情報実験

第 3 期レポート Jul.7[Thu.] 2016

BP14068 Ryu Tazaki

1. 課題・結果

1.1 課題

完全列挙法(又は順次生成・比較法)以外に少なくとも 1 つの方法で巡回セールスマン問題を解き，比較検討する。

1.2 結果

本課題に対し，完全列挙法と Nearest Neighbor 法(実験 6)のプログラムを実行し，消費時間・結果として出された最短経路の距離の観点から試行と分析・考察を行う。

また都市数 N(以後，N とする)に関して，N=1 は移動せず，N=2 は 1 通りである事が明らかである。

1.2.1 消費時間

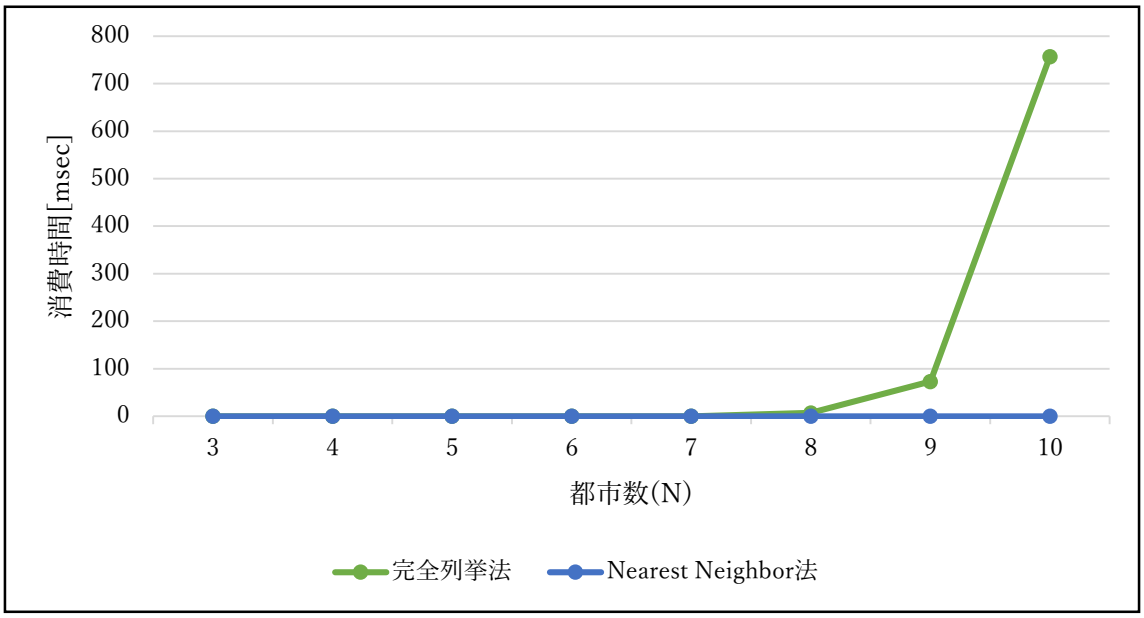

図 1 消費時間の比較[msec]

1.2.2 最短経路の距離

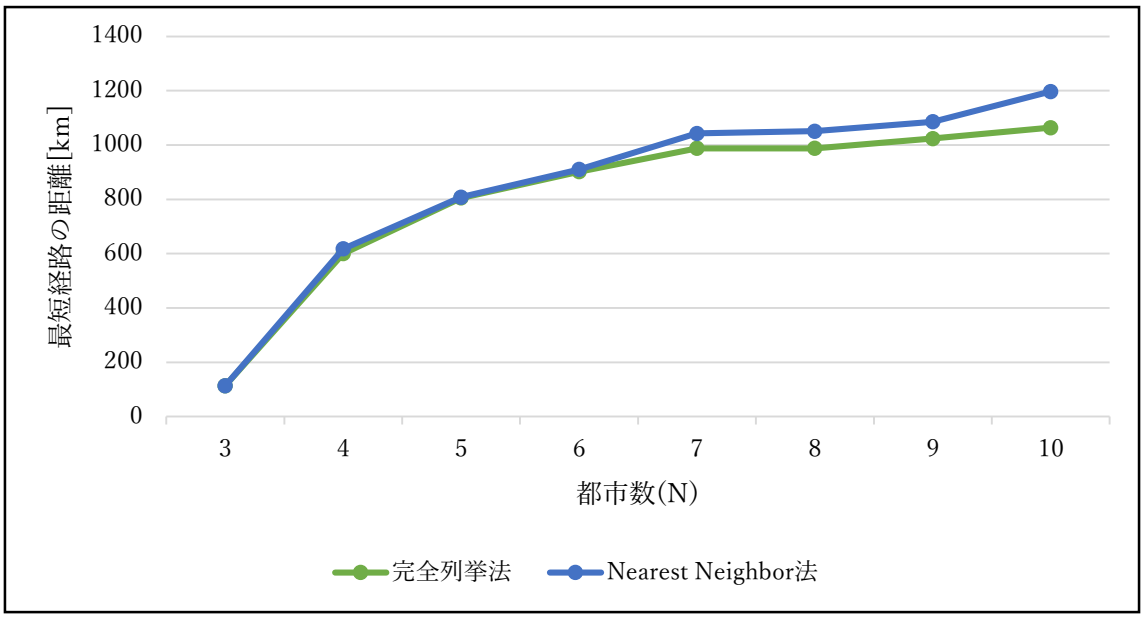

図 2 最短経路の距離の比較[km]

2. 分析・考察

2.1 消費時間

消費時間に関して，完全列挙法は，都市数の順列を用いて全ての組み合わせを列挙，その中から距離が最短である経路を探しだす方法であるが，都市の数が増える毎に探索すべき経路の数が爆発的に増加してしまう(組み合わせ的爆発)。一方 Nearest Neighbor 法は，目の前にある最短の最善な都市を選択し，辿っていない都市に対して繰り返し行い，完全列挙法に比べて簡単なアルゴリズムで高速である。それぞれの計算量を比較してみると，Order(N!) > m*Order(log N)であり，N=10 の時，N! = 3,628,800 であるので，図 1 の様になったと考えられる。また，Nearest Neighbor 法について，消費時間は N に依存しているか否かは不明である。

2.2 最短経路の距離

最短経路の距離に関して，完全列挙法は経路全通りを比べて結果を出しているので，正確に算出する事が出来る。一方 Nearest Neighbor 法は，高速である代わりに正確さが低く，算出した経路を辿っても最短でない。これは，近い場所を辿っているだけなので，辿っていない経路が存在し，図 2 の様になったと考えられる。

2.3 他

・N=11 以上に関しては計測していない。依って，2 種方法に於いて N は 3〜10 の間で計測している。11! (=39, 916,800), 12! (=479, 001, 600)…と爆発的に計算量が増加し，メモリのオーバーフローの回避の為に，この様にした。

3. ソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define NAME_MAX 256

//Quoted( http://goo.gl/yUOhFX )
#define pmttsize 10
int buffer[pmttsize];
bool used[pmttsize+1];
int p=0;

// Factorial -> n!
int factorial(int n) {
  int result = 1, k;

  for(k=1 ; k<=n ; k++) result *= k;
  return result;
}

// pmtt, Quoted( http://goo.gl/yUOhFX )
void prtpmtt(int n, int **cmp){
  int i;
  for(i=0; i<n; i++) cmp[p][i] = buffer[i];
  p++;
}

void pmtt(int n, int z, int **tst){
  int i;
  if (n == z) prtpmtt(n, tst);
  else{
    for(i=1; i<=n; i++){
      if(used[i]) continue;
```

```
      buffer[z] = i;
      used[i] = true;
      pmtt(n, z+1, tst);
      used[i] = false;
    }
  }
}

void pre_pmtt(int n, int **tst){
  pmtt(n, 0, tst);
}


// main
int main(int argc, char *argv[]){
  FILE *fp;
  char fn[NAME_MAX];

  int **mtrx, **cmp, N=0;
  int i, j;
  int n0, n1, n2, n3, n4, n5, n6, n7, n8, n9;
  int fctral, *fctralsum;
  int root1=0, root2=0, dstnc=0, checker=-1;

  if(argc != 2){
    fprintf(stderr, "Usage: %s graph_file¥n", argv[0]);
    exit(1);
  }
  strcpy(fn, argv[1]);
  if((fp = fopen(fn, "r")) == NULL){
    fprintf(stderr, "File Open Error: %s¥n", fn);
    exit(1);
  }

  fscanf(fp, "%d", &N);
```

```
  // Generate(mtrx)
  mtrx = (int**)malloc(sizeof(int*) * N);
  for(i=0 ; i<N ; i++){
    mtrx[i] = (int*)malloc(sizeof(int) * N);
    for(j=0 ; j<N ; j++) mtrx[i][j] = 0;
  }

  // Include
  i=0;

  switch (N) {;
  case 3:
    while(fscanf(fp, "%d %d %d", &n0, &n1, &n2) != EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      i++;
    }
    break;
  case 4:
    while(fscanf(fp, "%d %d %d %d", &n0, &n1, &n2, &n3) != EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      i++;
    }
    break;
  case 5:
    while(fscanf(fp, "%d %d %d %d %d", &n0, &n1, &n2, &n3, &n4) !=
EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      mtrx[i][4] = n4;
```

```
    i++;
  }
  break;

case 6:
  while(fscanf(fp, "%d %d %d %d %d %d",
             &n0, &n1, &n2, &n3, &n4, &n5) != EOF){
    mtrx[i][0] = n0;
    mtrx[i][1] = n1;
    mtrx[i][2] = n2;
    mtrx[i][3] = n3;
    mtrx[i][4] = n4;
    mtrx[i][5] = n5;
    i++;
  }
  break;

case 7:
  while(fscanf(fp, "%d %d %d %d %d %d %d",
             &n0, &n1, &n2, &n3, &n4, &n5, &n6) != EOF){
    mtrx[i][0] = n0;
    mtrx[i][1] = n1;
    mtrx[i][2] = n2;
    mtrx[i][3] = n3;
    mtrx[i][4] = n4;
    mtrx[i][5] = n5;
    mtrx[i][6] = n6;
    i++;
  }
  break;

case 8:
  while(fscanf(fp, "%d %d %d %d %d %d %d %d",
             &n0, &n1, &n2, &n3, &n4, &n5, &n6, &n7) != EOF){
    mtrx[i][0] = n0;
    mtrx[i][1] = n1;
```

```
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      mtrx[i][4] = n4;
      mtrx[i][5] = n5;
      mtrx[i][6] = n6;
      mtrx[i][7] = n7;
      i++;
    }
    break;

  case 9:
    while(fscanf(fp, "%d %d %d %d %d %d %d %d %d",
                &n0, &n1, &n2, &n3, &n4, &n5, &n6, &n7, &n8) != EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      mtrx[i][4] = n4;
      mtrx[i][5] = n5;
      mtrx[i][6] = n6;
      mtrx[i][7] = n7;
      mtrx[i][8] = n8;
      i++;
    }
    break;

  case 10:
    while(fscanf(fp, "%d %d %d %d %d %d %d %d %d %d",
                &n0, &n1, &n2, &n3, &n4, &n5, &n6, &n7, &n8, &n9) !=
EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      mtrx[i][4] = n4;
      mtrx[i][5] = n5;
```

```
    mtrx[i][6] = n6;
    mtrx[i][7] = n7;
    mtrx[i][8] = n8;
    mtrx[i][9] = n9;
    i++;
  }
  break;

default:
  fprintf(stderr, "File Open Error: Include Error!!¥n");
  break;
}

// Factorial
fctral = factorial(N);

// Display
printf("[Matrix of Distance]¥n");
for(i=0; i<N ; i++){
  for(j=0 ; j<N ; j++){
    printf("%d ", mtrx[i][j]);
  }
  printf("¥n");
}
printf("Vertex = %d, Factreal(N) = %d.", N, fctral);
printf("¥n");

// Generate(cmp)
cmp = (int**)malloc(sizeof(int*) * fctral);
for(i=0 ; i<fctral ; i++){
  cmp[i] = (int*)malloc(sizeof(int) * N);
  for(j=0 ; j<N ; j++) cmp[i][j] = 0;
}

// Pmtt
pre_pmtt(N, cmp);
```

```
// Generate(fctralsum)
fctralsum = (int*)malloc(sizeof(int*) * fctral);
for(i=0 ; i<fctral ; i++) fctralsum[i] = 0;

// Rooting
for(i=0 ; i<fctral ; i++){

  for(j=0 ; j<N-1 ; j++){ /* Consideration: Number of array. */
    root1 = cmp[i][j];   /* Rooting Start */
    root2 = cmp[i][j+1]; /* Rooting Next Phase */
    fctralsum[i] = fctralsum[i] + mtrx[root1-1][root2-1];
  }
  fctralsum[i] = fctralsum[i] + mtrx[root2-1][cmp[i][0]-1];
}

dstnc = fctralsum[0];

for(i=0 ; i<fctral ; i++){
  if(dstnc >= fctralsum[i]){ /* Change */
    checker = i;
    dstnc = fctralsum[i];
  }
}

// Display
printf("¥n[Result]¥n");
printf("Minimum Distance = %d,¥n", dstnc);
printf("Rooting: ");
for(i=0 ; i<N ; i++) printf("%d ", cmp[checker][i]);
printf("%d ¥n¥n", cmp[checker][0]);

free(fctralsum);
free(mtrx);
free(cmp);
return 0;
```

```
}
```

図 5 完全列挙法のプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define NAME_MAX 256

// Nearest Neighbor
void nn(int tmp, int **mtrx, int **mtrxcpy, int N, int dstnc){
  /* nn(tmp, mtrx, N, 0); */
  int i, focus, tmp_dstnc=1000;
  int itl=0;

  printf("%d", tmp);

  for(i=0 ; i<N ; i++){
    if(mtrx[tmp][i] !=0 &&
       mtrx[tmp][i] < tmp_dstnc){
      tmp_dstnc = mtrx[tmp][i];
      focus = i;
    }
  }

  if(tmp_dstnc == 1000){
    dstnc += mtrxcpy[tmp][0];
    printf(" %d, ¥n", itl);
    printf("Minimum Distance = %d¥n", dstnc);
    return;
  }

  dstnc += tmp_dstnc;
  for(i=0 ; i<N ; i++) mtrx[i][tmp]=0;
  tmp = focus;
```

```
  printf(" ");
  nn(tmp, mtrx, mtrxcpy, N, dstnc);
}


// main
int main(int argc, char *argv[]){
  FILE *fp;
  char fn[NAME_MAX];

  int **mtrx, **mtrxcpy, N=0;
  int i, j;
  int n0, n1, n2, n3, n4, n5, n6, n7, n8, n9;
  // int fctral, *fctralsum;
  // int root1=0, root2=0;
  int tmp=0/*, checker=-1*/;

  if(argc != 2){
    fprintf(stderr, "Usage: %s graph_file¥n", argv[0]);
    exit(1);
  }
  strcpy(fn, argv[1]);
  if((fp = fopen(fn, "r")) == NULL){
    fprintf(stderr, "File Open Error: %s¥n", fn);
    exit(1);
  }

  fscanf(fp, "%d", &N);

  // Generate(mtrx)
  mtrx = (int**)malloc(sizeof(int*) * N);
  for(i=0 ; i<N ; i++){
    mtrx[i] = (int*)malloc(sizeof(int) * N);
    for(j=0 ; j<N ; j++) mtrx[i][j] = 0;
  }
```

```
  // Generate(mtrxcpy)
  mtrxcpy = (int**)malloc(sizeof(int*) * N);
  for(i=0 ; i<N ; i++){
    mtrxcpy[i] = (int*)malloc(sizeof(int) * N);
    for(j=0 ; j<N ; j++) mtrxcpy[i][j] = 0;
  }

  // Include
  i=0;

  switch (N) {;
  case 3:
    while(fscanf(fp, "%d %d %d", &n0, &n1, &n2) != EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      i++;
    }
    break;
  case 4:
    while(fscanf(fp, "%d %d %d %d", &n0, &n1, &n2, &n3) != EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      i++;
    }
    break;
  case 5:
    while(fscanf(fp, "%d %d %d %d %d", &n0, &n1, &n2, &n3, &n4) !=
EOF){
      mtrx[i][0] = n0;
      mtrx[i][1] = n1;
      mtrx[i][2] = n2;
      mtrx[i][3] = n3;
      mtrx[i][4] = n4;
```

```
    i++;
  }
  break;

case 6:
  while(fscanf(fp, "%d %d %d %d %d %d",
             &n0, &n1, &n2, &n3, &n4, &n5) != EOF){
    mtrx[i][0] = n0;
    mtrx[i][1] = n1;
    mtrx[i][2] = n2;
    mtrx[i][3] = n3;
    mtrx[i][4] = n4;
    mtrx[i][5] = n5;
    i++;
  }
  break;

case 7:
  while(fscanf(fp, "%d %d %d %d %d %d %d",
             &n0, &n1, &n2, &n3, &n4, &n5, &n6) != EOF){
    mtrx[i][0] = n0;
    mtrx[i][1] = n1;
    mtrx[i][2] = n2;
    mtrx[i][3] = n3;
    mtrx[i][4] = n4;
    mtrx[i][5] = n5;
    mtrx[i][6] = n6;
    i++;
  }
  break;

case 8:
  while(fscanf(fp, "%d %d %d %d %d %d %d %d",
             &n0, &n1, &n2, &n3, &n4, &n5, &n6, &n7) != EOF){
    mtrx[i][0] = n0;
    mtrx[i][1] = n1;
```

```
     mtrx[i][2] = n2;
     mtrx[i][3] = n3;
     mtrx[i][4] = n4;
     mtrx[i][5] = n5;
     mtrx[i][6] = n6;
     mtrx[i][7] = n7;
     i++;
   }
   break;

  case 9:
    while(fscanf(fp, "%d %d %d %d %d %d %d %d %d",
                &n0, &n1, &n2, &n3, &n4, &n5, &n6, &n7, &n8) !=
EOF){
     mtrx[i][0] = n0;
     mtrx[i][1] = n1;
     mtrx[i][2] = n2;
     mtrx[i][3] = n3;
     mtrx[i][4] = n4;
     mtrx[i][5] = n5;
     mtrx[i][6] = n6;
     mtrx[i][7] = n7;
     mtrx[i][8] = n8;
     i++;
   }
   break;

  case 10:
    while(fscanf(fp, "%d %d %d %d %d %d %d %d %d %d",
                &n0, &n1, &n2, &n3, &n4, &n5, &n6, &n7, &n8,
&n9) != EOF){
     mtrx[i][0] = n0;
     mtrx[i][1] = n1;
     mtrx[i][2] = n2;
     mtrx[i][3] = n3;
     mtrx[i][4] = n4;
```

```
    mtrx[i][5] = n5;
    mtrx[i][6] = n6;
    mtrx[i][7] = n7;
    mtrx[i][8] = n8;
    mtrx[i][9] = n9;
    i++;
  }
  break;

default:
  fprintf(stderr, "File Open Error: Include Error!!¥n");
  break;
}

// Include(mtrxcpy)
for(i=0 ; i<N ; i++){
  for(j=0 ; j<N ; j++)
    mtrxcpy[i][j] = mtrx[i][j];
}

// Display
printf("[Matrix of Distance]¥n");
for(i=0; i<N ; i++){
  for(j=0 ; j<N ; j++){
    printf("%d ", mtrx[i][j]);
  }
  printf("¥n");
}
printf("Vertex = %d.", N);
printf("¥n");

// Display
printf("¥n[Result]¥n");

// Nearest Neighbor
printf("Rooting: ");
```

```
  nn(tmp, mtrx, mtrxcpy, N, 0);

  //  printf("Minimum Distance = %d,¥n", dstnc);
  printf("¥n");

  //  free(fctralsum);
  free(mtrx);
  free(mtrxcpy);
  //  free(cmp);
  return 0;
}
```

図 6 Nearest Neighbor 法のプログラム

4. ソースコード解説

4.1 共通事項

・N と距離情報に関して，「5. 参考サイトリスト[1]」に記載している行列を用いた。
・プログラム前半で使われている switch 文に関して，3*3 行列から 10*10 行列のプログラムを書き換えずとも読み込む事が出来る様にしたものである。
・2 次元配列”mtrx”に関して，上記の switch 文を通って読み込んだ行列の内容をそのまま格納する為のものである。
・変数”dstnc”に関して，最終的な結果として出された最短経路の距離が代入される。

4.2 完全列挙法のプログラム

・関数”factreal”に関して，N!(階乗)を計算する関数である。
・関数”prtpmtt”, “pre_pmtt”, “pmtt”に関して，順列を求める関数である。尚，「5. 参考文献・リストサイト[2]」より引用し，作成した。

4.3 Nearest Neighbor 法のプログラム

・比較的簡単なアルゴリズムなので，関数”nn”内部で全て完結している。
・2 次元配列”mtrxcpy”に関して，計算上この配列が無いと，一番最後に訪問した都市から最初の都市に戻る際の距離が”dstnc”に足されないので，作成したものである。2 次元配列”mtrx”と同じ大きさで作成された後，”mtrx”同様全て 0 で初期化する。既に距離が読み込まれている”mtrx”との比較を行い，差分のみを代入する事で，距離の算出に成功している。

5. 参考文献・サイトリスト

[1]都道府県庁間の距離
http://www.gsi.go.jp/common/000137532.pdf

以上を参照し距離グラフを作成。千葉県庁から 9 つの都県庁(東京，神奈川，新潟，富山，石川，福井，山梨，長野，岐阜)への距離，小数第 1 桁を四捨五入。

```
0 40 47
40 0 27
47 27 0
```

図 7 距離グラフ(N=3 の場合)

```
0 40 47 273 288 334 356 141 210 309
40 0 27 253 249 293 316 102 173 271
47 27 0 278 259 300 317 100 188 265
273 253 278 0 209 258 322 251 158 436
288 249 259 209 0 53 113 167 87 151
334 293 300 258 53 0 69 203 139 134
356 316 317 322 113 69 0 217 188 87
141 102 100 251 167 203 217 0 115 170
210 173 188 158 87 139 188 115 0 192
309 271 265 346 151 134 87 170 192 0
```

図 8 距離グラフ(N=10 の場合)

[2]お気楽Ｃ言語プログラミング超入門(問題 7)
http://www.geocities.jp/m_hiroi/linux/clang04.html

[3] 「巡回セールスマン問題」を解くアルゴリズムを可視化したムービー
http://gigazine.net/news/20160512-traveling-salesman-problem-visualization/

[4]2016 年度情報実験 I 第 2 期レジュメ