人工知能Ⅱ　レポート

2017/10/11

5CS40 田畑悠己

１．アルゴリズムの説明

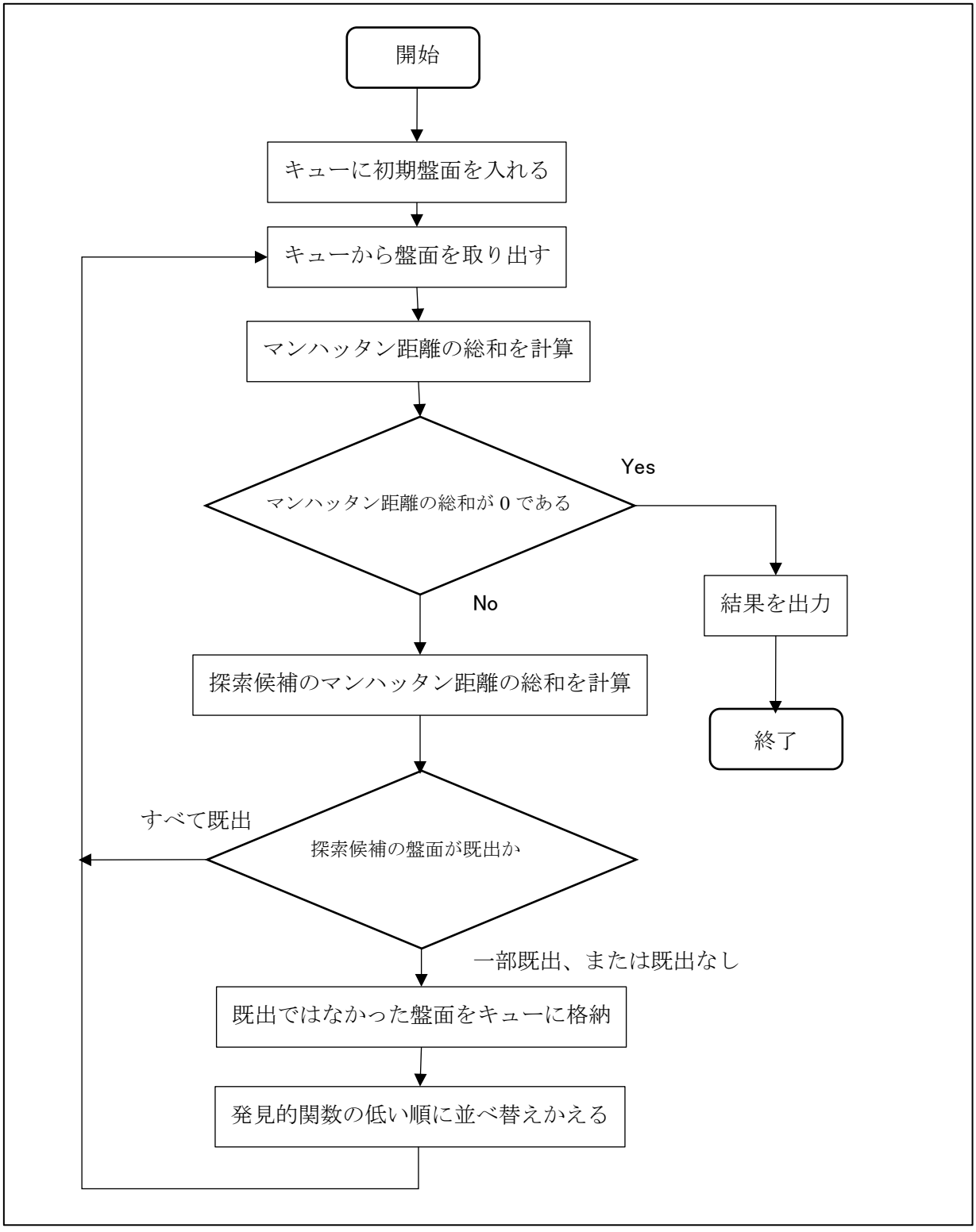


図 1:アルゴリズムのフローチャート

盤面ごとに探索候補のマンハッタン距離の総和を計算し、その小さい順に並び替える。既出ではない発見的関数が最小の盤面に移行して、再び探索候補のマンハッタン距離の総和を計算し、その小さい順に並び替える。これを繰り返して、マンハッタン距離の総和が０の盤面にたどり着いた場合、それまでの手数と、経路、発見的関数を出力して終了する。探索候補が存在しない(すべて既出)行き止まりについてしまった場合、一つ前の盤面に戻り、その盤面における、次に発見的関数が小さい盤面に移行する。

なお、このプログラムは未完成であり、ゴールへの到着、それまでの手数を出力することはできたが、経路及び発見的関数の出力が出来ていない。

２．ソースコード

8puzzle.cpp

```
#include<cstdio>
#include<iostream>
#include<cmath>
#include<map>
#include<queue>

using namespace std;

#define N 3                 //パズルの大きさを指定
#define N2 9      //N~2

//移動管理用
static const int dx[4] = { 0,-1,0,1 };
static const int dy[4] = { 1,0,-1,0 };
static const char dir[4] = { 'r','u','l','d' };


//マンハッタン距離の計算用
int MDT[N2][N2];


//パズル構造体
struct Puzzle {
        int f[N2], space, MD;
        int count;
```

図2:ソースコード-1

```
        bool operator<(const Puzzle &p)const {
                for (int i = 0;i < N2;i++) {
                        if (f[i] == p.f[i])continue;
                        return f[i] < p.f[i];
                }
                return false;
        }
};

//パズル構造体を内包したステイト構造体
struct State {
        Puzzle puzzle;
        int estimated;
        bool operator<(const State &s)const {
                return estimated > s.estimated;
        }
};

int GetALLMD(Puzzle m); //マンハッタン距離を計算
int astar(Puzzle s); //探索関数
void result(int n); //探索後の経路表示

//経路記録用
vector<State> Log(50);


//ここから開始
int main() {

        //マンハッタン距離の設定
        for (int i = 0;i < N2;i++)
                for (int j = 0;j < N2;j++)
                        MDT[i][j] = abs(i / N - j / N) + abs(i%N - j%N);

        Puzzle in;
```

図3:ソースコード-2

```
	//初期盤面を入力
	in.f[0] = 8; in.f[1] = 1; in.f[2] = 5;
	in.f[3] = 2; in.f[4] = 0; in.f[5] = 4;
	in.f[6] = 6; in.f[7] = 3; in.f[8] = 7;
	for (int i = 0;i < N2;i++) {
		//if (i % 3 == 0)printf("¥n");
		if (in.f[i] == 0) {
			in.f[i] = N2;
			in.space = i;
		}
		//printf("%d ",in.f[i]);
	}
	//printf("¥n");

	//手数を返り値とする
	int n=astar(in);

	//結果(経路と発見的関数)を出力
	result(n);

	getchar();
	return 0;
}

//マンハッタン距離を計算
int GetALLMD(Puzzle m) {
	int sum = 0;
	for (int i = 0;i < N2;i++) {
		if (m.f[i] == N2)continue;
		sum += MDT[i][m.f[i] - 1];
	}
	return sum;
}
```

図4:ソースコード-3

```
//探索関数
int astar(Puzzle s) {

	priority_queue<State> Qu; //探索候補を格納するキュー
	s.MD = GetALLMD(s); //現在の盤面でマンハッタン距離を計算
	s.count = 0; //探索の深さ
	map<Puzzle, bool>V; //既出盤面の判定
	Puzzle u, v;
	State first;
	first.puzzle = s; //初期盤面を格納
	first.estimated = GetALLMD(s); //マンハッタン距離を格納
	Qu.push(first); //キューに初期場面を格納

	while (!Qu.empty()) {
		State st = Qu.top();Qu.pop();		//キューから盤面を取り出す
		u = st.puzzle;


		Log[st.puzzle.count] = st; //経路を格納

		if (u.MD == 0)return u.count; //マンハッタン距離が0ならば終了

		V[u] = true;

		//マンハッタン距離計算用
		int sx = u.space / N;
		int sy = u.space % N;

		for (int r = 0;r < 4;r++) {
			//探索候補の盤面作成用
			int tx = sx + dx[r];
			int ty = sy + dy[r];

			if (tx < 0 || ty < 0 || tx >= N || ty >= N)continue;
			v = u;
```

図5:ソースコード-4

```
                        //マンハッタン距離を更新して計算
                        v.MD -= MDT[tx * N + ty][v.f[tx * N + ty] - 1];
                        v.MD += MDT[sx * N + sy][v.f[tx * N + ty] - 1];



                        //探索候補の盤面を作成
                        swap(v.f[sx * N + sy], v.f[tx * N + ty]);
                        v.space = tx * N + ty;



                        //盤面が既出でなければキューに格納
                        if (!V[v]) {
                                v.count++;
                                State next;
                                next.puzzle = v;
                                next.estimated = v.count + v.MD;
                                Qu.push(next); //キューに格納
                        }
                }
        }
        return -1; //エラー判定
}

//探索後の経路表示…がしたかった
void result(int n) {
        FILE *fp1, *fp2;
        fopen_s(&fp1, "経路.txt", "w");
        fopen_s(&fp2, "発見的関数.txt","w");
        for (int i = 0;i <= n;i++) {
                for (int j = 0;j < N2;j++) {
                        if (j % 3 == 0) {
                                printf("¥n");
                                fprintf(fp1, "¥n");
                        }
```

図6:ソースコード-5

```
                printf("%d ", Log[i].puzzle.f[j]);
                fprintf(fp1,"%d ", Log[i].puzzle.f[j]);
            }
            printf("¥n");
            fprintf(fp1, "¥n");
            printf("%d¥n", GetALLMD(Log[i].puzzle));
            fprintf(fp2,"%d %d¥n",i,GetALLMD(Log[i].puzzle)+i);
            fprintf(fp3, "%d %d¥n", i, GetALLMD(Log[i].puzzle));
        }
        fprintf(fp1, "¥n手数%d¥n",n);
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
}
```

図7:ソースコード-6

３．プログラムの実行結果

実行環境　Windows 10

visual stutio 2015

コマンドプロンプトに出力した結果

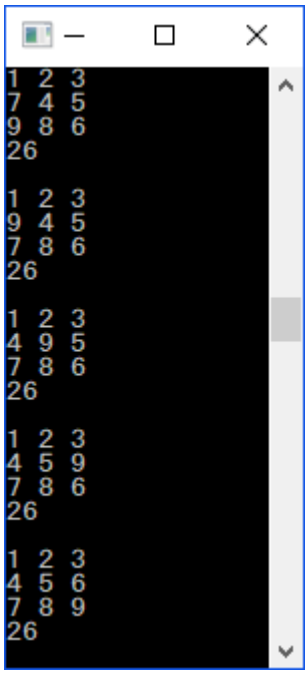


図 8;実行結果

実行結果は正しくない。経路と発見的関数の出力がうまくいっておらず、正しい経路を表示できていない。探索自体は正常に完了している。

一応、出力した経路に基づいた発見的関数と手数の関係をグラフに記述する。

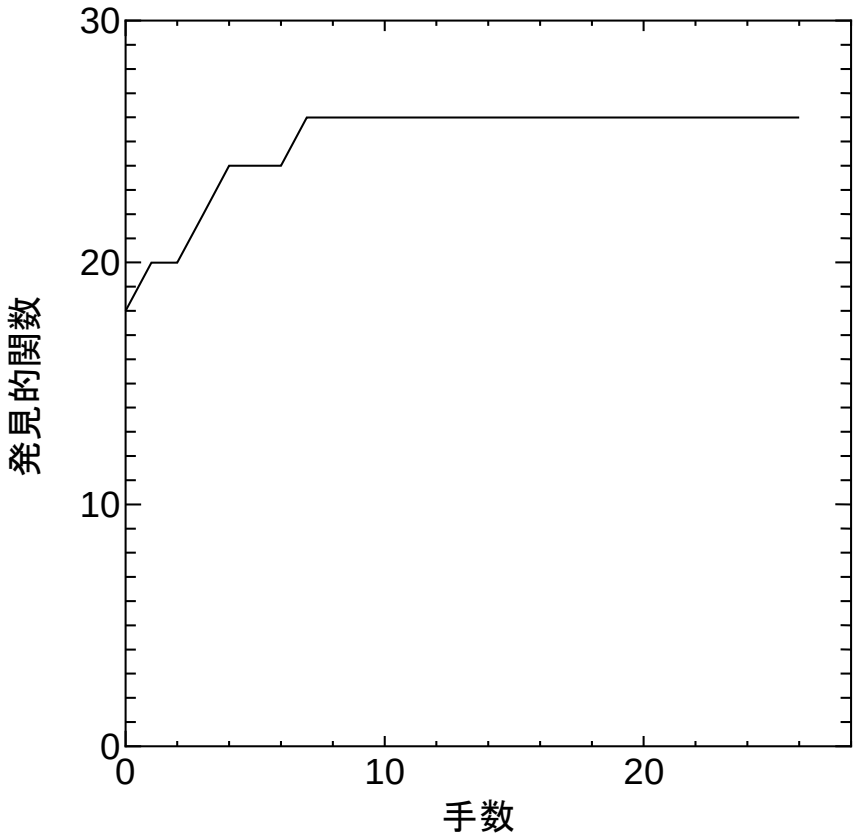
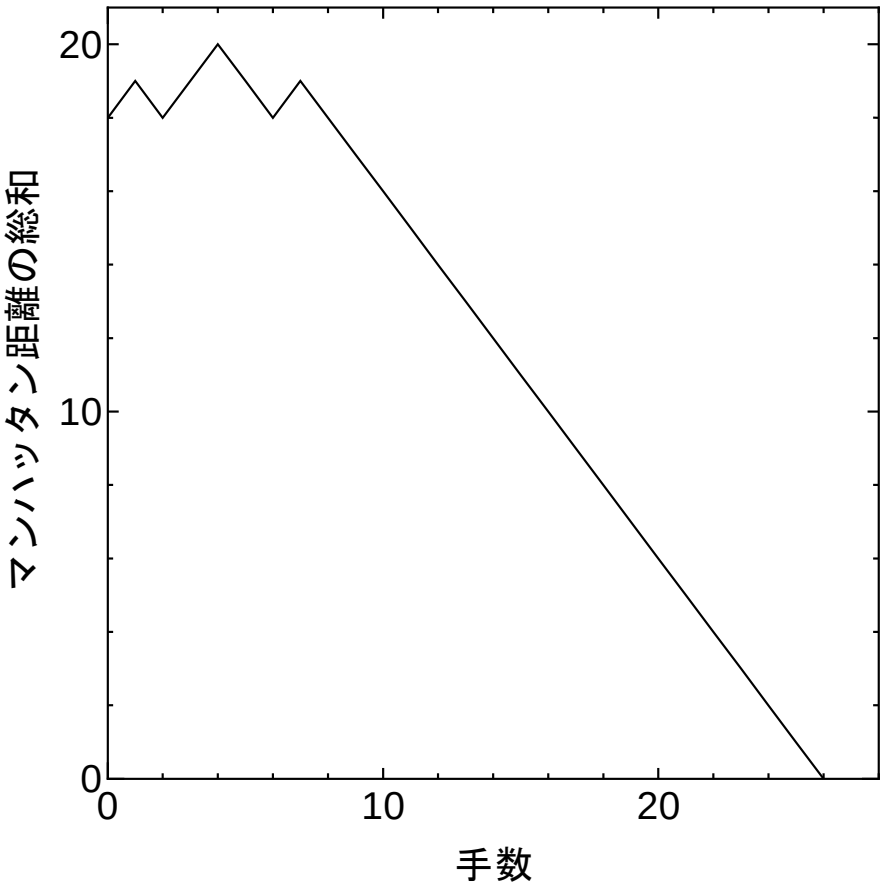


図 9:発見的関数と手数の関係

初期値は 20 だが、徐々に増えていき、最終的に 26 で収束した。また、増える値は 2 で固定であることがわかる。

また経路におけるマンハッタン距離の総和と手数の関係についてもグラフに記述した。



初期値は 20 である。最初のあたりで値のブレがあるものの、徐々に値は下がっていき最終的に 0 となった。

4.参考文献

プログラミングコンテスト攻略のためのアルゴリズムとデータ構造
2015 年 マイナビ社
著:渡部有隆