情報実験I

第４期レポート

BP14003 秋山和哉

2018

1. 課題・結果
   1. 課題

完全列挙法 以外に少なくとも２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. 結果

本課題に対し、完全列挙法とNearestNeighbor法とNearestAddition法のプログラムを実行し、消費時間の結果から施行と分析・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通りであることが明らかである。

* 1. 消費時間のグラフ

1. 分析・考察
   1. 消費時間

消費時間に関してだが、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出す方法であるが、都市の数N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のである。これを組み合わせ的爆発という。一方NearsestNeighbor法は目の前にある最短の最善な都市を選択し、たどっていない都市に対して繰り返し行い、完全列挙法と比べて簡単なアルゴリズムで高速である。

NearsetAddirion法は小さな点をどんどん取り入れていく探索方法であり、完全列挙法より高速であるが、NearestNeighbor法よりは低速という結果となった。NearestNeighbor法、NearestAddition法２つの方法とも、都市数Nが1500、2000あたりから消費時間に差がつき始めたため、Nがとても大きい数になるとグラフから読み取ると、比例して増加することが読み取れる。S

以下、作成したプログラム

完全列挙法

#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;

}

NearestNeighbor法。

#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;

}

◎NearestAddition法

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/time.h>

#define DIST\_MAX 1.0e100

//構造体定義

struct node{

int number;

struct node \*parent;

struct node \*child;

}typedef Node;

Node \*listhead = NULL;

Node \*listtail = NULL;

// 時刻を計測して double値として返す関数

double gettimeofday\_sec() {

struct timeval tv;

gettimeofday( &tv, NULL );

return tv.tv\_sec + tv.tv\_usec \* 1e-6;

}

//閉路を巡回して、経路とその長さを表示する関数

void tour(Node \*p,double \*\*dist){

double distance = 0.0;

do{

printf("->%d",p->number);

distance += dist[p->number][p->child->number];

p = p->child;

}while(p != listhead);

printf("->%d\n",listhead->number);

printf("最短閉路長:%lf(NearestAddition)\n",distance);

}

//スタートする点を設定する関数

Node \* setStart(int s){

Node \*p;

p = (Node \*)malloc(sizeof(Node));

p->number = s;

p->parent = NULL;

p->child = NULL;

listhead = listtail = p;

return p;

}

void nearestSearch1(int N,int s,int \*T,Node \*p,double \*\*dist){

int i,j;

int v;

double distMin = DIST\_MAX;

Node \*q;

q = (Node \*)malloc(sizeof(Node));

q->number = 0;

q->parent = NULL;

q->child = NULL;

for(i=0;i<N;i++)

if(dist[s][i] < distMin && s != i){

distMin = dist[s][i];

v = i;

}

T[v] = 1;

q->number = v;

//双方向に接続して閉路とする

q->parent = q->child = p;

p->parent = p->child = q;

}

void nearestSearch(int N,int \*T,int nodeCount,Node \*p,double \*\*dist){

int i,j;

double distMin = DIST\_MAX;

Node \*q;

Node \*tmp;

q = (Node \*)malloc(sizeof(Node));

q->parent = NULL;

q->child = NULL;

for(j=0;j<nodeCount;j++){ //閉路に含まれる点を巡回していく

for(i=0;i<N;i++){ //全ての点の中で

if(T[i] == 0){ //まだ訪問していない点

if(distMin > dist[p->number][i]){

distMin = dist[p->number][i];

q->number = i;

q->parent = p->parent;

tmp = p;

}

}

}

p = p->child;

}

p = tmp;

T[q->number] = 1;

p->parent->child = q;

p->parent = q;

q->child = p;

}

void nearestAddition(int N,int s,double \*\*dist){

int i;

int \*T; //部分閉路

int nodeCount = 2; //訪問した都市のカウンタ

Node \*p;

Node \*tmp;

//配列を動的に確保

T = (int \*)malloc(sizeof(int) \* N);

for(i=0;i<N;i++)

T[i] = 0;

//スタートを設定

p = setStart(s);

T[s] = 1;

//次の都市を決定

nearestSearch1(N,s,T,p,dist);

//3都市目以降の都市を決定

while(nodeCount < N){

nearestSearch(N,T,nodeCount,listhead,dist);

nodeCount++;

}

tour(listhead,dist);

//領域を解放

p = listhead;

for(i=0;i<N;i++){

tmp = p->child;

free(p);

p = tmp;

}

}

int main(int argc,char \*argv[]){

int i,j,a,b,s,repeat;

double d,t1,t2;

int N; //点の次数

double \*\*dist; //距離行列

FILE \*fp;

//実行時引数のチェック

if(argc != 3){

fprintf(stderr,"Useage:%s data repeat\_number\n",argv[0]);

exit(1);

}

//ファイルオープン

fp = fopen(argv[1],"r");

if(fp == NULL){

fprintf(stderr,"File open error!\n");

exit(1);

}

//繰り返し回数

repeat = atoi(argv[2]);

//点の次数を読み取る

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

//配列を動的に確保

dist = (double \*\*)malloc(sizeof(double \*) \* N);

for(i=0;i<N;i++)

dist[i] = (double \*)malloc(sizeof(double) \* N);

//配列を初期化

for(i=0;i<N;i++)

for(j=0;j<N;j++)

dist[i][j] = 0;

//距離行列を作成

while(fscanf(fp,"%d %d %lf",&a,&b,&d) != EOF){

dist[a][b] = dist[b][a] = d;

}

//ファイルクローズ

fclose(fp);

// 時刻計測(t1)

t1 = gettimeofday\_sec();

for(i=0;i<repeat;i++){

s = i % N;

//Nearest Addtion法を実行

nearestAddition(N,s,dist);

}

// 時刻計測(t2)

t2 = gettimeofday\_sec();

fprintf( stdout, "かかった時間(平均): %.10f\n", (t2-t1)/repeat );

//領域を解放

for(i=0;i<N;i++)

free(dist[i]);

free(dist);

return 0;

}