人工知能Ⅱ
課題レポート

提出者：5CS（11）稲川拓海

提出日：2017 年 10 月 12 日（木）

提出先：情報工学科　清水先生

1 アルゴリズムの説明

A*アルゴリズム

現在の状態からゴール状態までの距離を「マンハッタン距離」として求め，それぞれの手のマンハッタン距離の総和をコストとし，これが最小となる手を次の手として選択していくアルゴリズムである。このアルゴリズムを「8パズル問題」に用いた時のフローチャートを図1に示す。

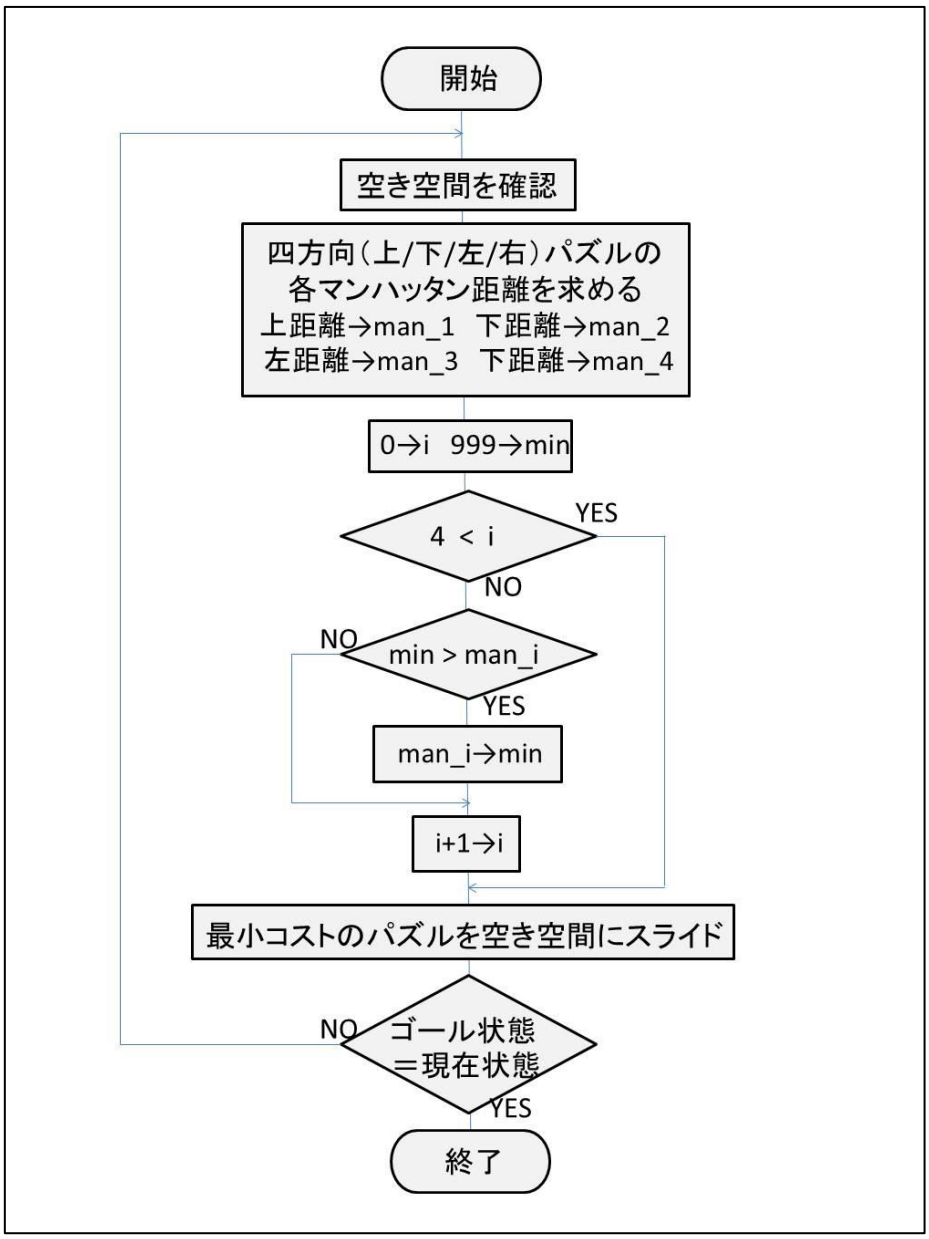

図1：A*アルゴリズムを用いた8パズル問題のフローチャート

2 ソースコード

作成したプログラムのソースコードを図２に示す。

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

main(){

	//変数定義
	int resul[3][3] = { { 1, 2, 3 },
	                    { 4, 5, 6 },
	                    { 7, 8, 0 } };

	int first[3][3] = { { 8, 1, 5 },
	                    { 2, 0, 4 },
	                    { 6, 3, 7 } };

	int manh[10] = { 1, 2, 2, 2, 2, 3, 2, 4 }; //マンハッタン距離
	int kin[4];
	int min = 0; //最短コスト
	int min_pazl = 0; //最小コストの手
	int ireko0_1 = 0, ireko0_2 = 0; //空いている空間座標を記憶
	int ireko1 = 0, ireko2 = 0;//その他の空間座標を記憶

	int change1,change2; //パズル入れ替え時の元数値の座標を保持

	int f = 0;
	int g = 0; //入れ替え回数カウント

	int i, j, k;
	int flug = 0; //終了フラグ
	int end = 0; //強制終了用
	int manh_i, manh_j;
	int kin1_1, kin1_2, kin2_1, kin2_2, kin3_1, kin3_2, kin4_1, kin4_2;//空いている空
間に近接する座標を記憶
```

```
int manh_1 = 999, manh_2 = 999, manh_3 = 999, manh_4 = 999;
int mang_1 = 999, mang_2 = 999, mang_3 = 999, mang_4 = 999;

FILE *file1,*file2;

//////////////////////////////////////////
////A*アルゴリズムを用いた「8パズル問題」///
//////////////////////////////////////////

//=======================================================

fopen_s(&file1, "パズル状態.txt", "w"); //ファイルを書き込み専用に開く
fopen_s(&file2, "発見的関数.txt", "w");

do{

//配列firstより"0"を探索する
for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
                if (first[i][j] == 0){
                        ireko0_1 = i;
                        ireko0_2 = j;
                        i = 999; //多重ループ脱出用
                        j = 999;
                }
        }
}

//0の位置に四方で近接する座標を求める
kin1_1 = ireko0_1 - 1; //上で近接する座標を記録
kin1_2 = ireko0_2;

kin2_1 = ireko0_1 + 1; //下で近接する座標を記録
kin2_2 = ireko0_2;

kin3_1 = ireko0_1;     //左で近接する座標を記録
```

```
        kin3_2 = ireko0_2 - 1;

        kin4_1 = ireko0_1;      //右で近接する座標を記録
        kin4_2 = ireko0_2 + 1;



        //0の位置に四方で近接する値を記憶
        kin[0] = first[kin1_1][kin1_2];
        kin[1] = first[kin2_1][kin2_2];
        kin[2] = first[kin3_1][kin3_2];
        kin[3] = first[kin4_1][kin4_2];

        //=====================================================

        //=====================================================
        //=====================================================
        if (0 <= kin1_1 && kin1_1 < 3 && 0 <= kin1_2 && kin1_2 < 3){ //座標に値が存在するかを判定

                //配列resulより"kin[0]"を探索する
                for (i = 0; i < 3; i++){
                        for (j = 0; j < 3; j++){
                                if (resul[i][j] == kin[0]){
                                        ireko1 = i;
                                        ireko2 = j;
                                        i = 999; //多重ループ脱出用
                                        j = 999;
                                }
                        }
                }

                //マンハッタン距離を求める
                manh_i = kin1_1 - ireko1;
                manh_j = kin1_2 - ireko2;

                manh_i = abs(manh_i); //絶対値に直す
```

```
                manh_j = abs(manh_j);

                manh_1 = manh_i + manh_j;

                //マンハッタン距離(合計)を求める
                        mang_1 = 0;
                for (i = 0; i<10; i++){ //manh_1にmanh[]に格納した各マンハッタン距離を加算する
                        mang_1 = mang_1 + manh[i];
                }
                        mang_1 = mang_1 - manh[kin[0] - 1];
                        mang_1 = mang_1 + manh_1 + 1;
        }

        //=====================================================

        if (0 <= kin2_1 && kin2_1 < 3 && 0 <= kin2_2 && kin2_2 < 3){ //座標に値が存在するかを判定

                //配列resulより"kin[1]"を探索する
                for (i = 0; i < 3; i++){
                        for (j = 0; j < 3; j++){
                                if (resul[i][j] == kin[1]){
                                        ireko1 = i;
                                        ireko2 = j;
                                        i = 999; //多重ループ脱出用
                                        j = 999;
                                }
                        }
                }

                //マンハッタン距離を求める
                manh_i = kin2_1 - ireko1;
                manh_j = kin2_2 - ireko2;

                manh_i = abs(manh_i); //絶対値に直す
```

```
                manh_j = abs(manh_j);

                manh_2 = manh_i + manh_j; //マンハッタン距離算出

                //マンハッタン距離(合計)を求める
                        mang_2 = 0;
                for (i = 0; i < 10; i++){ //manh_1にmanh[]に格納した各マンハッタン距離を
加算する
                        mang_2 = mang_1 + manh[i];
                }
                        mang_2 = mang_2 - manh[kin[1] - 1];
                        mang_2 = mang_2 + manh_2 + 1;

        }

        //=================================================

        if (0 <= kin3_1 && kin3_1 < 3 && 0 <= kin3_2 && kin3_2 < 3){ //座標に値が存在する
かを判定

                //配列resulより"kin[2]"を探索する
                for (i = 0; i < 3; i++){
                        for (j = 0; j < 3; j++){
                                if (resul[i][j] == kin[2]){
                                        ireko1 = i;
                                        ireko2 = j;
                                        i = 999; //多重ループ脱出用
                                        j = 999;
                                }
                        }
                }

                //マンハッタン距離を求める
                manh_i = kin3_1 - ireko1;
                manh_j = kin3_2 - ireko2;
```

```
                manh_i = abs(manh_i); //絶対値に直す
                manh_j = abs(manh_j);

                manh_3 = manh_i + manh_j;

                //マンハッタン距離(合計)を求める
                        mang_3 = 0;
                for (i = 0; i < 10; i++){ //manh_1にmanh[]に格納した各マンハッタン距離を
加算する
                        mang_3 = mang_3 + manh[i];
                }
                        mang_3 = mang_3 - manh[kin[1] - 1];
                        mang_3 = mang_3 + manh_3 + 1;

        }

        //=====================================================

        if (0 <= kin4_1 && kin4_1 < 3 && 0 <= kin4_2 && kin4_2 < 3){ //座標に値が存在する
かを判定

                //配列resulより"kin[3]"を探索する
                for (i = 0; i < 3; i++){
                        for (j = 0; j < 3; j++){
                                if (resul[i][j] == kin[3]){
                                        ireko1 = i;
                                        ireko2 = j;
                                        i = 999; //多重ループ脱出用
                                        j = 999;
                                }
                        }
                }

                //マンハッタン距離を求める
                manh_i = kin4_1 - ireko1;
                manh_j = kin4_2 - ireko2;
```

```
                manh_i = abs(manh_i); //絶対値に直す
                manh_j = abs(manh_j);

                manh_4 = manh_i + manh_j;

                //マンハッタン距離(合計)を求める
                        mang_4 = 0;
                for (i = 0; i < 10; i++){ //manh_1にmanh[]に格納した各マンハッタン距離を
加算する
                        mang_4 = mang_4 + manh[i];
                }
                        mang_4 = mang_4 - manh[kin[1] - 1];
                        mang_4 = mang_4 + manh_4 + 1;

        }
        //=====================================================
        //=====================================================

        //=====================================================

        //求めたマンハッタン距離(合計)より最小コストの手を選択
        min = mang_1; //manh_1をとりあえずの最小コストとする
        min_pazl = kin[0];
        change1 = kin1_1;
        change2 = kin1_2;

        if (min > mang_2){ //manh_1よりmanh_2の方がコストが小さいとき新たな最小コストとす
る
                min = mang_2;
                min_pazl = kin[1];
                change1 = kin2_1;
                change2 = kin2_2;

        }if (min > mang_3){ //minよりmanh_3の方がコストが小さいとき新たな最小コストとする
                min = mang_3;
```

```
                min_pazl = kin[2];
                change1 = kin3_1;
                change2 = kin3_2;

        }if (min > mang_4){ //min3よりmanh_4の方がコストが小さいとき新たな最小コストとす
る
                min = mang_4;
                min_pazl = kin[3];
                change1 = kin4_1;
                change2 = kin4_2;
        }

        //=====================================================

        first[ireko0_1][ireko0_2] = min_pazl; //最少コストの手を空間位置に動かす
        first[change1][change2] = 0;

    g = g + 1; //入れ替え回数をカウント

        f = min + g; //発見的関数の計算

        //=====================================================

        fprintf(file1, "%d %d %d¥n", first[0][0], first[0][1], first[0][2]);
        fprintf(file1, "%d %d %d¥n", first[1][0], first[1][1], first[1][2]);
        fprintf(file1, "%d %d %d¥n¥n", first[2][0], first[2][1], first[2][2]);

        fprintf(file2, "%d¥n",f);

        //=====================================================

        //配列resulより"kin[3]"を探索する
        for (i = 0; i < 3; i++){
                for (j = 0; j < 3; j++){
                        if (resul[i][j] != first[i][j]){
                                i = 999; //多重ループ脱出用
```

```
                                j = 999;
                                flug = 0;

                        }else{
                                flug = flug + 1;
        }}}

        if(end == 100){ //100回回ったら強制終了
                flug = 9;
        }

        end = end + 1;

        }while(flug != 9);

        fclose(file1); //ファイルを閉じる
        fclose(file2);


}
```

図2：ソースコード

3 プログラムの実行結果

実行環境

・Windows7 搭載 PC

・Microsoft Visual Studio 2013

Visual C++ 「空のプロジェクト」より実行

実行結果

テキスト形式のファイル「パズル状態」に現在のパズル状態，「発見的関数」に発見的関数の値を，1 回のスライド毎に書き込む。また，今回はソースコード未完成のため途中から無限ループに陥ってしまうので，100 回スライドを行った後に終了する様にしている。

発見的関数の推移

発見的関数の推移を図 3 に示す。

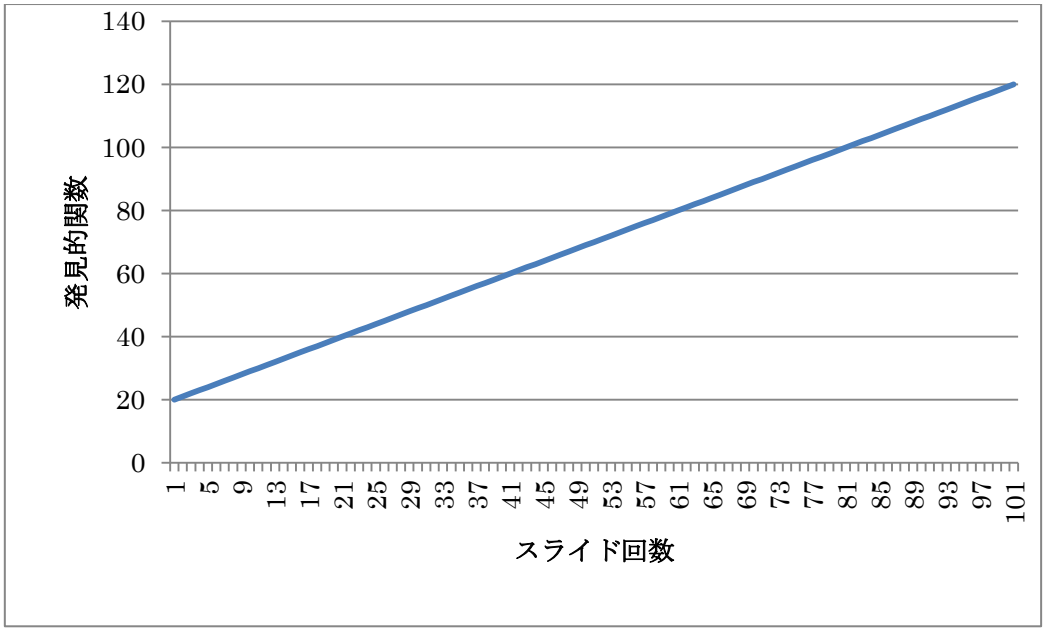

図 3：発見的関数の推移グラフ