渋滞の基本図による分析

2019年度　明治大学総合数理学部現象数理学科
卒業研究レポート
冨岡 祐希

2020年2月28日

目 次

1　はじめに

週末やゴールデンウィーク、お正月休みなどの連休に車で出かけることがある。そのときに車の渋滞に巻き込まれることがある。車の渋滞でストレスがたまり、せっかくの休みなのに疲れることになる。私はこのような状況をできるだけ避けたい。そこで私は車の渋滞が起こるメカニズムを解明するために本研究に取り組んだ。本論文では渋滞をセルオートマトンモデルを用いてコンピューターシミュレーションで再現し、セルオートマトンモデルの基本図から渋滞が起こるメカニズムを解明していくことを目標にする。

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

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」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道路をセルに分割して、各セルごとに内部状態を離散的な時間の変化で表す数理モデルである。セルオートマトンモデルは特に複雑な対象を簡素化して考えるときによく用いられる。各セルの内部状態を車がある状態とない状態の２通りで表し、隣接するセルの状態によって次のステップでの動きが決まる。動き方は様々な条件を与えることで決め、現実の渋滞に近い条件にしていく。

2.2　セルオートマトンモデルの種類

セルオートマトンモデルには下記のようなモデルが存在する。

- Rule184
- ASEP
- Quick-Start モデル
- Slow-Start モデル

他にもモデルは存在するが、今回は主に Rule184、Quick-Start モデル、Slow-Start モデルを調べていく。

3　条件

今回セルオートマトンモデルは次の条件に各モデルごとの条件を加えて行う。

- 1 つのセルに入れる車は 1 台
- 車線は 1 車線で行う
- 周期境界条件で行う

周期境界条件とは最後のセルと最初のセルを繋げているという考え方である。最後のセルにいる車が前に進むと、最初のセルに移動する。図 1 では 8 のセルにいる車は進むとき 1 に進む

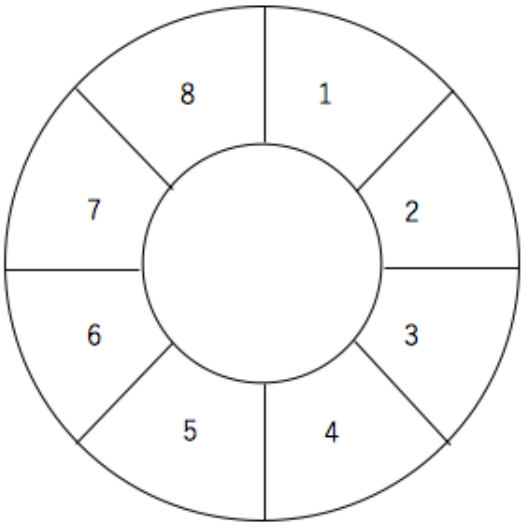

図 1: 周期境界条件

4　Rule184

4.1　Rule184 の条件

Rule184 はセルオートマトンモデルの中で最も単純なモデルである。Rule184 の条件を説明する。

- 周期境界条件

- 前のセルが空いている場合、次のステップで進む
- 前のセルが空いていない場合、次のステップでは進むことができない

図 2 は Rule184 の動き方である。

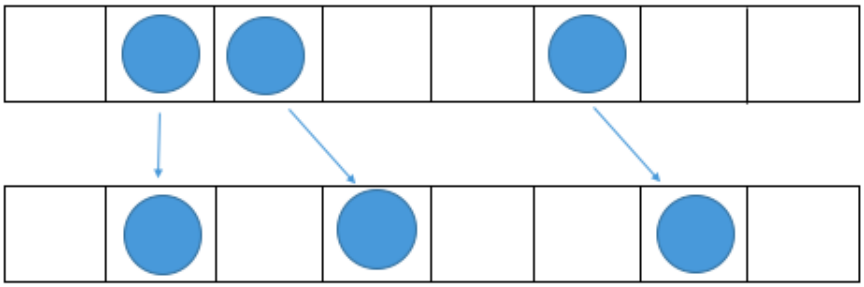

図 2: Rule184

この条件で Rule184 を C 言語でシミュレーションしてみる。

4.2 Rule184 のシミュレーション

下記は Rule184 の C 言語によるプログラム

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

int main(int argc, char **argv)
{
    int n=1;
    int C;//セルの数
    double M;//車の数
    double A=0;//動いた車の数
    double V;//平均速度
    if (argc != 3) {
      printf("セルの数");
      scanf("%d",&C);
      printf("車の数");
      scanf("%lf",&M);
```

```
  }
  else {
    C = atoi(argv[1]);
    M = atof(argv[2]);
  }
  int a[C];
  int a2[C];
  char b[4]={'0','N','?','S'};
  for(int i=0; i<C; i++){
      a[i]=0;
      a2[i]=0;
}

int value;
value=0;
srand(0);
while(value<M){
  int r=rand()%C; //ランダムに車を配置

  if(a[r]==0){//r のマスが空いていれば
    a[r]=1;//その r のマスに 1 台配置する。
    value=value+1;//M 台になるまで続ける
  }
}
printf("%3d ",n-1);
//printf("");

for(int i=0; i<C; i++){
  printf("%c",b[a[i]]);//マスに車があるかないかを 0 と N で表す。
}
printf("\n");

for(int x=0;x<100;x++){
  if(a[C-1] == 1 && a[0] == 0){
    a2[C-1] = 0; a2[0]=1;
    A=A+1;//追加
```

```
    }//C-1 セルに車があり、0 セルに車がない時、次のステップで C-1 セルに車
はなく、0 セルに車がある。
    for(int i=0; i<C-1; i++){
      if(a[i] == 1){//i セルに車がある時
        if(a[i+1] == 0){//i+1 のセルに車がないならば
          a2[i] = 0; a2[i+1] = 1;//次の時に i にある車が i+1 に進む
          A=A+1;//追加
        }
        else
          a2[i] = 1;//i+1 のセルに車があるなら動かない
      }
    }

    for(int i=0; i<C; i++)
      a[i]=a2[i];//次の状態のセルを現在の状態に変える。
    printf("%3d ", n++);//ステップ数
    printf("");//車があるかないかの結果を 0 と N で表している。

    V=A/M;
    printf("%2f",V);
    A=0;

    for(int i=0; i<C; i++){
      printf("%3c" , b[a[i]]);//車があるかないかの結果を 0 と N で表して
いる。
    }
    printf("\n");
  }
}
```

5 Quick-Start

5.1 Quick-Start の条件

Quick-Start は前のセルが空いていなくても 2 台先の動きを見て進むことができる。Quick-Start の条件を説明する。

- 周期境界条件
- 前のセルが空いている場合、次のステップで進む
- 前のセルが空いていない場合、次のステップでは進むことができない。ただし 2 台先が空いている場合のときのみ、前の車が進むのを見越して、進むことができる

図 3 は Quick-Start の動き方である。

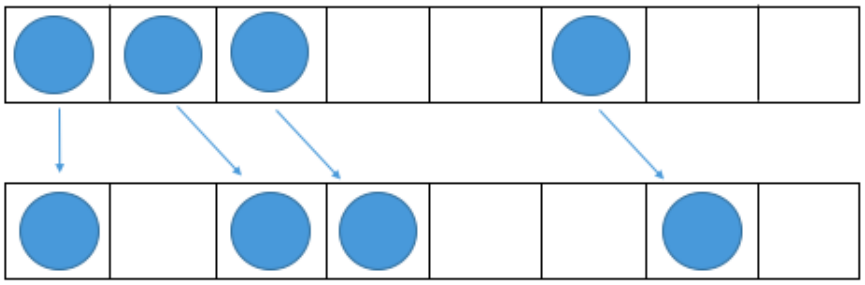

図 3: Quick-Start

この条件で Quick-Start を C 言語でシミュレーションしてみる。

5.2 Quick-Start のシミュレーション

下記は Quick-Start の C 言語によるプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <time.h>

int main(int argc, char **argv)
```

```
{
  int n;
  int C;//セルの数
  double M;//車の数
  double A=0;//動いた車の数
  double V;//平均速度
  double P;//密度 (M/C)
  double F;//流量 (P*V)
  unsigned int seed;
  printf("# セルの数");
  scanf("%d",&C);

  seed = (unsigned) time(NULL);
  printf("# seed=%ud\n", seed); // gnuplot では、# の後は注釈として扱
う。
  srand(seed);

  for(M=1; M<100; M++){
    for(int M2=0; M2<2; M2++){
      int a[C];
      int a2[C];
      char b[4]={'O','N','?','S'};
      for(int i=0; i<C; i++){
        a[i]=0;
        a2[i]=0;
      }
      int value;
      value=0;
      while(value<M){
        int r=rand()%C+1; //ランダムに車を配置
        if(a[r]==0){//r のマスが空いていれば
          a[r]=1;//その r のマスに 1 台配置する。
          value=value+1;//M 台になるまで続ける
        }
      }
      for(int n=0;n<100;n++){
```

```
        if(a[C-2] == 1 && a[C-1] == 1 && a[0] == 0){
          a2[C-1] = 0; a2[0]=1;
          A=A+1;
          a2[C-2] = 0; a2[C-1]=1;
          A=A+1;//追加
        }//C-1 セルに車があり、0 セルに車がない時、次のステップで C-1 セルに車はなく、0 セルに車がある。
        if(a[C-2] == 0 && a[C-1] == 1 && a[0] == 0){
          a2[C-1] = 0; a2[0]=1;
          A=A+1;//追加
        }

        if(a[C-1] == 1 && a[0] == 1 && a[1] == 0){
          a2[0] = 0; a2[1]=1;
          A=A+1;
          a2[C-1] = 0; a2[0]=1;
          A=A+1;//追加
        }

        if(a[C-1] == 0 && a[0] == 1 && a[1] == 0){
          a2[0] = 0; a2[1]=1;
          A=A+1;//追加
        }

        for(int i=0; i<C-2; i++){
          if(a[i] == 1 && a[i+1] == 1 && a[i+2] == 0){//i セルに車がある時
            a2[i+1] = 0; a2[i+2]=1;
            A=A+1;
            a2[i] = 0; a2[i+1] = 1;//次の時に i にある車が i+1 に進む
            A=A+1;//追加
          }
          if(a[i] == 0 && a[i+1] == 1 && a[i+2] == 0){//i セルに車がある時
            a2[i+1] = 0; a2[i+2]=1;
            A=A+1;//追加
```

```
          }
        }
          for(int i=0; i<C; i++)
            a[i]=a2[i];//次の状態のセルを現在の状態に変える。
          V=A/M;
          P=M/C;
          F=P*V;
          A=0;

      } // n
      printf("M=%f %.15f %.15f %.15f\n",M,V,P,F);
    } // M2
  }// M
}
```

6 Slow-Start

6.1 Slow-Start の条件

Slow-Start は 1 度止まった車はすぐに動けないので、前が空いていても 1 ステップ待ってから進む。Slow-Start の条件を説明する。

- 周期境界条件
- 前のセルが空いている場合、次のステップで進む
- 前のセルが空いていない場合、次のステップでは進むことができない
- 止まっている車の前のセルが空いた場合、１ステップ待ってから進む

図 4,5 は Slow-Start の動き方である。
この条件で Slow-Start を C 言語でシミュレーションしてみる。

6.2 Slow-Start のシミュレーション

下記は Slow-Start の C 言語によるプログラム

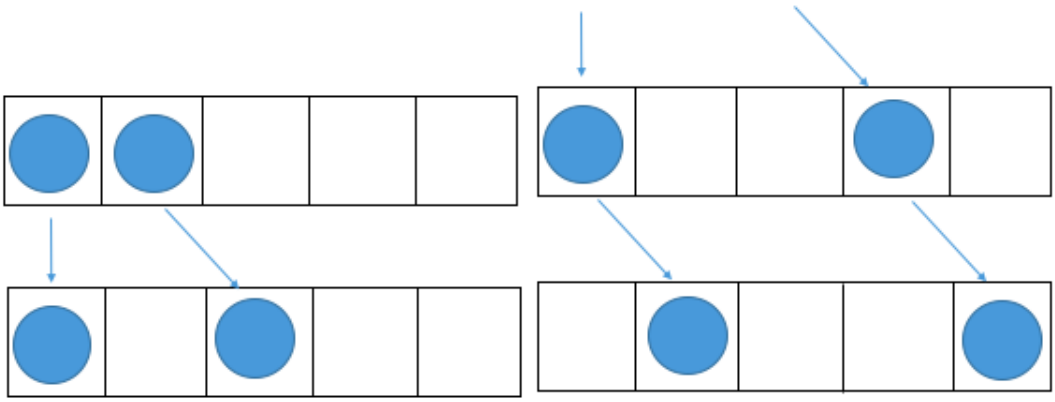

図 4: Slow-Start

図 5:

```
//スロースタート（周期境界条件）
/*
   0:車がないセル
   N:次のステップで動ける車があるセル
   S:スロースタートの車があるセル
*/

/*
   [ルール]
   (0<=i<=C-1)
   ・初期配置はランダム
   ・i セルが N かつ、i+1 セルが 0 のとき、
     次のステップで i セルは 0 になり、i+1 セルは N になる。
   ・i セルが N かつ、i+1 セルが N または S のとき、
     次のステップで i セルは S になる。
     （i+1 セルがどうなるかはここだけで決まらない）
   ・i セルが S かつ、i+1 セルが 0 のとき、
     次のステップで i セルは N になる。
   ・i セルが S かつ、i+1 セルが N または S のとき、
     次のステップで i セルは S のままになる。
     （i+1 セルがどうなるかはここだけで決まらない）
   ・i=C-1 のとき、i+1=0 とみなす。
   ・
*/
```

```
/*
      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り、次のステップで動ける状態を a[i]=1,
     次のステップで動けない状態を a[i]=3 で表す。
*/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>


#define C 10//セルの数
#define M 7//車の台数
#define D 100//ステップ数

int main()
{
 int n=1;
 int verbose = 1;
 int a[C][D];
 char b[4][D]={'0','N','?','S'};
 /*for(int i=0;i<C;i++){
   a[i][j]=0;
   }*/

 int value;
 value=0;
 srand(0);
 while(value<M){
   int r=rand()%C;//ランダムに車を配置
   if(a[r][0]==0){//r のセルが空いていれば
     a[r][0]=1;//その r のセルに 1 台配置する
     value=value+1;//7 台になるまで続ける
   }
```

```
 }
 printf("%3d",n-1);
 //printf("");

 for(int i=0;i<C;i++){
   for(int j=0;j<D;j++){//C-1 セルと 0 セルの関係
     printf("%3s",b[a[i][j]]);//セルに車がない状態を 0、次のステップで動
ける状態を N、スロースタートの状態を S で表す
      printf("\n");

     if(a[C-1][j]==1 && a[0][j]==0){//C-1 セルが N かつ、0 セルが 0 の時、
次のステップで C-1 セルは 0 になり、0 セルは N になる。
       a[C-1][j+1]=0; a[0][j+1]=1;
     }
     if(a[C-1][j]==1 && (a[0][j]==1 || a[0][j]==3)){//C-1 セルが N かつ、
0 セルが N または S の時、次のステップで C-1 セルは車は S になる。
       a[C-1][j+1]=3;
     }
     if(a[C-1][j]==3 && a[0][j]==0){//C-1 セルが S かつ、0 セルが 0 の時、
次のステップで C-1 セルは N になり、0 セルは 0 のままになる。
       a[C-1][j+1]=1; a[0][j+1]=0;
     }
     if(a[C-1][j]==3 && (a[0][j]==1 || a[0][j]==3)){//C-1 セルが S かつ、
0 セルが N または S の状態の時、次のステップで C-1 セルのは S になる。
       a[C-1][j+1]=3;
     }
   }
 }

     for(int i=0;i<C-1;i++){//0<=i<=C-2 の時
       //ある車が N かつ、前のセルが 0 のとき、次のステップで進み N になる
       //ある車が N かつ、前のセルが N または S のとき、次のステップで進まず
S になる
       for(int j=0;j<D;j++){
       if(a[i][j]==1){
         if(a[i+1][j]==0){
```

```
          a[i][j+1]=0; a[i+1][j+1]=1;
        }
        else{
            a[i][j+1]=3;
        }
      }
      else if(a[i][j]==3){
          //ある車がSかつ、前のセルが0のとき、次のステップで進まずNにな
る
          //ある車がSかつ、前のセルがNまたはSのとき、次のステップで進
まずSになる
        if(a[i+1][j]==0){
          a[i][j+1]=1;
        }
        else{
            a[i][j+1]=3;
        }
      }
      /*else{//a[i]==0
        int p;
        if(i==0){
          p=C-1;
        }
        else{
          p=i-1;
        }
        if(a[p]==1){
          a2[i]=1;
        }
        else{
          a2[i]=0;
        }
      }*/
    }
    }
    printf("%3d",n++);//ステップ数
```

```
      printf("");//車があるかないかの結果を 0 と N で表している

      for(int i=0;i<C;i++){
        for(int j=0;j<D;j++){
        printf("%3s",b[a[i][j]]);
        }
      }
      printf("\n");
}
```

7　基本図

7.1　基本図とは

渋滞が起こる原因を解明するために基本図というものを使っていく。基本図は交通の流れの分析において重要で基本的な役割を果たすため、基本図という名称がついている。基本図とは各モデルが定常状態のときのセル全体の密度と流量を求めて、プロットした図のことである。縦軸に流量、横軸に密度をとる。下記の図 6 は東名高速道路の静岡県焼津市付近における実際のデータです。

7.2　用語

基本図を描くためには密度、流量、平均速度が必要になる。また他にも基本図を理解するために大事な単語があるため、それらを説明する。

7.2.1　密度

密度は「渋滞学」ではその地点の付近 1km あたりに何台の車がいるかというものである。セルオートマトンモデルでは車の総数をセルの総数で割った値を密度としている。

$$\text{密度} = \rho$$

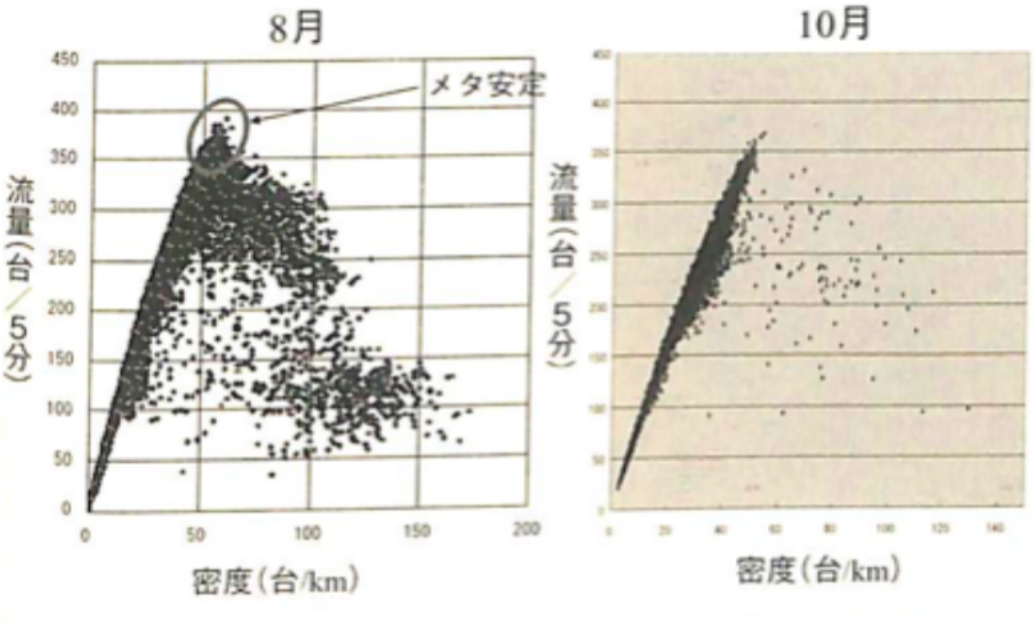

図 6: 基本図 (東名高速道路、静岡県焼津市付近)

$$\text{セルの総数} = S$$

$$\text{車の総数} = M$$

とすると密度は

$$\rho = M/S$$

という式で表すことができる。
図 7 の場合、車の総数=4、セルの総数=8 より密度=1/2 となる。

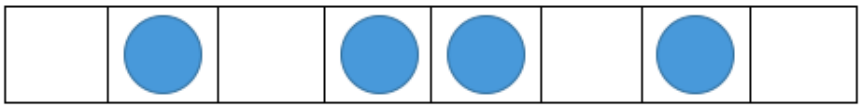

図 7: 密度

7.2.2　平均速度

セルオートマトンモデルではある時刻で動いた車の数を車の総数で割った値を平均速度としている。

$$\text{ある時刻で動いた車の数} = K$$

$$\text{平均速度} = u$$

とすると平均速度は

$$u = K/M$$

という式で表すことができる平均速度は時刻によって変わるが、時間が経つと平均速度が一定になる。この状態を定常状態といい、この時の速度を定常速度という。

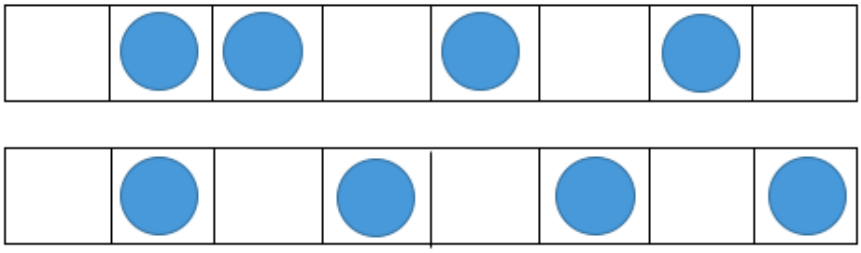

図 8: 平均速度

図 8 の場合、動いた車の数 3、車の総数 4 より平均速度=3/4 となる。

7.2.3　流量

流量は「渋滞学」ではその地点を 5 分間に通過する車の総台数というものである。

$$\text{流量} = Q$$

$$\text{流量} = \text{密度} \times \text{平均速度}$$

という式で表すことができるので流量は

$$Q = \rho * u = M/S * K/M = K/S$$

という式で表すことができる。つまり、セルオートマトンモデルではある時刻で動いた車の数をセルの総数で割った値が流量になる。

7.2.4 自由走行相・渋滞相

下記の図を見ると図の左側部分は右上がりの直線になっている。これを自由走行相という。図の右側は渋滞相という。この自由走行相と渋滞相からどの密度のときに自由走行相から渋滞相へ変化するかが読み取れる。

7.2.5 臨界密度

自由走行相から渋滞相へ変わるときの密度を臨界密度という。図 9 の場合密度が 1/2 のところで自由走行相から渋滞相へ変化しているため臨界密度は 1/2 になる。

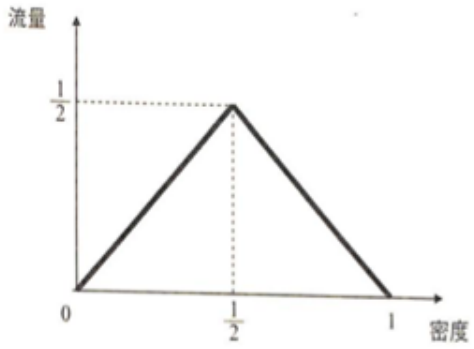

図 9: Rule184 理論図

7.2.6 準安定状態 (メタ安定)

「準安定状態」は、真の安定状態では無いが、大きな乱れが与えられない限り安定に存在できるような状態。準安定状態は小さな乱れに対しては安定であるが、大きな乱れが与えられると不安定になり、真の安定状態へ変化する。例えば、通行量の多い高速道路を時速 100km くらいで走行しているのにもかかわらず、車間距離が 10 メートルあるかないかという集団密集走行状態になっていることがある。

このような状態は、統計物理学でいる準安定状態である。本来は渋滞になってしまう密度で、この状態は一見安定に見えるが、実は不安定で、ちょっとしたことで渋滞になる。渋滞の研究者はこのことをメタ安定ということが多い。

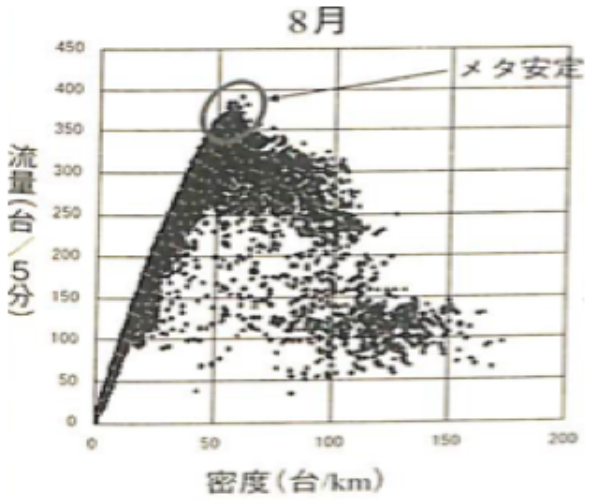

図 10: 基本図

8　各モデルの基本図

8.1　Rule184 基本図

Rule184 では前が空いていれば進めるため、車の台数がセルの総数の 1/2 以下までならどの車も止まらずに動けるため平均速度が 1 になる。流量は密度×平均速度だから、密度が 1/2 以下のとき

$$Q = \rho$$

密度が 1/2 より大きい時は動ける車の数は開いているセルの数になるから、動ける車の数は 1 − 密度で表すことができる。よって平均速度は

$$(1 - \rho)/\rho$$

になります。流量=密度×平均速度だから、

$$Q = \rho \times (1 - \rho)/\rho = (1 - \rho)$$

よって密度が 1/2 より大きいとき流量=1 − 密度になる。

$$Q = \begin{cases} \rho & (\rho \leq 1/2 \text{ のとき}) \\ 1-\rho & (\rho > 1/2 \text{ のとき}) \end{cases}$$

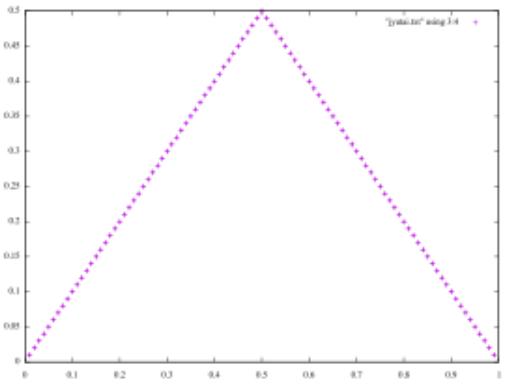

図 11: Rule184 シミュレーション

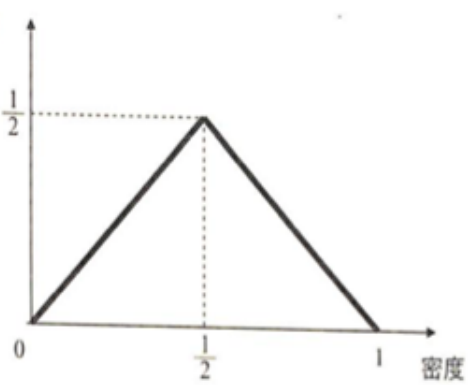

図 12: Rule184 基本図

左の図がシミュレーション結果の基本図、右の図が理論図である。この図を見ると、臨界密度が 1/2 で、基本的な渋滞相転移の様子が見れる。下記は Rule184 基本図の C 言語によるプログラム

```
/*
 *
 */

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <time.h>

int main(int argc, char **argv)
{
  int n;
  int C;//セルの数
  double M;//車の数
  double A=0;//動いた車の数
  double V;//平均速度
```

```
  double P;//密度(M/C)
  double F;//流量(P*V)
  unsigned int seed;
  printf("# セルの数");
  scanf("%d",&C);

  seed = (unsigned) time(NULL);
  printf("# seed=%ud\n", seed); // gnuplot では、# の後は注釈として扱う。
  srand(seed);

  for(M=1; M<100; M++){
    for(int M2=0; M2<2; M2++){
      int a[C];
      int a2[C];
      char b[4]={'0','N','?','S'};
      for(int i=0; i<C; i++){
    a[i]=0;
        a2[i]=0;
      }
       int value;
      value=0;
      while(value<M){
    int r=rand()%C+1; //ランダムに車を配置
    if(a[r]==0){//r のマスが空いていれば
      a[r]=1;//その r のマスに 1 台配置する。
      value=value+1;//M 台になるまで続ける
    }
      }
      for(int n=0;n<100;n++){
    if(a[C-1] == 1 && a[0] == 0){
      a2[C-1] = 0; a2[0]=1;
      A=A+1;//追加
    }//C-1 セルに車があり、0 セルに車がない時、次のステップで C-1 セルに車はなく、0 セルに車がある。
    for(int i=0; i<C-1; i++){
```

```
      if(a[i] == 1){//i セルに車がある時
        if(a[i+1] == 0){//i+1 のセルに車がないならば
          a2[i] = 0; a2[i+1] = 1;//次の時に i にある車が i+1 に進む
          A=A+1;//追加
        }
        else
          a2[i] = 1;//i+1 のセルに車があるなら動かない
      }
    }

    for(int i=0; i<C; i++)
      a[i]=a2[i];//次の状態のセルを現在の状態に変える。
    V=A/M;
    P=M/C;
    F=P*V;
    A=0;
      } // n
      printf("M=%f %.15f %.15f %.15f\n",M,V,P,F);

    } // M2

  }// M
}
```

8.2 Quick-Start 基本図

Quick-Start ではセルが 1 つ開いていると 2 台進むことができるため、密度が 2/3 以下のとき平均速度が 1 になるから、密度が 2/3 以下のとき

$$Q = \rho$$

となる。

この図を見ると、Rule184 では臨界密度が 1/2 だったが、Quick-Start では臨界密度が 2/3 になっている。このことにより渋滞の発生を遅らせることができることがわかる。しかし現実の基本図と見比べると、臨界密度の位置やメタ安定が出ていない。下記は Quick-Start 基本図の C 言語によるプログラム

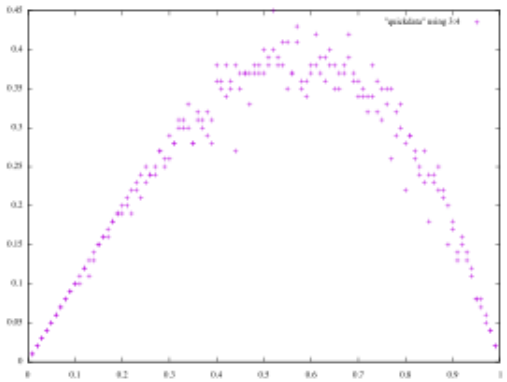

図 13: Quick-Start シミュレーション

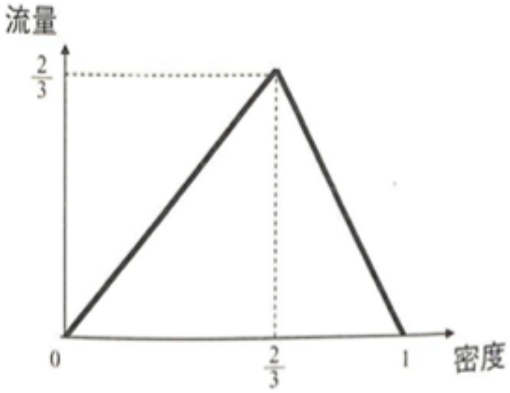

図 14: Quick-Start 基本図

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <time.h>

int main(int argc, char **argv)
{
  int n;
  int C;//セルの数
  double M;//車の数
  double A=0;//動いた車の数
  double V;//平均速度
  double P;//密度 (M/C)
  double F;//流量 (P*V)
  unsigned int seed;
  printf("# セルの数");
  scanf("%d",&C);

  seed = (unsigned) time(NULL);
  printf("# seed=%ud\n", seed); // gnuplot では、# の後は注釈として扱
う。
  srand(seed);

  for(M=1; M<100; M++){
```

```
    for(int M2=0; M2<2; M2++){
      int a[C];
      int a2[C];
      char b[4]={'0','N','?','S'};
      for(int i=0; i<C; i++){
    a[i]=0;
        a2[i]=0;
      }
      int value;
      value=0;
      while(value<M){
    int r=rand()%C+1; //ランダムに車を配置
    if(a[r]==0){//r のマスが空いていれば
      a[r]=1;//その r のマスに 1 台配置する。
      value=value+1;//M 台になるまで続ける
    }
      }
      for(int n=0;n<100;n++){
    if(a[C-2] == 1 && a[C-1] == 1 && a[0] == 0){
      a2[C-1] = 0; a2[0]=1;
      A=A+1;
      a2[C-2] = 0; a2[C-1]=1;
      A=A+1;//追加
    }//C-1 セルに車があり、0 セルに車がない時、次のステップで C-1 セルに車
はなく、0 セルに車がある。
    if(a[C-2] == 0 && a[C-1] == 1 && a[0] == 0){
      a2[C-1] = 0; a2[0]=1;
      A=A+1;//追加
    }

    if(a[C-1] == 1 && a[0] == 1 && a[1] == 0){
      a2[0] = 0; a2[1]=1;
      A=A+1;
      a2[C-1] = 0; a2[0]=1;
      A=A+1;//追加
    }
```

```
    if(a[C-1] == 0 && a[0] == 1 && a[1] == 0){
      a2[0] = 0; a2[1]=1;
      A=A+1;//追加
    }

    for(int i=0; i<C-2; i++){
      if(a[i] == 1 && a[i+1] == 1 && a[i+2] == 0){//i セルに車がある時
        a2[i+1] = 0; a2[i+2]=1;
        A=A+1;
        a2[i] = 0; a2[i+1] = 1;//次の時に i にある車が i+1 に進む
        A=A+1;//追加
      }
      if(a[i] == 0 && a[i+1] == 1 && a[i+2] == 0){//i セルに車がある時
        a2[i+1] = 0; a2[i+2]=1;
        A=A+1;//追加
      }
    }
      for(int i=0; i<C; i++)
        a[i]=a2[i];//次の状態のセルを現在の状態に変える。
      V=A/M;
      P=M/C;
      F=P*V;
      A=0;

      } // n
      printf("M=%f %.15f %.15f %.15f\n",M,V,P,F);
    } // M2
  }// M
}
```

8.3 Slow-Start 基本図

Slow-Start ではある時刻で動けなかった車はその前が空いていても 1 ステップ待ってから動くため、密度が 1/3 以下のとき平均速度が 1 になるから、密度が 1/3

以下のとき

$$Q = \rho$$

となる。

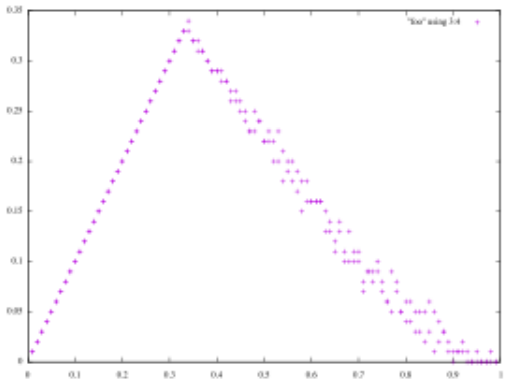

図 15: Slow-Start シミュレーション

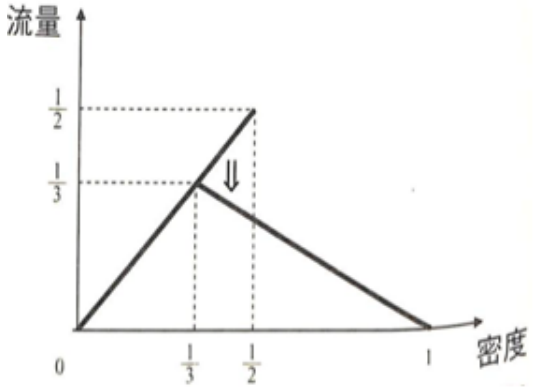

図 16: SlowStart 基本図

この図を見ると臨界密度は 1/3 でシミュレーションした図ではわずかにメタ安定が見れる。右の理論図のようにはっきりは出なかったが他のモデルよりは現実の基本図にかなり近くなった。下記は Slow-Start 基本図の C 言語によるプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <time.h>

int main(int argc, char **argv){
  int n;
  int C;//セルの数
  double M;//車の数
  double A=0;//動いた車の数
  double V;//平均速度
  double P;//密度(M/C)
  double F;//流量(P*V)
  unsigned int seed;
  printf("# セルの数");
  scanf("%d",&C);
```

```
  //seed = (unsigned) time(NULL);
  //printf("# seed=%ud\n", seed); // gnuplot では、# の後は注釈として
扱う。
  srand(seed);
  srand(77);

  for(M=1; M<100; M++){
    for(int M2=0; M2<2; M2++){
      int a[C];
      int a2[C];
      char b[4]={'0','N','?','S'};
      for(int i=0; i<C; i++){
    a[i]=0;
        a2[i]=0;
      }
      int value;
      value=0;
      while(value<M){
    int r=rand()%C+1; //ランダムに車を配置
    if(a[r]==0){//r のマスが空いていれば
      a[r]=1;//その r のマスに 1 台配置する。
      value=value+1;//M 台になるまで続ける
    }
      }
      for(int n=0;n<100;n++){
    if(a[C-1] == 1 && a[0] == 0){
      a2[C-1] = 0; a2[0]=1;
      A=A+1;//追加
    }//C-1 セルに車があり、0 セルに車がない時、次のステップで C-1 セルに車
はなく、0 セルに車がある。
    if(a[C-1]==1 && (a[0]==1 || a[0]==3)){//C-1 セルが N かつ、0 セルが
N または S の時、次のステップで C-1 セルは車は S になる。
      a2[C-1]=3;
    }
    if(a[C-1]==3 && a[0]==0){//C-1 セルが S かつ、0 セルが 0 の時、次のス
テップで C-1 セルは N になり、0 セルは 0 のままになる。
```

```
      a2[C-1]=1; a2[0]=0;
    }
    if(a[C-1]==3 && (a[0]==1 || a[0]==3)){//C-1 セルが S かつ、0 セルが
N または S の状態の時、次のステップで C-1 セルのは S になる。
      a2[C-1]=3;
    }


    for(int i=0; i<C-1; i++){
      if(a[i] == 1){//i セルに車がある時
        if(a[i+1] == 0){//i+1 のセルに車がないならば
          a2[i] = 0; a2[i+1] = 1;//次の時に i にある車が i+1 に進む
          A=A+1;//追加
        }
        else{
          a2[i] = 3;//i+1 のセルに車があるなら動かない
        }
      }
      else if(a[i]==3){
        //ある車が S かつ、前のセルが 0 のとき、次のステップで進まず N にな
る
        //ある車が S かつ、前のセルが N または S のとき、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;
        }
      }
    }

    for(int i=0; i<C; i++)
      a[i]=a2[i];//次の状態のセルを現在の状態に変える。
    V=A/M;//平均速度
    P=M/C;//密度
```

```
    F=P*V;//流量
    A=0;
      } // n
      printf("M=%f %.15f %.15f %.15f\n",M,V,P,F);
    } // M2
  }// M
}
```

9　まとめ

シミュレーションして、基本図を描いてわかったことは各セルオートマトンモデルに臨界密度が存在し、モデルによって異なることがわかった。また、３つのモデルの中では Slow-Start モデルが現実の基本図に近くなることがわかった。基本図から渋滞が起こるメカニズムを解明するという目標を掲げてやってきて、臨界密度やメタ安定などが渋滞の原因に関わっていることから、これらをより詳しく見ていくことが大事になってくる。

今回行ったシミュレーションでは車線を１つに限定して周期境界条件で行ったが、現実の状況と比べるとまだまだ条件が足りない。今後は車線を複数にしてみたり、周期境界条件ではなく開放境界条件で行ってみたりなど、より現実に近くなるような条件を加えたシミュレーションを行いたい。またセルオートマトンモデルも他のモデルを試したり、複数のモデルを組み合わせてシミュレーションを行えば、渋滞が起こるメカニズムを解明できそうだ。

参考文献

[1] 西成活裕　(2006)「渋滞学 (新潮選書)」新潮社

[2] 西成活裕, 渋滞のサイエンスとその解消法, 日本物理学会誌, Vol. 71, No. 3, 2016, pp 170-173 \https://www.jps.or.jp/books/gakkaishi/2016/03/71-03mijika.pdf

[3] 理系の備忘録　http://kenbell.hatenablog.com/entry/2017/02/12/172735

[4] 準安定状態 (ウィキペディア) https://ja.wikipedia.org/wiki/準安定状態