人工知能２ A*アルゴリズム

5CS56
安田大輝

1. アルゴリズムの説明
- 概要は図１参照のこと。各処理の詳細は各段落にて説明を行う。

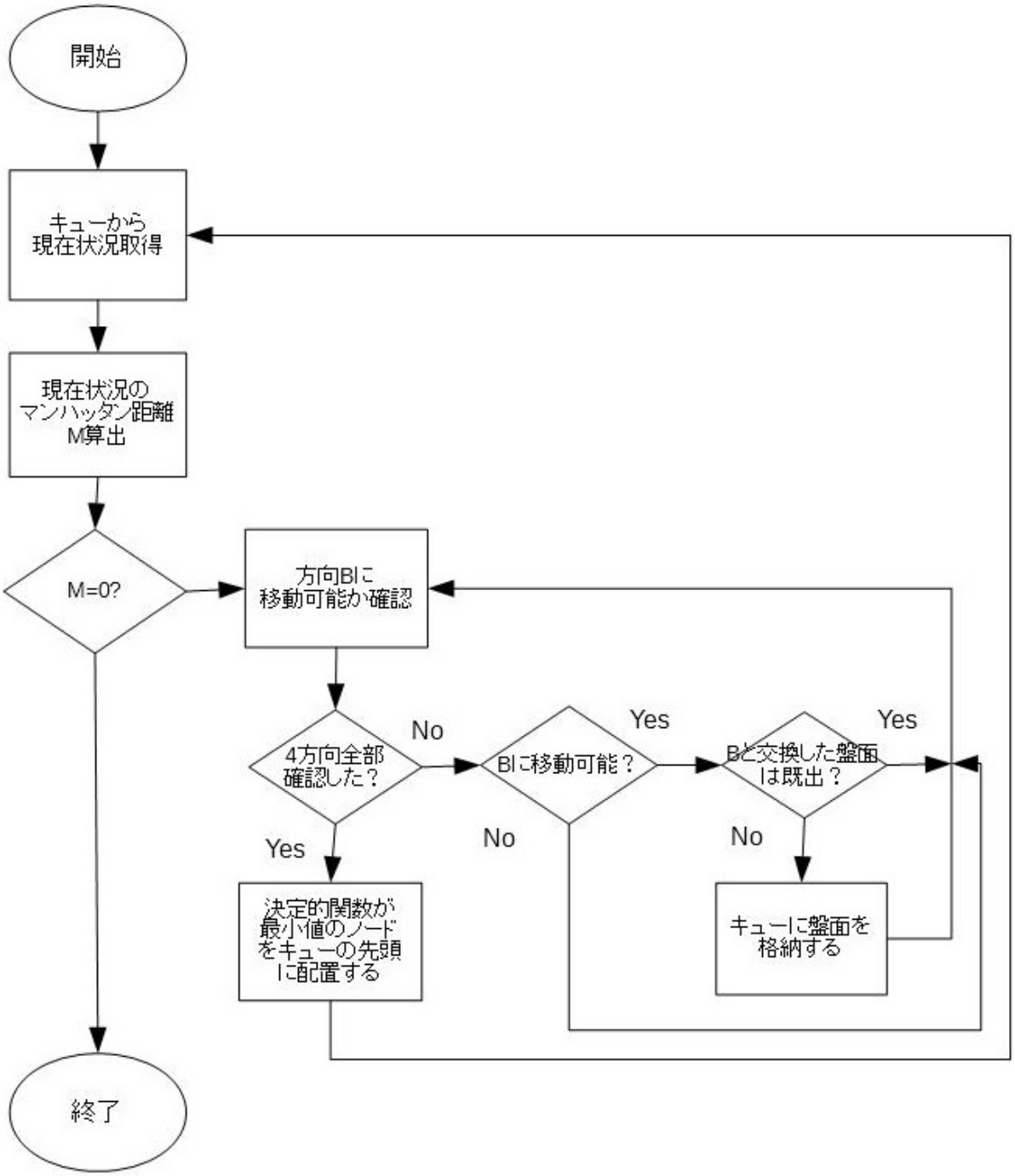

図１：８パズルの最短経路探索アルゴリズム

2. キューから現在状況取得
- 優先順位付きキューを用いて、現在状況を取得する。この場合の優先順位基準は決定的関数、即ちマンハッタン距離（以下Ｍ）＋移動回数（以下Ｃ）である。

3. 現在状況のＭ算出
- 予め宣言しておいた各距離を総和する。もしＭが０ならば、ゴール状態と一致していることとなる。

4. 方向Ｂに移動確認か確認
- 今回の場合、移動する方向は上下左右の計４方向である。これを移動用の変数に入れることで盤面の変更を行う。

5. キューに盤面を格納する

・ B 方向と交換した盤面が既出でないならば、その盤面・決定的関数をキューに入れる。

6. 決定的関数が最小値のノードをキューの先頭に配置する

• 優先順位付きキュー内部を決定的変数を基準にしてソートし直す。結果最小コストを持ったノードが先頭となり、これを繰り返す。

7. ソースコード

スコードは図 2 参照のこと。

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

using namespace std;

#define N (3)
#define N2 (9)
#define MBAN (30);

//移動方向の設定
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' };
static const int question[N2] = {
	8, 1, 5,
	2, 0, 4,
	6, 3, 7
};

//マンハッタン距離一覧
int man_d[N2][N2];

//パズル本体
struct Puzzle {
	int f[N2], space, man;
	int cost;

	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;
	}
};

//ガワ 総コストも用意
struct State {
	Puzzle puzzle;
	int estimated;
	bool operator<(const State &s)const {
		return estimated > s.estimated;
	}
};

//書き出し用
vector<State> ban(30);

//マンハッタン距離算出
//いちいち abs やるの面倒なので一覧から取得
int GetManhattan(Puzzle pz) {
	int sum = 0;
	for (int i = 0; i < N2; i++) {
		if (pz.f[i] == N2) {
			continue;
		}
		sum += man_d[i][pz.f[i] - 1];
	}
```

```
	return sum;
}

//表示、ファイル書きこみ
void Write(int count){
	FILE *fp, *fp2,*fp3;//fpは推移、fp2はコストとマンハッタン距離
	fopen_s(&fp, "Puzzle.txt", "w");
	fopen_s(&fp2, "Cost.txt", "w");
	fopen_s(&fp3, "Man.txt", "w");
	for (int i = 0; i <= count; i++){
		for (int j = 0; j < N2; j++){
			printf("%d ", ban[i].puzzle.f[j]);
			fprintf(fp,"%d ", ban[i].puzzle.f[j]);
			if (j == 2)printf(" cost:%d", ban[i].puzzle.cost);
			if (j == 5)printf(" dist:%d", ban[i].puzzle.man);
			if (j % N == N - 1) {
				printf("¥n");
				fprintf(fp, "¥n");
			}
		}
		printf("¥n");
		fprintf(fp, "¥n");
		fprintf(fp2, "%d,%d¥n", ban[i].puzzle.cost, ban[i].estimated);
		fprintf(fp3, "%d,%d¥n", ban[i].puzzle.cost, ban[i].puzzle.man);
	}
	fclose(fp);
	fclose(fp2);
	fclose(fp3);
}

bool Answer(Puzzle p){
	return
		p.f[0] == 1 &&
		p.f[1] == 2 &&
		p.f[2] == 3 &&
		p.f[3] == 4 &&
		p.f[4] == 5 &&
		p.f[5] == 6 &&
		p.f[6] == 7 &&
		p.f[7] == 8 &&
		p.f[8] == 9;
}

//本体
int Astar(Puzzle s) {
	//初期宣言
	priority_queue<State> PQ;
	s.man = GetManhattan(s);
	s.cost = 0;
	map<Puzzle, bool>V;
	Puzzle u, v, temp;
	//初期値作成
	State init;
	init.puzzle = s;
	init.estimated = GetManhattan(s);
	PQ.push(init);

	ban[init.puzzle.cost] = init;

	while (!PQ.empty()) {
		//先頭取り出し
		State st = PQ.top(); PQ.pop();

		u = st.puzzle;

		//マンハッタン算出
		if (u.man == 0){
			ban[st.puzzle.cost] = st;
			return u.cost;
```

```
                }
                V[u] = true;

                //Write(u);

                //座標取得
                int sx = u.space / N;
                int sy = u.space % N;

                for (int r = 0; r < 4; r++) {
                        //各方向と交換
                        int tx = sx + dx[r];
                        int ty = sy + dy[r];
                        if (tx < 0 || ty < 0 || tx >= N || ty >= N)continue;
                        v = u;
                        v.man -= man_d[tx * N + ty][v.f[tx * N + ty] - 1];
                        v.man += man_d[sx * N + sy][v.f[tx * N + ty] - 1];

                        //座標交換
                        swap(v.f[sx * N + sy], v.f[tx * N + ty]);

                        v.space = tx * N + ty;
                        //既出か否か
                        if (!V[v]) {
                                //交換後の盤面をキューにシュゥゥゥーッ！
                                v.cost++;
                                State news;
                                news.puzzle = v;
                                news.estimated = v.cost + v.man;
                                PQ.push(news);
                                ban[news.puzzle.cost] = news;
                        }
                }
        }
        return -1;
}

int main() {
        //マンハッタン距離をねじ込む
        for (int i = 0; i < N2; i++)
                for (int j = 0; j < N2; j++)
                        man_d[i][j] = abs(i / N - j / N) + abs(i%N - j%N);
        //初期配置
        Puzzle in;

        /*
        for (int i = 0;i < N2;i++) {
        cin >> in.f[i];
        if (in.f[i] == 0) {
        in.f[i] = N2;
        in.space = i;
        }
        }
        */
        //初期宣言
        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¥n");

        int n = Astar(in);
```

```
        Write(n);


        //getchar();
        return 0;
}
```

図２：ソースコード

8. 実行結果
9. 実行環境は以下のとおりである。

- OS:Windows10
- CPU:Intel core i5 1.7GHz
- RAM:8GB
- 盤面の移動結果

1. 移動結果の一部は図３を参照のこと。

```
1 2 3
7 9 5
8 4 6

1 2 3
9 4 7
8 5 6

1 2 3
7 4 5
9 8 6

1 2 3
9 4 5
7 8 6

9 2 3
1 4 5
7 8 6

1 2 3
4 8 5
7 9 6

1 2 3
4 5 6
7 8 9
```

図３：パズル移動の一部

パズル実行結果の一部に、一気に移動している部分があった。この部分は、それ以前のコストで行き止まりとなった結果、別の盤面から移動してきた結果と思われる。この点は課題の一つである。

10. 決定的関数の推移

- 決定的関数およびマンハッタン距離の推移は図４を参照のこと。

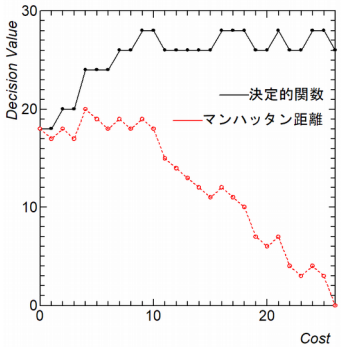

図 4：移動コストと決定的関数・マンハッタン距離の推移

- 発的関数は 2 刻みに変動しており、最終的には 26～28 に収束している。
  - 一方、マンハッタン距離は所どころ上昇しているものの、徐々に 0 に収束していることがわかる。

1. 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 著：渡部有隆 2015 年 マイナビ社