人工知能Ⅱ課題

5CS54 黛 礼雄

1　アルゴリズムの説明

このプログラムは A*アルゴリズムを用いた深さ優先探索を行なっている．以下にフローチャートを示す．

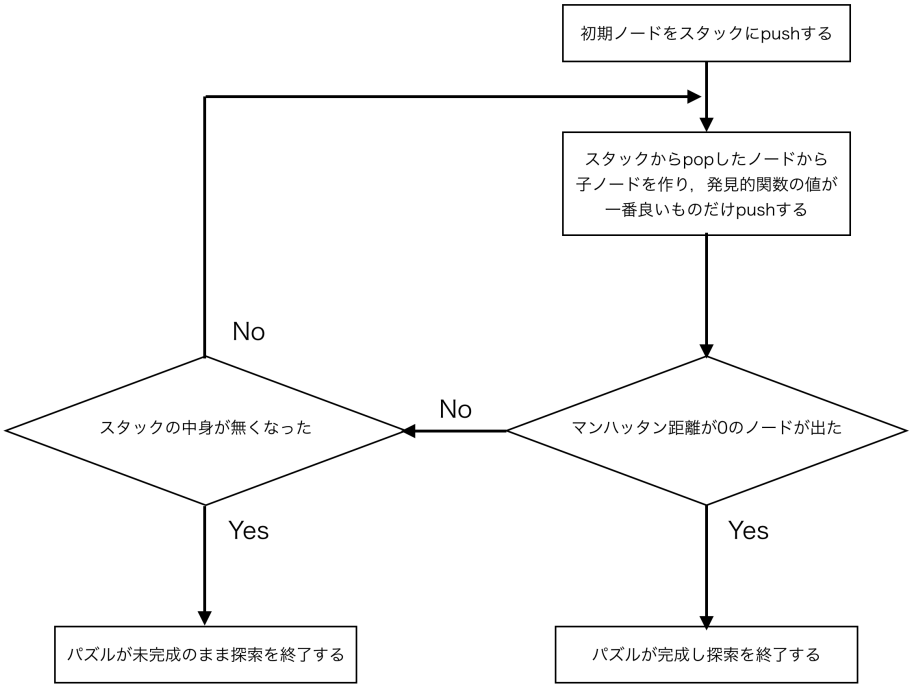

図 1　フローチャート

探索をするにあたり，1 つのパズルの状態とマンハッタン距離の総和，木構造での深さ，正しい状態であるかどうか，操作手順の 5 つの情報をまとめてノードと表した．そのノードをデータ構造であるスタックに push することにより探索を進める．また，このプログラムでは空白マスを 9 として扱う．

そのまま探索を進めるとノードの数が莫大になってしまうため，これらの枝刈りを行なった．

- 過去に探索した状態を保持し，同じ状態のノードは push しない．
- 一つのノードから得られる子ノードは発見的関数の値の良いもののみ push する．同じ値が複数ある場合はそれぞれ push する．
- 発見的関数の値を 28 までとし，それ以上の値をもつノードは push しない．
- 親ノードで行なった操作を元に戻す動作をするノードは正しくない状態とする．

特に発見的関数の値に上限を持たせることにより解にたどり着く探索回数が大幅に少なくなった．

2　ソースコード

以下に C#で書かれたソースコードを示す.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.IO;
5
6  namespace A_Star
7  {
8      class MainClass
9      {
10         public struct Node//ノード構造体
11         {
12             public int[,] Table;//パズルの盤面
13                         public int Manhattan_Cost;//マンハッタンコスト
14                         public int Deep_Cost;//深さ
15                         public bool Currect_node;//ノードが正しいかどうか
16             public string Operation;//操作手順:u(上)r(右)d(下)l(左)
17
18                         public Node(int[,] _Table,int _Manhattan_Cost,int _Deep_Cost,
                               bool _Currect_node,string _Operation)
19                         {
20                 Table = _Table;
21                 Manhattan_Cost = _Manhattan_Cost;
22                 Deep_Cost = _Deep_Cost;
23                 Currect_node = _Currect_node;
24                 Operation = _Operation;
25                         }
26         }
27         public static void Main(string[] args)
28         {
29             System.Diagnostics.Stopwatch sw = new System.Diagnostics.Stopwatch();//時間
                   計測
30             sw.Start();//時間計測スタート
31             Node data = new Node();//初期データ用ノード
32
33             data.Table = new int[3, 3] { { 8, 1, 5 }, { 2, 9, 4 }, { 6, 3, 7 } };//初期
                   状態, 空白パネルは9とする
34
35             List<int[,]> All_scene = new List<int[,]>();//全ての盤面を格納するリスト
36
37             for (int i = 0; i < 3;i++)//
38             {
39                 for (int j = 0; j < 3;j++)
40                 {
41                     Console.WriteLine("(" + (i + 1)  + " ," + (j + 1) + ") " +
42                                       " Value = " + data.Table[i,j] + " Cost = " +
                                             Manhattan(j + 1, i + 1, data.Table[i, j]));
43                     data.Manhattan_Cost += Manhattan(j + 1, i + 1, data.Table[i, j]);
                           //初期ノードにマンハッタン距離を
                           加算
44                 }
45             }
46             Console.WriteLine("コストの総和 : " + data.Manhattan_Cost);
47
48             Console.WriteLine("コストのTotal :"+ Total_Manhattan(data.Table));
49
50             data.Deep_Cost = 0;//初期の深さ
51
```

```
52              All_scene.Add(data.Table);//初期の場面を記録
53
54                  Stack<Node> nodes = new Stack<Node>();//深さ優先探索をするために
                        stackを用意する
55                  nodes.Push(data);//根をstackに追加
56
57              Node currentnode = new Node();
58              currentnode.Table = new int[3, 3];
59
60              int Genetic_count = 0;
61              using (StreamWriter w = new StreamWriter(@"./output.txt"),graph = new
                    StreamWriter(@"./graph.txt"))
62              {
63                  while (nodes.Count() != 0)//探索開始～スタックが空になるまで
64                  {
65                      currentnode = Copy_node(nodes.Pop());//現在のノードにスタックからポップし
                            コピー
66
67                      List<Node> nextnode = new List<Node>();//
                            currentnodeからできる子ノードの格納リスト
68
69                      int bestcost = int.MaxValue;
70
71                      for (int i = 0; i < 4; i++)//親ノードから生まれる子は最大4つ
72                      {
73                          nextnode.Add(GetNextNode(currentnode, i));//子ノードを取得
74                          if (nextnode[i].Currect_node == false)//正しくない操作は弾く
75                              continue;
76
77                          if (isMatchScene(All_scene, nextnode[i].Table))//過去に同じ盤面が
                                あった場合弾く
78                              continue;
79
80                          if (bestcost > nextnode[i].Manhattan_Cost + nextnode[i].
                                Deep_Cost)//最良の発見的関数を
                                取得
81                                                  bestcost = nextnode[i].
                                                        Manhattan_Cost + nextnode[
                                                        i].Deep_Cost;
82
83                          if (nextnode[i].Manhattan_Cost == 0)//完成した時
84                          {
85                              currentnode = Copy_node(nextnode[i]);//現在のノードに完成ノードを
                                    コピー
86                              Console.WriteLine("Exitloop");
87                              goto exitloop;//2重ループを抜ける
88                          }
89                      }
90                      foreach (var push_node in nextnode)//得られた子ノードをstackにプッシュ
91                      {
92                          if (push_node.Manhattan_Cost + push_node.Deep_Cost != bestcost
                                )//いいやつだけ
                                push
93                            continue;
94
95                          if (push_node.Deep_Cost + Total_Manhattan(push_node.Table) >
                                28)//発見的関数の値を
                                28までで制限する
96                              continue;
97
98                          All_scene.Add(push_node.Table);//スタックにプッシュする盤面を記録する
99
100                         nodes.Push(push_node);//プッシュ
101                         Console.WriteLine("発見的関数の値 : " + (push_node.Deep_Cost +
```

```
                        Total_Manhattan(push_node.Table)));
102                     w.WriteLine("発見的関数の値 : " + (push_node.Deep_Cost +
                        Total_Manhattan(push_node.Table)));
103                     Console.WriteLine("Deep : " + push_node.Deep_Cost);
104                     Console.WriteLine("Manhattan" + Total_Manhattan(push_node.
                        Table));
105                     Console.WriteLine(push_node.Operation);
106                     Genetic_count++;
107                     graph.WriteLine(Genetic_count + " " + (push_node.Deep_Cost  +
                        Total_Manhattan(push_node.Table)));//深さ
                         マンハッタン
108                     for (int i = 0; i < 3; i++)
109                     {
110                         for (int j = 0; j < 3; j++)
111                         {
112                             Console.Write(push_node.Table[i, j] + " ");
113                             w.Write(push_node.Table[i, j] + " ");
114                         }
115                         Console.WriteLine();
116                         w.WriteLine();
117                     }
118                     Console.WriteLine();
119                     Console.WriteLine();
120                     w.WriteLine();
121                     w.WriteLine();
122                 }
123
124             }
125         exitloop:;
126             Console.WriteLine();
127             Console.WriteLine();
128             Console.WriteLine("初期盤面");
129             for (int i = 0; i < 3; i++)
130             {
131                 for (int j = 0; j < 3; j++)
132                 {
133                     Console.Write(data.Table[i, j] + " ");
134                 }
135                 Console.WriteLine();
136             }
137             Console.WriteLine("最後のマンハッタン距離 = " + currentnode.Manhattan_Cost);
138             Console.WriteLine("発見的関数の値 : " + (currentnode.Deep_Cost +
                Total_Manhattan(currentnode.Table)));
139             w.WriteLine("最終発見的関数の値 : " + (currentnode.Deep_Cost +
                Total_Manhattan(currentnode.Table)));
140             Console.WriteLine("Deep : " + currentnode.Deep_Cost);
141             Console.WriteLine("Manhattan" + Total_Manhattan(currentnode.Table));
142             Console.WriteLine("操作手順 " + currentnode.Operation);
143             Console.WriteLine("最終盤面");
144             w.WriteLine("操作手順 " + currentnode.Operation);
145             for (int i = 0; i < 3; i++)
146             {
147                 for (int j = 0; j < 3; j++)
148                 {
149                     Console.Write(currentnode.Table[i, j] + " ");
150                     w.Write(currentnode.Table[i, j] + " ");
151                 }
152                 Console.WriteLine();
153                 w.WriteLine();
154             }
155             sw.Stop();
156             Console.WriteLine(sw.Elapsed);
```

```
157             }
158         }
159
160         public static Node GetNextNode(Node currentNode,int direction)
161         {
162                         Node nextnode = new Node();
163                         nextnode.Table = new int[3, 3];
164                         nextnode.Currect_node = true;
165                         for (int i = 0; i < 3; i++)
166                         {
167                                 for (int j = 0; j < 3; j++)
168                                 {
169                                         nextnode.Table[i, j] = currentNode.Table[i, j
                                            ];//盤面だけコ
                                            ピー
170                                 }
171                         }
172             nextnode.Operation = currentNode.Operation;//操作手順のコピー
173
174             int[] Empty_position = Find_9(currentNode.Table);//空白パネルの座標の取得
175
176             switch(direction)//空白パネルとその上下左右の向き(direction)のパネルの入れ替え操作
177             {
178                 case 0://上
179                     if (Empty_position[0]  == 0)//上を選択すると範囲外となってしまうので正しくない
                            ノードとする
180                     {
181                         nextnode.Currect_node = false;
182                         return nextnode;
183                     }
184                     try
185                     {
186                         if (nextnode.Operation.Substring(nextnode.Operation.Length -
                                1) == "d")//手順のループを
                                防ぐ
187                         {
188                             nextnode.Currect_node = false;
189                             return nextnode;
190                         }
191                     }
192                     catch(System.NullReferenceException)//操作手順がないとエラーを吐いてしまう
193                     {
194                     }
195                     //入れ替えと操作手順の追加
196                     nextnode.Table[Empty_position[0],Empty_position[1]] = nextnode.
                            Table[Empty_position[0] - 1, Empty_position[1]];
197                     nextnode.Table[Empty_position[0] - 1, Empty_position[1]] = 9;
198                     nextnode.Operation += "u";
199                     break;
200
201                 case 1://右
202                     if (Empty_position[1]  == 2)//右を選択すると範囲外となってしまうので正しくない
                            ノードとする
203                                     {
204                                             nextnode.Currect_node = false;
205                                             return nextnode;
206                                     }
207                                     try
208                                     {
209                                             if (nextnode.Operation.Substring(
                                                    nextnode.Operation.Length - 1) ==
                                                    "l")//手順のループを
                                                    防ぐ
```

```
                                        {
                        nextnode.Currect_node = false;
                                            return nextnode;
                                        }
                                    }
                                    catch (System.NullReferenceException)
                                    {
                                    }
                    nextnode.Table[Empty_position[0], Empty_position[1]] = nextnode.
                        Table[Empty_position[0], Empty_position[1] + 1];
                    nextnode.Table[Empty_position[0], Empty_position[1] + 1] = 9;
                    nextnode.Operation += "r";
                    break;

                case 2://下
                    if (Empty_position[0] + 1 == 3)//下を選択すると範囲外となってしまうので正しくな
                        いノードとする
                                    {
                                        nextnode.Currect_node = false;
                                        return nextnode;
                                    }
                                    try
                                    {
                                        if (nextnode.Operation.Substring(
                                            nextnode.Operation.Length - 1) ==
                                            "u")//手順のループを
                                            防ぐ
                                        {
                        nextnode.Currect_node = false;
                                            return nextnode;
                                        }
                                    }
                                    catch (System.NullReferenceException)
                                    {
                                    }
                    nextnode.Table[Empty_position[0], Empty_position[1]] = nextnode.
                        Table[Empty_position[0] + 1, Empty_position[1]];
                    nextnode.Table[Empty_position[0] + 1, Empty_position[1]] = 9;
                    nextnode.Operation += "d";
                    break;

                case 3://左
                    if (Empty_position[1] == 0)//左を選択すると範囲外となってしまうので正しくない
                        ノードとする
                                    {
                                        nextnode.Currect_node = false;
                                        return nextnode;
                                    }
                                    try
                                    {
                                        if (nextnode.Operation.Substring(
                                            nextnode.Operation.Length - 1) ==
                                            "r")//手順のループを
                                            防ぐ
                                        {
                        nextnode.Currect_node = false;
                                            return nextnode;
                                        }
                                    }
                                    catch (System.NullReferenceException)
                                    {
                                    }
                    nextnode.Table[Empty_position[0], Empty_position[1]] = nextnode.
```

```
                    Table[Empty_position[0], Empty_position[1] - 1];
263                 nextnode.Table[Empty_position[0], Empty_position[1] - 1] = 9;
264                 nextnode.Operation += "l";
265                 break;
266             }
267             nextnode.Manhattan_Cost = Total_Manhattan(nextnode.Table);//マンハッタン距離の
                    総和
268             nextnode.Deep_Cost = currentNode.Deep_Cost + 1;//深さを加算
269             return nextnode;
270                 }
271         public static bool isMatchScene(List<int[,]> all_panel,int[,] current_panel)//
                過去の盤面と同じものがあるかど
                うか
272         {
273             int count = 0;
274             for (int h = 0; h < all_panel.Count();h++)
275             {
276                 for (int i = 0; i < 3;i++)
277                 {
278                     for (int j = 0; j < 3;j++)
279                     {
280                         if (all_panel[h][i, j] == current_panel[i, j])
281                         {
282                             count++;
283                         }
284                     }
285                 }
286                 if (count == 9)//あった
287                 {
288                     return true;
289                 }
290                 count = 0;
291             }
292             return false;
293         }
294         public static Node Copy_node(Node data)//ノードのコピー
295         {
296                         Node node = new Node();
297             node.Table = new int[3, 3];
298             for (int i = 0; i < 3;i++)
299             {
300                 for (int j = 0; j < 3;j++)
301                 {
302                     node.Table[i, j] = data.Table[i, j];
303                 }
304             }
305                     node.Manhattan_Cost = data.Manhattan_Cost;
306                     node.Deep_Cost = data.Deep_Cost;
307                     node.Currect_node = data.Currect_node;
308             node.Operation = data.Operation;
309                     return node;
310                 }
311         public static int Total_Manhattan(int[,] data)//マンハッタン距離の総和
312         {
313             int cost = 0;
314                     for (int i = 0; i < 3;i++)
315                     {
316                         for (int j = 0; j < 3;j++)
317                         {
318                             cost += Manhattan(j + 1, i + 1, data[i, j]);
319                         }
320                     }
```

```
321             return cost;
322                 }
323         public static int[] Find_9(int[,] data)//空白マスの座標取得
324         {
325             int[] answer = new int[2];
326             for (int i = 0; i < 3;i++)
327             {
328                 for (int j = 0; j < 3;j++)
329                 {
330                     if (data[i, j] == 9)
331                     {
332                         answer[0] = i;
333                         answer[1] = j;
334                         break;
335                     }
336                 }
337             }
338             return answer;
339         }
340         public static int Manhattan(int x,int y,int value)//マンハッタン距離を求める
341         {
342             int cost = 0;
343             if (value == 9)//9は空白パネルなのでカウントしない
344                 return cost;
345             //行揃え
346             if(y == 1)
347             {
348                 if(!(value >= 1 && value <= 3))//行で移動する必要がある
349                 {
350                     if(value >= 4 && value <= 6)//1段下に移動するのでコスト＋1
351                     {
352                         cost += 1;
353                     }
354                     else if(value >= 7 && value <= 9)//2段下に移動するのでコスト＋2
355                     {
356                         cost += 2;
357                     }
358                 }
359             }
360             else if (y == 2)
361             {
362                 if(!(value >= 4 && value <= 6))//行で移動する必要がある
363                                 {
364                     cost += 1;//2行目から上下に移動してもコストは必ず1
365                 }
366             }
367             else if (y == 3)
368             {
369                 if(!(value >= 7 && value <= 9))//行で移動する必要がある
370                                 {
371                                         if (value >= 1 && value <= 3)//2段上に移動するのでコスト＋2
372                                         {
373                                                 cost += 2;
374                                         }
375                                         else if (value >= 4 && value <= 6)//1段上に移動するのでコスト＋1
376                                         {
377                                                 cost += 1;
378                                         }
379                 }
```

```
380             }
381             //列揃え
382             if(x == 1)
383             {
384                 if (!(value == 1 || value == 4 || value == 7))//列で移動する必要がある
385                 {
386                     if(value == 2 || value == 5 || value == 8)//1列右に移動するのでコスト＋1
387                     {
388                         cost += 1;
389                     }
390                     else if(value == 3 || value == 6 || value == 9)//1列右に移動するのでコスト＋2
391                                         {
392                         cost += 2;
393                     }
394                 }
395             }
396             else if (x == 2)
397             {
398                 if (!(value == 2 || value == 5 || value == 8))//列で移動する必要がある
399                                     {
400                     cost += 1;//2列目から左右に移動してもコスト＋1
401                 }
402             }
403             else if (x == 3)
404             {
405                 if(!(value == 3 || value == 6 || value == 9))//列で移動する必要がある
406                                     {
407                     if(value == 1 || value == 4 || value == 7)//2列左に移動するのでコスト＋2
408                                         {
409                         cost += 2;
410                     }
411                     else if(value == 2 || value == 5 || value == 8)//1列左に移動するのでコスト＋1
412                                         {
413                         cost += 1;
414                     }
415                 }
416             }
417             return cost;
418         }
419     }
420 }
```

図2　ソースコード

3　実行

3.1　実行環境

- 動作確認 PC
  - macOS High Sierra 10.13
  - 3.1GHz Intel Core i5
  - LPDD3 16GB
- コンパイラコマンド
  - mcs -version 5.2.0.0
- 実行コマンド
  - mono -version 5.2.0.224

3.2　実行結果

実行した結果は output.txt として出力される．
以下に発見的関数の値と探索回数の推移のグラフを示す．

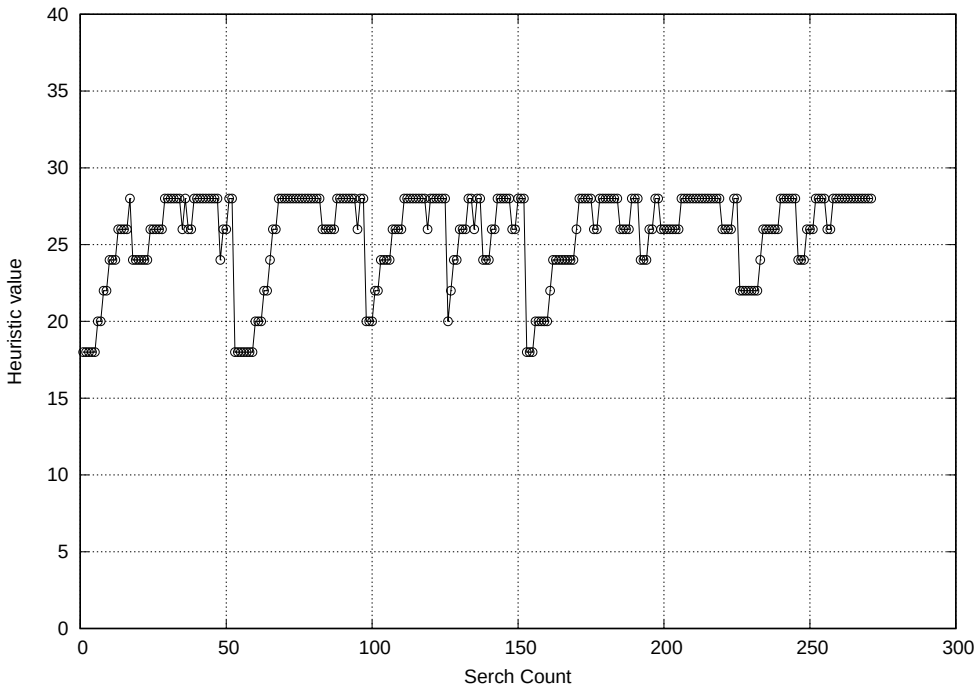

図 3　発見的関数の推移

図 3 を見ると探索を初めた直後は発見的関数の値は小さいが，進めていくにつれて大きくなることがわかる．また，急激に値が小さくなる箇所については，子ノードが作れなくなり根に近いノードが pop され探索が再び始まったと考えれられる．