セルオートマトンモデルによる
渋滞のシミュレーション

2019 年 2 月 27 日

卒業研究レポート

明治大学　総合数理学部　現象数理学科 4 年

小川将貴

目次

１　はじめに

私は普段から車やバイクを趣味としており、ドライブやツーリングを行い各地に出掛けている。そこで必ずと言っていいほど悩まされるのが渋滞である。特に週末や連休時期になると１、２時間もの間渋滞にはまってしまうことさえ容易にある。そこで私は渋滞問題における原因と解決策を知るべく本研究に取り組んだ。本論文では、コンピューターシミュレーションにより仮想渋滞を作り出すことで、現実に起こっている様々な条件下での渋滞を再現することを目標とする。

渋滞などの交通流を解析する方法として、流体モデルやセルオートマトンモデルなどがある。流体モデルは交通の流れを連続した液体のように考えたモデルで、セルオートマトンモデルは道路をセルに分割し、各セルごとに車の状態を離散的な時間の変化で考えたものである。

本論文ではセルオートマトンモデルを用いたシミュレーションを行い、渋滞の可視化を行うこととする。

２　　セルオートマトンモデル

２.１　セルオートマトンモデルとは

セルオートマトンの概念は 1940 年代に、ロスアラモス国立研究所で同僚であった Stanistaw Marcin Ulam と John von Neumann によって発見された。その後も研究が続き 1980 年代には、Stephen Wolfram が 1 次元セルオートマトンを体系的に研究し、セルオートマトンが様々な科学の領域で応用できると主張したことから、渋滞学においてもセルオートマトンが注目されるようになった。

ここでは、渋滞学においてセルオートマトンモデルによる研究を行っている西成活裕氏著書「渋滞学」[1]を参考に本論文を書き進めるものとする。

セルオートマトンモデルは、格子状に並んだセルによって構成されており、セルの状態は{0,1}の２通りで、内部状態を時間の経過とともに変化させていく

数理モデルである。ここでの時間・空間・状態量は離散的なもので、ある時間における各セルの内部状態はそのセルとそのセルに隣接するセルの内部状態によって決定される。様々な条件（ルール）を導入することで、より複雑な現象や、限られた条件下におけるシミュレーションが行うことが可能である。

２.２　既存のセルオートマトンモデルの紹介

ここでは既存の有名モデルの紹介だけになるが、下記のようなモデルが存在する。

・Rule184
  - 前方に車がいる→止まる　前方に車がいない→進む

・Slow-Start モデル
  - 車の慣性（１度停まった車は加速が遅れる効果）に対応

・Quick-Start モデル
  - 運転手の見通しに対応

・ASEP
  - 車の動きに確率を導入

・Fukui-Ishibashi(FI)モデル
  - 高速度に対応

・Nagel-Schreckenberg(NS)モデル
  - 高速度に対応、ランダムブレーキ効果に対応

Rule184 はセルオートマトンモデルに中では最も基本的かつ単純なモデルである。Slow-Start モデルは停止している車がすぐには加速できないという条件を模擬導入するため、１度停止した車は１ステップ待って動き出すというルールのモデルである。Quick-Start モデルは運転手が２台先の車の動きを見てある程度前方の車の動きを予測するというルールのモデルである。ASEP は Rule184 の確率版である。動ける状態にあるときに確率 p で進み、確率 1−p で停止するというものである。FI モデルは１度に２セル以上進むことができるというルールのモデルである。NS モデルは FI モデルにランダムブレーキ効果が導入され

たものである。ランダムブレーキ効果とは、２セル以上進むことができる場合でもある確率で１セル分減速するというルールのモデルである。

３　Rule184

３.１　Rule184 とは

Rule184 は、セルオートマトンモデルの中では最も基本的かつ単純なモデルである。

ここでは、「渋滞学のセルオートマトン」[2]を参考に Rule184 について説明する。

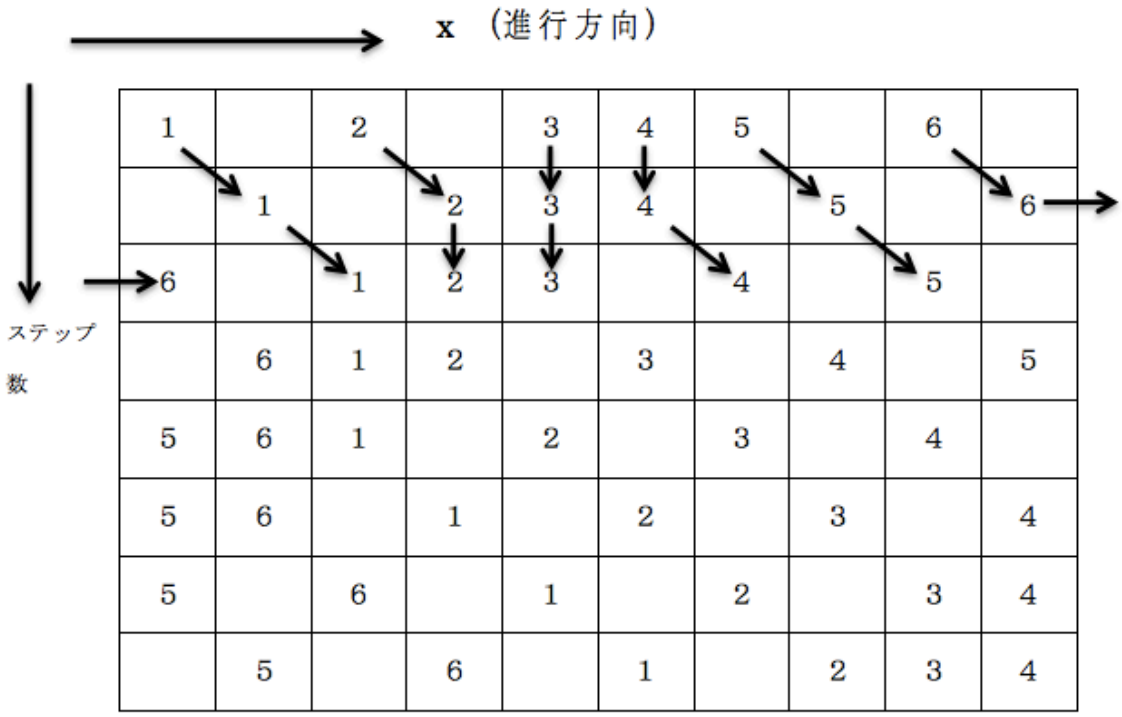
・車の進行方向は x 軸の正方向にとり、車が進める場合は１ステップで１セル分進む

・車は、前方のセルに車が存在する場合、次のステップで停止する

・車は、前方のセルに車が存在しない場合、次のステップで進む

・周期境界条件（最後のセルは、最初のセルを最後のセルの前方のセルと考え、進む場合は最初のセルに進む。言い換えると、道が環状になっているということ）

上記のルールに従いステップ数が増えていく。



1		2		3	4	5		6	
	1		2	3	4		5		6
6		1	2	3		4		5	
	6	1	2		3		4		5
5	6	1		2		3		4	
5	6		1		2		3		4
5		6		1		2		3	4
	5		6		1		2	3	4

数字が書かれてあるセルは車があることを表す。

上の図が Rule184 における車の動きである。上の図からわかることは、初期配置において 3 台の車が固まった渋滞クラスター（渋滞群）が、ステップ数が増加するごとに車の進行方向とは反対に移動しているということである。これは実際の交通渋滞においても見られる現象である。

３．２　Rule184 のシミュレーション

実際に Rlue184 をコンピューターによるシミュレーションを行ってみる。

下記に Processing によるプログラムと実行結果を写す。

「プログラム」

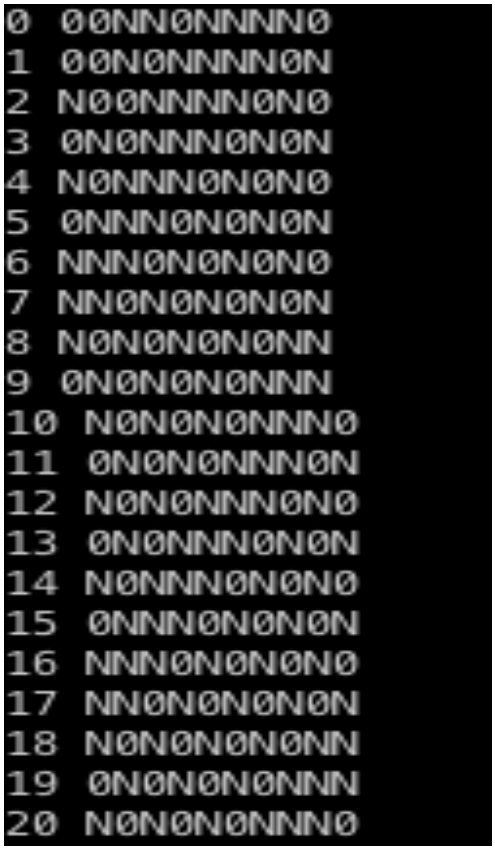
```
/*Rule184*/

/*
    0:車がないセル(空)
    1:車があるセル
*/
/*
```

```
   「ルール」
(0 <= i <= C−2)
        ・i セルに車があり、i+1 セルに車がないとき、
        次のステップでは、i セルに車はなく、i+1 セルに車がある。
        ・C-1 セル（最後のセル）に車があり、0 セル（最初のセル）に車がないとき、 次のステップでは、C-1 セルに車はなく、0 セルに車がある。
        ・初期配置はランダム。
*/

/*
セルの状態を配列 a で表す。
i 番目のセルに車がない状態を  a[i] = 0  で表す。
i 番目のセルに車がある状態を  a[i] = 1  で表す
*/

/*セルの数*/
int C=10;
/*車の数*/
int M=6;

/*C 個の要素を持つ配列を用意*/
int n=1;
int a[] = new int[C];
int a2[] = new int [C];     /*a も a2 も 1 つの配列である。*/
char b[] = {'0','N','?','S'};

void setup() {
   /*全ての配列要素を 0*/
   for (int i=0; i<C; i++) {
      a[i]=0;
      a2[i]=0;
   }
```

```
  /*M 台の車をランダムにセルに配置*/
  int value;
  value=0;
  randomSeed(0);
  while (value<M) {
    int r=int(random(0, C));

    if (a[r]==0) {
      a[r]=1;
      value=value+1;
    }
  }
  print(n-1); print(" ");

  for (int i=0; i<C; i++) {
    print(b[a[i]]);
  }
  frameRate(60);              /*移動の速さ*/
  println("");
}

void draw() {
  if (a[C-1] == 1 && a[0] == 0) {
    a2[C-1] = 0; a2[0] = 1;
  }
  for (int i=0; i<C-1; i++) {
    if (a[i] == 1) {
      if (a[i+1] == 0) {
        a2[i] = 0; a2[i+1] = 1;
      }
      else
      a2[i] = 1;
```

```
    }
  }
  for (int i = 0; i < C; i++)
    a[i] = a2[i];
  print(n++); print(" ");

  for (int i=0; i<C; i++) {
   print(b[a[i]]);
  }
  println("");
}
```

「実行結果」



ステップ数２０までの実行結果を載せておく。

ここから、ステップ数が増加する毎に「渋滞クラスターが進行方向と反対側に移動していること」が見て取れる。これは 3.1 でも述べたように、初期配置に

よって渋滞クラスターが存在する場合、その渋滞クラスターは時間経過によって進行方向とは反対方向に移動するという現象をシミュレーションによって再現することができた。この現象は実際の交通渋滞においても見られる現象である。

ここで、[1]で先行研究されていることだが、十分な時間が経過した際に、渋滞が発生する原因に密度が関係していることがわかっている。上のシミュレーションでは、セルの数１０個に対し、車の台数６台でシミュレーションを行った。ここでは渋滞が発生しているが、密度を考えると密度(ρとする)は 0.6 である。

渋滞が発生する際の密度には以下のような関係がある。

M=4, 密度 ρ <0.5	M=5, 密度 ρ =0.5	M=6, 密度 ρ >0.5
0 00N000NNN0 1 000N00NN0N 2 N000N0N0N0 3 0N000N0N0N 4 N0N000N0N0 5 0N0N000N0N 6 N0N0N000N0 7 0N0N0N000N 8 N0N0N0N000 9 0N0N0N0N00 10 00N0N0N0N0 11 000N0N0N0N 12 N000N0N0N0 13 0N000N0N0N 14 N0N000N0N0 15 0N0N000N0N 16 N0N0N000N0 17 0N0N0N000N 18 N0N0N0N000 19 0N0N0N0N00 20 00N0N0N0N0	0 00NN00NNN0 1 00N0N0NN0N 2 N00N0NN0N0 3 0N00NN0N0N 4 N0N0N0N0N0 5 0N0N0N0N0N 6 N0N0N0N0N0 7 0N0N0N0N0N 8 N0N0N0N0N0 9 0N0N0N0N0N 10 N0N0N0N0N0 11 0N0N0N0N0N 12 N0N0N0N0N0 13 0N0N0N0N0N 14 N0N0N0N0N0 15 0N0N0N0N0N 16 N0N0N0N0N0 17 0N0N0N0N0N 18 N0N0N0N0N0 19 0N0N0N0N0N 20 N0N0N0N0N0	0 00NN0NNNN0 1 00N0NNNN0N 2 N00NNNN0N0 3 0N0NNN0N0N 4 N0NNN0N0N0 5 0NNN0N0N0N 6 NNN0N0N0N0 7 NN0N0N0N0N 8 N0N0N0N0NN 9 0N0N0N0NNN 10 N0N0N0NNN0 11 0N0N0NNN0N 12 N0N0NNN0N0 13 0N0NNN0N0N 14 N0NNN0N0N0 15 0NNN0N0N0N 16 NNN0N0N0N0 17 NN0N0N0N0N 18 N0N0N0N0NN 19 0N0N0N0NNN 20 N0N0N0NNN0
渋滞なし	渋滞なし	渋滞あり

いずれもステップ数２０とし比較した。

上のシミュレーションにより、密度ρの値によって渋滞が起こる場合と起こらない場合が確認できた。

３．３　Rule184 の語源

Rule184 において３つのセルの場合を考えてみる。あるセルとその両隣のセルの計３つのセルの状態から、次のステップの状態が決まり、初期配置は８パターンのみで考えることができる。初期の状態をt_0とし、t_1の時の中心のセルについて考える。

① t_0 1 1 1 / t_1 1 1 0

② t_0 1 1 0 / t_1 1 0 1

③ t_0 1 0 1 / t_1 0 1 0

④ t_0 1 0 0 / t_1 0 1 0

⑤ t_0 0 1 1 / t_1 0 1 0

⑥ t_0 0 1 0 / t_1 0 0 1

⑦ t_0 0 0 1 / t_1 0 0 0

⑧ t_0 0 0 0 / t_1 0 0 0

上記①～⑧のt_0時のセルの状態でt_1時のセルの状態が決まる。

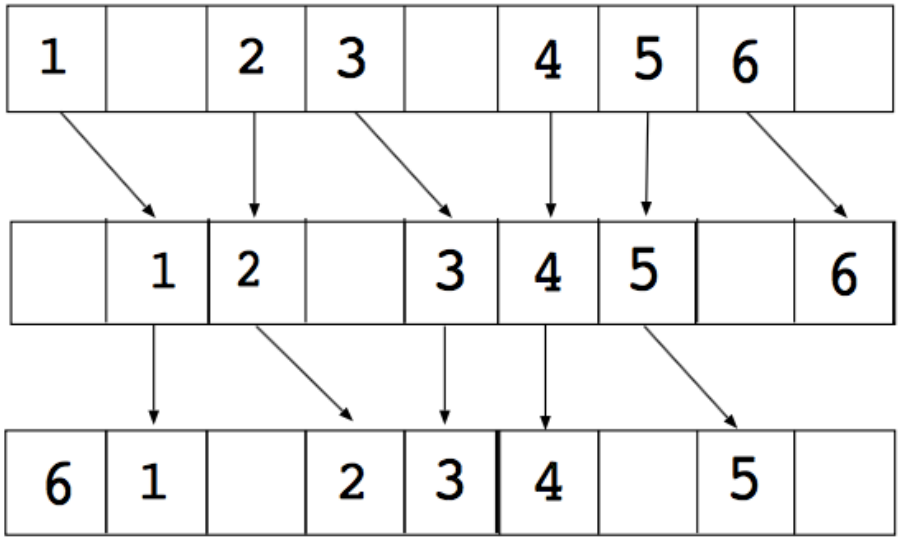
ここで①～⑧におけるt_1時の、中心のセルを並べると、「１０１１１０００」となる。これを２進数として考えると、１０進数の「１８４」に対応する。これが Rule184 の語源である。

４　　境界条件

Rule184 は周期境界条件を仮定したモデルであるが、他のモデルを考えるにあたって開放境界条件についても定義する必要がある。この章では、周期境界条件と開放境界条件について説明する。

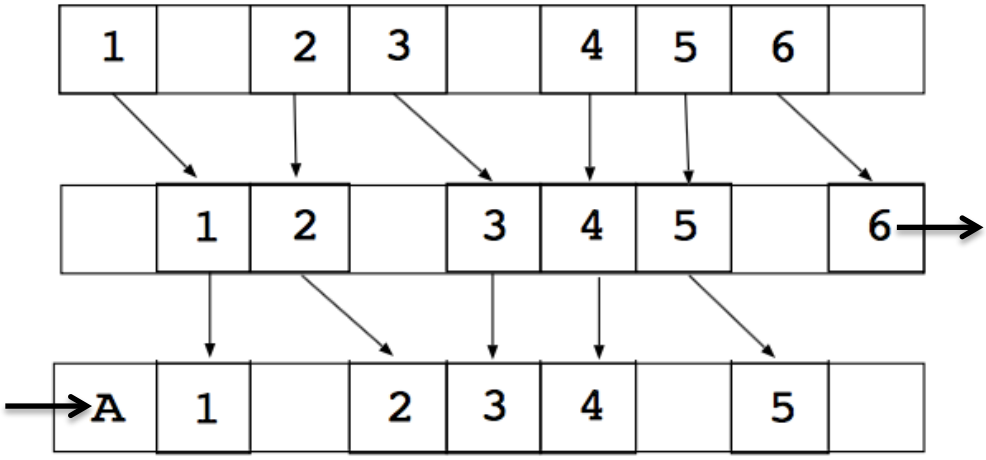
４.１　周期境界条件

この境界条件の特徴として、最後のセルと最初のセルが接続されていると考えることである。最後のセルにある車は、次のステップにおいて最初のセルを前方のセルと考え、進む場合は最初のセルに入る。



４.２　開放境界条件

この境界条件の特徴は、最初のセルと最後のセルは別に考えるということである。あるステップにおいて最初のセルに車が存在しない場合、次のステップで最初のセルに車が入ってくるというものである。最後のセルにおいては、あるステップにおいて最後のセルに車が存在する場合、次のステップで最後のセルに車がいなくなるというものである。



５　　スロースタートモデル

５.１　スロースタートモデルとは

Rule184 では単純すぎるという問題点が指摘されていた。それを解消するために、１度停止した車は、再び動き出すときにすぐには動けないということを考慮したモデルになる。

そのため、基本的な Rule184 に、新たなルールとして

・１度停止した車は１ステップ待って動き出す

を導入したモデルになる。

このモデルにおいては、周期境界条件と開放境界条件の２つで考える。

５.２　スロースタートモデル（周期境界条件）のシミュレーション

実際にスロースタートモデルをコンピューターによるシミュレーションを行ってみる。

下記に Processing によるプログラムと実行結果を写す。

「プログラム」

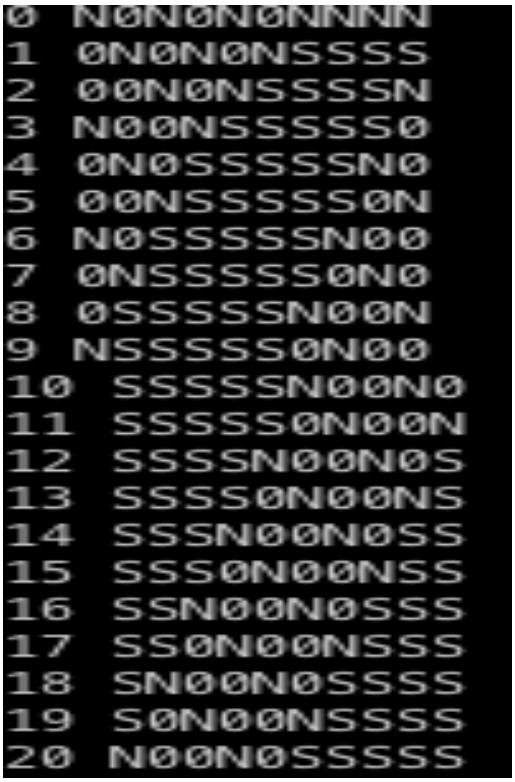
```
/*スロースタート　（周期境界条件）*/
/*
    0:車がないセル(空)
    1:N の車があるセル
    3:S の車があるセル
*/
/*
    「ルール」
(0 <= i <= C−2)
   ・初期配置はランダム。
   ・ i セルに N の車があり、i+1 セルに車がないとき、
      次のステップでは i セルは車がなく、i+1 セルに N の車がある。
   ・ i セルに N の車があり、i+1 セルに車があるとき,
      次のステップでは、i セルに S の車がある。(i+1 セルがどうなるかはここだけで決まらない)
   ・ i セルに S の車があり、i+1 セルに車がないとき、
      次のステップでは i セルに N の車がある。(i+1 セルは車がない)
   ・ i セルに S の車があり、i+1 セルに車があるとき、
      次のステップで i セルに S の車がある。(i+1 セルがどうなるかはここだけで決まらない)
*/

/*
セルの状態を配列 a で表す。
i 番目のセルに車がない状態を a[i] = 0 で表す。
i 番目のセルに車がある状態を a[i] = 1 ,a[i] = 3 で表す
*/

/*セルの数*/
int C=10;
/*車の台数*/
int M=7;
```

```
/*C 個の要素を持つ配列を用意*/
int n=1;
int a[] = new int[C];
int a2[] = new int [C];        /*a も a2 も 1 つの配列である。*/
char b[] = {'0','N','?','S'};

void setup() {
  /*全ての配列要素を 0*/
  for (int i=0; i<C; i++) {
    a[i]=0;
    a2[i]=0;
  }

  /*M 個の車を、ランダムにセルに配置*/
  int value;
  value=0;
  randomSeed(4);
  while (value<M) {
    int r=int(random(0, C));

    if (a[r]==0) {
      a[r]=1;
      value=value+1;
    }
  }
  print(n−1);
  print(" ");

  for (int i=0; i<C; i++) {
    print(b[a[i]]);
  }
```

```
  frameRate(60);                    /*移動の速さ*/
  println("");
}

void draw() {
  if (a[C-1] == 1 && a[0] == 0) {
    a2[C-1] = 0; a2[0] = 1;
  }
  if (a[C-1] == 1 && a[0] == 1) {
    a2[C-1] = 3;
  }
  if (a[C-1] == 1 && a[0] == 3) {
    a2[C-1] = 3;
  }
  if (a[C-1] == 3 && (a[0] == 1 || a[0] == 3)) {
    a2[C-1]=3;
  }
  if (a[C-1] == 3 && a[0] == 0) {
    a2[C-1] = 1; a2[0] = 0;
  }
  for (int i=0; i<C-1; i++) {
    //ある車が N であり前のセルが空のとき、次のステップで進み N になる
    //ある車が N であり前のセルが空でないとき、次のステップで進まず S になる
    if (a[i] == 1) {
      if (a[i+1] == 0) {
        a2[i] = 0; a2[i+1] = 1;
      }
      else {
        a2[i] = 3;
      }
    }
    else if (a[i] == 3) {
```

```
        //ある車が S であり前のセルが空のとき、次のステップで進まず N になる
        //ある車が S であり前のセルが空でないとき、次のステップで進まず S になる
          if (a[i+1] == 0) {
            a2[i] = 1;
          }
          else {
            a2[i] = 3;
          }
        }
        else {              // a[i] == 0;
          int p;
          if (i == 0) p = C−1; else p = i−1;
          if (a[p] == 1)
            a2[i] = 1;                          // やってある
          else
            a2[i] = 0;
        }
    }
    for (int i = 0; i < C; i++)
      a[i] = a2[i];
    print(n++); print(" ");

    for (int i=0; i<C; i++) {
      print(b[a[i]]);
    }
    println("");
}
```

「実行結果」



ステップ数２０までの実行結果を載せておく。

ここから、「渋滞クラスターが進行方向と反対側に移動していること」「渋滞クラスターが徐々に大きくなること」が見て取れる。3.1 の Rule184 に、１度停止した車は１ステップ待って動き出す、という条件を追加することでより実際の交通渋滞に近いシミュレーションを行うことができた。

ところが、ステップが増加しても車の台数が増えることはないため、初期配置の車の台数によって渋滞クラスターの大きさには上限が決まっている。そこ

で、より実際の交通に近いシミュレーションをするため、前方の交通状況に関わらず車が一定のペースで流入してくる開放境界条件で行うこととする。

５.３　スロースタートモデル（開放境界条件）のシミュレーション

実際にスロースタートモデル（開放境界条件）をコンピューターによるシミュレーションを行ってみる。

下記に Processing によるプログラムと実行結果を写す。

「プログラム」

```
/*スロースタート　開放境界条件*/

/*
    0:車がないセル(空)
    1:N の車があるセル
    3:S の車があるセル
*/
/*
    「ルール」
(0 <= i <= C−2)
```

・初期配置はランダム。

・ i セルに N の車があり、i+1 セルに車がないとき、
次のステップでは i セルは車がなく、i+1 セルに N の車がある。

・ i セルに N の車があり、i+1 セルに車があるとき,
次のステップでは、i セルに S の車がある。(i+1 セルがどうなるかはここだけで決まらない)

・ i セルに S の車があり、i+1 セルに車がないとき、
次のステップでは i セルに N の車がある。(i+1 セルは車がない)

・ i セルに S の車があり、i+1 セルに車があるとき、

次のステップで i セルに S の車がある。(i+1 セルがどうなるかはここだけで決まらない)
*/

/*
セルの状態を配列 a で表す。
i 番目のセルに車がない状態を a[i] = 0 で表す。
i 番目のセルに車がある状態を a[i] = 1 ,a[i] = 3 で表す
*/

/*セルの数*/
int C=10;
/*車の台数*/
int M=7;

/*C 個の要素を持つ配列を用意*/
int n=1;
int a[] = new int[C];
int a2[] = new int [C];　　/*a も a2 も 1 つの配列である。*/
char b[] = {'0','N','?','S'};

void setup() {
  /*全ての配列要素を 0*/
  for (int i=0; i<C; i++) {
    a[i]=0;
    a2[i]=0;
  }

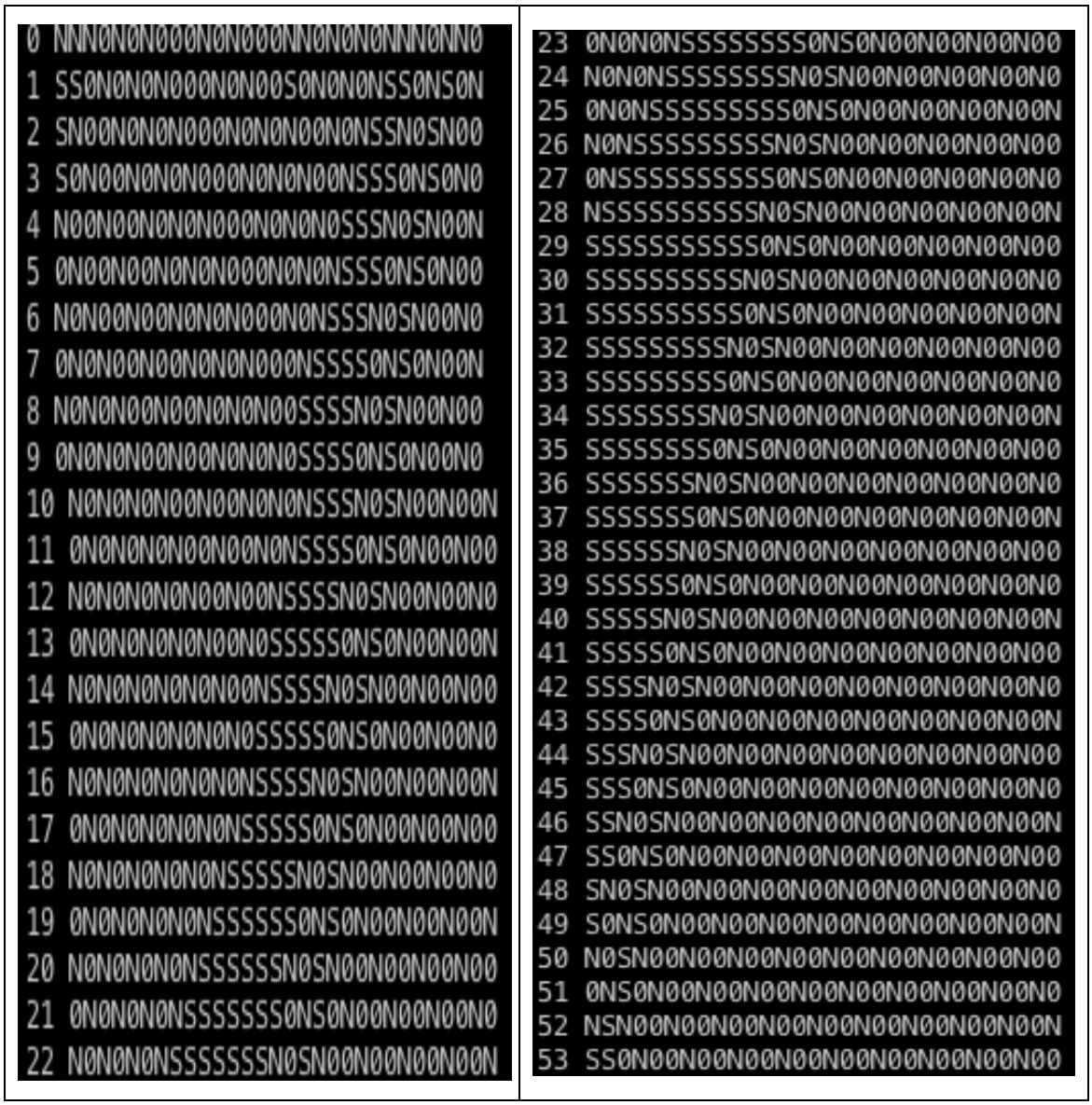
  /*M 個の車を、ランダムにセルに配置*/
  int value, M;
  value=0;
  M=16;
  randomSeed(3);

```
    while (value<M) {
      int r=int(random(0, C));

      if (a[r]==0) {
        a[r]=1;
        value=value+1;
      }
    }
    print(" ");
    print(n−1);
    print(" ");

    for (int i=0; i<C; i++) {
      print(b[a[i]]);
    }
    frameRate(60);     /*移動の速さ*/
    println("");
}

int prev(int i) {
    if (i > 0)
      return i−1;
    else
      return C−1;
}

int nect(int i) {
    if (i < C−1)
      return i+1;
    else
      return 0;
}
```

```
void draw() {
  for (int i=0; i<C−1; i++) {
    //ある車が N であり前のセルが空のとき、次のステップで進み N になる
    //ある車が N であり前のセルが空でないとき、次のステップで進まず S になる
    if (a[i] == 1) {
      if (a[i+1] == 0) {
        a2[i] = 0; a2[i+1] = 1;
      }
      else {
        a2[i] = 3;
      }
    }
    else if (a[i] == 3) {
     //ある車が S であり前のセルが空のとき、次のステップで進まず N になる
     //ある車が S であり前のセルが空でないとき、次のステップで進まず S になる
      if (a[i+1] == 0) {
        a2[i] = 1;
      }
      else {
        a2[i] = 3;
      }
    }
    else { // a[i] == 0;
      int p;
      if (i == 0) p = C−1; else p = i−1;
      if (a[p] == 1)
        a2[i] = 1; // やってある
      else
        a2[i] = 0;
    }
  }
```

```
  //最後のセルが N のとき、次で 0 になる
    if (a[C−1] == 1){
    a2[C−1] = 0;
  }
  //最初のセルが 0 のとき、次で N になる
  if(a[0] == 0){
    a2[0] = 1;
  }

  for (int i = 0; i < C; i++)
    a[i] = a2[i];
  print(" "); print(n++); print(" ");

  for (int i=0; i<C; i++) {
   print(b[a[i]]);
  }
  println("");
}
```

「実行結果」



ステップ数５３までの実行結果を載せておく。

ここから、「渋滞クラスターが進行方向と反対側に移動していること」「渋滞クラスターは徐々に大きくなること」が見て取れる。さらに開放境界条件にすることで、初期配置の車の台数に関わらず渋滞クラスターの成長を再現することができた。これは実際の交通渋滞においても見られる現象であり、一度渋滞が発生すると時間経過に比例して渋滞クラスターが大きくなるという現象の再現である。

６　まとめ

今回は、セルオートマトンモデルによって交通渋滞を再現するため、基本的なモデルの中でも特に Rule184 とスロースタートモデルによるシミュレーションを行った。スロースタートモデルにおいては、周期境界条件と開放境界条件の２つの条件でシミュレーションを行った開放境界条件においては、実際の交通渋滞でも見られる渋滞クラスターの成長を再現することができた。

今回の論文では「1 はじめに」で述べたように、コンピューターシミュレーションにより仮想渋滞を作り出すことで、現実に起こっている様々な条件下での渋滞を再現することを目標としたわけだが、そこで渋滞の発生原因をいくつか確認することができた。渋滞の発生原因として、３章の Rule184 で述べたように密度 ρ が関係していることがわかった。また５章のスロースタートモデルでは、より現実の交通渋滞を再現するため「1 度停止した車はすぐにはスタートできない」という模擬条件の下シミュレーションを行ったが、そこでは渋滞クラスターの大きさの成長が見て取れた。現実に近い模擬条件でシミュレーションを行うことで渋滞現象を再現することができた。

ところが、今回は基本的なモデルのセルオートマトンについてしかシミュレーションすることができなかった。そのため今後の課題としては、2.2 や「理系の備忘録」[3]に記されているような、その他のモデルやそれらモデルを組み合わせたシミュレーションを行うことでより現実に近い渋滞現象を再現することだ。

７　参考文献

[1]　西成活裕（２００６）「渋滞学（新潮選書）」新潮社

[2]　柳澤大地,西成活裕,渋滞学のセルオートマトンモデル,応用数理,
Vol.22 (1), pp. 2--14(2012)

[3]　理系の備忘録（最終閲覧日：２０１９年２月８日）
http://kenbell.hatenablog.com/entry/2017/02/12/172735