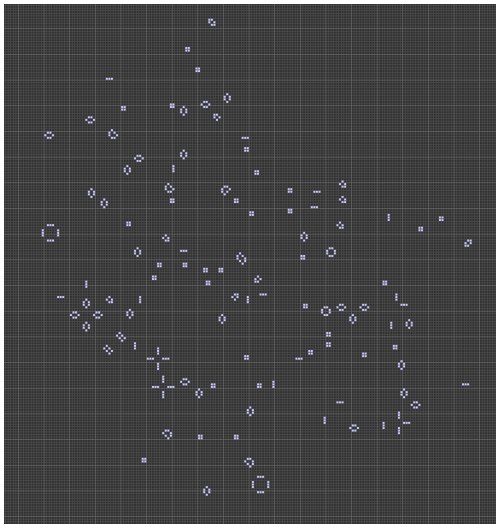
3 Autómatas celulares bidimensionales

Un autómata celular finito de 2 estados es la simulación más simplificada posible de una célula viva sobre un tejido. Representa una unidad de interacción que tan solo puede presentar 2 estados (activo e inerte, encendido y apagado, vivo o muerto, o como quieran denominarse). A pesar de su simplicidad inicial permite simular la evolución de sistemas complejos y analizar pautas tales como el comportamiento emergente. Un nuevo nivel de complejidad consiste en trabajar con un autómata celular de 2 estados, situado en un universo plano (de 2 dimensiones) ilimitado (toroidal, cerrado circularmente en sus dos direcciones).

El autómata posee solo 2 estados, representados por dos imagenes distintas. El estado evolutivo del autómata dependerá de su estado actual y el de su entorno (sus 8 células vecinas, derecha, izquierda, arriba, abajo y las cuatro esquinas).El universo es cerrado e ilimitado (toroidal), de forma que las últimas celdas de la derecha están unidas (interactuan) con las primeras de la izquierda y viceversa, al igual que las de la línea inferior lo hacen con la primera y viceversa. La evolución es discreta y simultánea para todos los componentes (células) del universo. Cada generación surgirá de golpe reemplazando completamente a la anterior.

Las reglas para una evolucion son las siguientes:

- Una célula se inactiva (o permanece inactiva) si posee menos de 2 o más de 3s vecinas activas (muerte).
- Una célula mantiene su estado, sea este cual fuera, si tiene tan solo 2 células vecinas activas (conservación).
- Una célula cobra actividad (o permanece activa si ya lo estaba) cuando la rodean 3 células activas (nacimiento).

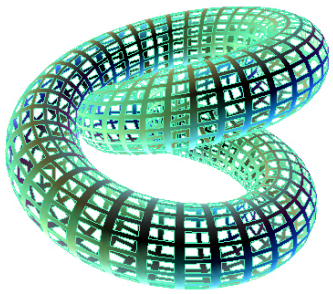


4 Autómatas celulares tridimensionales

Ampliamente estudiados por Carter Bays, los automatas celulares de tres dimensiones se enfocan principalmente para encontrar una regla de evolución en tres dimensiones que sea la sucesora del Juego de la Vida en el espacio tridimensional, muchos de sus resultados son de tipo cuantitativo, basados principalmente en la simulación de varias reglas de evolución en pequeños espacios tridimensionales para encontrar estructuras que sean similares al Juego.La aplicación de estos autómatas celulares se centra en el estudio en campos estadísticos.

Son automatas celulares que se desarrollan en un ámbito de Z x Z x Z, que usan el principio de vecindad de Moore a nivel tridimensional. Esto implica que la función de transición ha de tener en cuenta el estado de veintiseis células vecinas además del estado de la célula en cuestión.

Los automatas celulares tridimensionales pueden seguir los mismos principios que los anteriores, aunque dado el incremento de la complejidad de los calculos y el procesamiento, su puesta en práctica no es simple. Sin embargo, y puesto que se pueden considerar automatas celulares tridimensionales con las mismas variables de tipo frontera en el ámbito teórico, se debe considerar que la representacion de los automatas celulares tridimensionales considerando una frontera periódica es en forma de toroide.



5 Codigo twocellular

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;


/*Funcion en la cual consulta la cantidad de vecinos que tiene la celula en
int neighbours(int a, int b, int c, int d, int e, int f, int g, int h)
{
 int result =a+b+c+d+e+f+g+h;
 return result;
}




int main(void)
{
 int i, j;
 int N = 32; // tamanio de la lattice
 //Crecion de objetos dinamicos para el arreglo de tamanio N
 int** A = new int* [N];

 for(i=0;i<N;i++) // creacion de los objetos en el arreglo dinamico
  A[i] = new int[N] ;

  int** B = new int* [N];

  for(i=0;i<N;i++)
   B[i] = new int[N] ;

  // se generan las celulas las celulas en la lattice 0 si no existe y 1 si
  // Estado inicial
  cout << "Initial State " << endl;
  for(i=0;i<N;i++)
  {
   for(j=0;j<N;j++)
```

```
    {
     A[i][j] = rand() % 2;
    }
   }

//impresion de la primer lattice
   for (i=0; i<N; i++)
   {
    for(j=0; j<N; j++)
    {
     cout << A[i][j] << " ";
    }
     cout << endl;
   }

   int T = 100; // cantidad de iteraciones
   for(int t=0;t<T;t++)
   {
    for(i=0;i<N;i++)
    {
     for(j=0;j<N;j++)
     {
      B[i][j] = A[i][j]; // paso de la lattice original a la nueva lattice
     }
    }

    /*
         Recorre la lattice
    */
   for(i=0;i<N;i++)
    {
     for(j=0;j<N;j++)
     {
      int temp=  //GUarda la cantidad de vecinos al rededor de la celula
      neighbours(
      B[((i+1)%N)][j],
      B[((i+1)%N)][((j+1)%N)],
      B[i][((j+1)%N)],
      B[((i-1+N)%N)][((j+1)%N)],
      B[((i-1+N)%N)][j],
```

```
      B[((i-1+N)%N)][((j-1+N)%N)] ,
      B[i][((j-1+N)%N)] ,
      B[((i+1)%N)][((j-1+N)%N)]);

      // rule 1:Una celula cobra actividad (o permanece activa si ya lo esta
      if((B[i][j] == 0) && (temp == 3))
      {
       A[i][j] = 1;
      }

      // rule 2: Una celula mantiene su estado, sea este cual fuera, si tien
      if((B[i][j] == 1) && ((temp ==2) || (temp == 3)))
      {
       A[i][j] = 1;
      }

      // rule 3: Una celula se inactiva (o permanece inactiva) si posee men
      if((B[i][j] == 1) && ((temp >= 4) || (temp <= 1)))
      {
       A[i][j] = 0 ;
      }

   }
  }

cout << endl; //impresion de iteracion
cout << "t = " << t;
cout << endl;

//impresion de la lattice despues de cada evolucion
for(i=0;i<N;i++)
{
 for(j=0;j<N;j++)
 {
  cout <<  A[i][j] << " ";
 }
 cout << endl;
}
  cout << endl << endl;
}
```

```
// livera la memoria para no hace stackoverflow
for(i=0;i<N;i++)
 delete[] A[i];

delete[] A;


for(i=0;i<N;i++)
 delete[] B[i];

delete[] B;

return 0;


}
```

6 celullar automata

```
# include <cstdlib>
# include <iostream>
# include <ctime>

using namespace std;

int main ( );
void timestamp ( );

int main ( )
{
  int i;
  int j;
  int n;
  int step_num;
  char *x;
  char *x_old;

  timestamp ( );
```

```
  cout << "\n";
  cout << "CELLULAR_AUTOMATON:\n";
  cout << "  C++ version.\n";

  n = 80; // cantidad de pasos
  step_num = 80;

  x = new char[n+2]; //se crea en memorial la ubicacion de la celula actual
  x_old = new char[n+2]; //se crea en memorial la ubicacion de la celula muerta

  for ( i = 0; i <= n + 1; i++ ) // inicialicacion de la lattice
  {
    x[i] = ' ';
  }
  x[40] = '*'; //en la ultima ubicacion de ponte la primera celular

  for ( i = 1; i <= n; i++ )
  {
    cout << x[i];  // se imprime la primera fila
  }
  cout << "\n";

  for ( j = 1; j <= step_num; j++ ) // se guarda la lattice anterior
  {
    for ( i = 0; i < n + 2; i++ )
    {
      x_old[i] = x[i];
    }
    for ( i = 1; i <= n; i++ )
    {
//
//  The transformation rules are:
//
//  111  110  101  100  011  010  001  000
//   |    |    |    |    |    |    |    |
//   0    0    0    1    1    1    1    0
//
//  which means this rule has binary code 00011110 = 16 + 8 + 4 + 2 = 30

//  Determina si vive o muere depende de lo que se tenga en la lattice donde se guard
```

```
      if ( ( x_old[i-1] == ' ' && x_old[i] == ' ' && x_old[i+1] == '*' ) ||
           ( x_old[i-1] == ' ' && x_old[i] == '*' && x_old[i+1] == ' ' ) ||
           ( x_old[i-1] == ' ' && x_old[i] == '*' && x_old[i+1] == '*' ) ||
           ( x_old[i-1] == '*' && x_old[i] == ' ' && x_old[i+1] == ' ' ) )
      {
        x[i] = '*';
      }
      else
      {
        x[i] = ' ';
      }
    }
//
// Hacer cumplir las condiciones de contorno periódicas .
//
    x[0] = x[n];
    x[n+1] = x[1];

    for ( i = 1; i <= n; i++ )
    {
      cout << x[i];
    }
    cout << "\n";
  }
//
//  Se libera la memoria por el stackoverflow
//
  delete [] x;
  delete [] x_old;
//
//  Terminate.
//
  cout << "\n";
  cout << "CELLULAR_AUTOMATON:\n";
  cout << "  Normal end of execution.\n";
  cout << "\n";
  timestamp ( );

  return 0;
```

```
}
//*******************************************************************************80

void timestamp ( )

/*
Imprime el tiempo en el cual se corrio el programa

*/
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct std::tm *tm_ptr;
  size_t len;
  std::time_t now;

  now = std::time ( NULL );
  tm_ptr = std::localtime ( &now );

  len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );

  std::cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}
```