シミュレーション物理

演習課題 (2)

January 27, 2018

筑波大学 情報学群 情報科学類 二年
江畑 拓哉 (201611350)

Contents

1 実験の目的

本実験はセル・オートマトン（CA）の挙動を確認するとともに、そのアルゴリズムを理解するためのものである。具体的には、184 規則と呼ばれる規則に基づいてセル・オートマトンのプログラムを作成し、これを複数の密度を持つ初期状態に対して適用、その挙動を観察する。

2 実験方法

作成したプログラムを実行する。　実行環境についての情報を以下に列挙する。

- プログラムとコンパイル
  - Manjaro Linux 17.1.2
  - gcc (GCC) 7.2.1 20171224
  - GNU Emacs 26.0.91 (build 1, $x86_{64}$-pc-linux-gnu, GTK+ Version 3.22.26) of 2018-01-26

3 実験結果

以下の実験結果を得た。

Listing 1: Standard Output

```
density: 0.3
t =  0: 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 1 0 0 0
t =  1: 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0
t =  2: 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 1 0
t =  3: 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1
t =  4: 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0
t =  5: 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1
t =  6: 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0
```

```
t =  7: 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
t =  8: 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
t =  9: 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1
t = 10: 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0
t = 11: 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1
t = 12: 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0
t = 13: 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1
t = 14: 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0
t = 15: 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0
t = 16: 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0
t = 17: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0
t = 18: 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0
t = 19: 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0
t = 20: 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0

density: 0.5
t =  0: 0 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 1 1 0 0
t =  1: 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0
t =  2: 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 0 1
t =  3: 1 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 0 1 0
t =  4: 0 1 0 0 0 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1
t =  5: 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 1 0 1 0
t =  6: 0 1 0 1 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1
t =  7: 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1 0
t =  8: 0 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1
t =  9: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
t = 10: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
t = 11: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
t = 12: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
t = 13: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
t = 14: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
t = 15: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
t = 16: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
t = 17: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
t = 18: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
t = 19: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
t = 20: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

density: 0.7
t =  0: 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1
t =  1: 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 0
t =  2: 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1
t =  3: 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0
t =  4: 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1
```

```
t =  5: 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0
t =  6: 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1
t =  7: 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0
t =  8: 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1
t =  9: 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0
t = 10: 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1
t = 11: 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1
t = 12: 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1
t = 13: 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1
t = 14: 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1
t = 15: 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1
t = 16: 1 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1
t = 17: 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1
t = 18: 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1
t = 19: 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0
t = 20: 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1
```

4 考察

この実験によって 184 規則に従うセル・オートマトンの挙動はその殆どがずれていく様子を観察することができた。また境界同士で繋がっているモデルを採用しているため、大凡において整然とした斜め柄が描かれている。　また密度に応じて斜め柄の向きが変わっているように観察ができるが、これが 184 規則の特徴的ないくつかの遷移が原因しているものだと考えられる。例えば、 $001 \to 0$ $100 \to 1$ といった “1” の数が少ない場合の遷移は右に遷移しているように考えられ、 $011 \to 1$ といった “1” の数が多い場合の遷移は左に遷移しているように考えられる。　つまりこの規則はランダムに設定された結果現れたものではなく、はっきりとした恣意があった上で設定されて現れたものであると考えることができる。

5 プログラムのリスト

Listing 2: cellular-automaton.c

```c
#include <stdio.h>
#include <stdlib.h>

// 111, 101, 100, 011 -> 1 : others -> 0
static int rules[4] = {111, 101, 100, 11};

int check (int);
void celluar_automaton (int, int*, int*);
void generate_vector (float, int, int*);
```

```
10 void shuffle (int, int*);
11 void print_vector (int, int, int*);
12 void move_vector (int, int*, int*);
13
14 int main (void) {
15   int i, j;
16   float density[3] = {0.3, 0.5, 0.7};
17   int size = 20;
18   int prev[size];
19   int next[size];
20   for (i = 0; i < 3; ++i) {
21     printf ("density: %.1f\n", density [i]);
22     generate_vector(density[i], size, prev);
23     print_vector(0, size,  prev);
24     for (j = 1; j < 21; ++j) {
25       celluar_automaton(size, prev, next);
26       print_vector(j, size, next);
27       move_vector(size, prev, next);
28     }
29     printf ("\n");
30   }
31   return 0;
32 }
33
34 int check (int k) {
35   int i;
36   for (i = 0; i < (sizeof (rules)/ sizeof (rules [0])); ++i) {
37     if (k == rules [i]) {
38       return 1;
39     }
40   }
41   return 0;
42 }
43
44 void celluar_automaton (int size, int* prev, int* next) {
45   int i, k;
46   // i = 0
47   k = prev [size - 1] * 100 + prev [0] * 10 + prev [0 + 1] * 1;
48   next [0] = check (k);
49   // i = 1 ... (size - 1) - 1
50   for (i = 1; i < (size - 1); ++i) {
51     k = prev [i - 1] * 100 + prev [i] * 10 + prev [i + 1] * 1;
52     next [i] = check (k);
53   }
```

```
54    // i = size - 1
55    k = prev [(size - 1) - 1] * 100 + prev [size - 1] * 10 + prev [0] * 1;
56    next [i] = check (k);
57  }
58
59  void generate_vector (float density, int size, int* vec) {
60    int i;
61    int l = (int) (density * size);
62    for (i = 0; i < size; ++i) {
63      if (i < l) {
64        vec [i] = 1;
65      } else {
66        vec [i] = 0;
67      }
68    }
69    shuffle (size, vec);
70  }
71
72  void shuffle (int size, int* vec) {
73    int i;
74    srand (691);
75    for (i = 0; i < size; ++i) {
76      int j = (int) (10 * (rand () / (RAND_MAX + 1.0)));
77      int t = vec [i];
78      vec [i] = vec [j];
79      vec [j] = t;
80    }
81  }
82
83  void print_vector (int n, int size, int* vec) {
84    int i;
85    printf ("t = %2d:", n);
86    for (i = 0; i < size; ++i) {
87      printf (" %d", vec [i]);
88    }
89    printf ("\n");
90  }
91
92  void move_vector (int size, int* prev, int* next) {
93    int i;
94    for (i = 0; i < size; ++i) {
95      prev [i] = next [i];
96    }
97  }
```