School of Data and Computer Science, Sun Yat-sen University

E04 Futoshiki Puzzle (Forward Checking)

18340149 孙新梦

September 21, 2020

目录

1 Futoshiki

Futoshiki is a board-based puzzle game, also known under the name Unequal. It is playable on a square board having a given fixed size (4×4 for example).

The purpose of the game is to discover the digits hidden inside the board' s cells; each cell is filled with a digit between 1 and the board' s size. On each row and column each digit appears exactly once; therefore, when revealed, the digits of the board form a so-called Latin square.

At the beginning of the game some digits might be revealed. The board might also contain some inequalities between the board cells; these inequalities must be respected and can be used as clues in order to discover the remaining hidden digits.

Each puzzle is guaranteed to have a solution and only one.

You can play this game online: http://www.futoshiki.org/.

2 Tasks

1. Please solve the above Futoshiki puzzle (Figure 1) with forward checking algorithm.
2. Write the related codes and take a screenshot of the running results in the file named E04 YourNumber.pdf, and send it to ai 2018@foxmail.com.

3 My Analasis

1. Futoshiki Puzzle 的游戏目标是将数字横竖都不重复地填入框中，且遵循大小于号不等式指示。游戏形式与常见的数独游戏十分相似，不同之处只有：无需遵循九宫格限制，添加了相邻格子之间的不等号限制。
2. Futoshiki游戏和Sudoku数独游戏一样，是典型的Constraint satisfaction problem (CSP) 约束满足问题，其解法基于Forward Checking算法向前检测来完成基于回溯算法的优化

Forward Checking算法介绍

1. FC算法是回溯算法的扩展，旨在通过对没有赋值的变量提前试图寻找明显的failure，若存在则可直接由值域DWO回溯，不再检测之后的状态。若无DWO也可明显缩减之后的值域。
2. 当一个变量被实例化时（该变量在循环中被访问到，亦可理解为assigned），对所有包含该变量的限制条件（constraints）进行检查，并且找到只剩一个未确定的变量的限制条件。

3. 每次在值域中挑选一个值，赋值一个变量，检查所有和该变量有关的约束，此时对其他变量值域用约束进行更新，如果把这个变量的这个赋值使得有约束不满足的话，则把这个值从值域移除，若达到空值域，也就是DWO则直接返回

4. 需要注意的是约束传播是需要资源的。如果判断不能很高效地完成，则容易在搜索更少节点和搜索速度更快的tradeoff中，在推断中花费更多时间。

形式化本问题

1. Futoshiki游戏的状态节点是对一个棋盘的不完全赋值，解是叶子节点也就是最下面一层的完全赋值。初始状态是空赋值，也就是题目给出的取值

2. 对于下一个赋值节点的选取，我们使用MRV，也就是最小剩余值启发的方法，优先赋值值域更小的变量，更容易快速削减值域找到DWO——heuristicpick函数

3. 我们维护一个值域domin，对于一个赋值操作之后，将三条约束（1.行不同，2.列不同，3.不等于号约束）依次使用，来对每一个位置的domin进行更新——propagate函数

4. FCCheck做的事情是对于不同数字代表的约束检测，调用对应检测函数来更新当前位置的值域

5. 因为考虑到空间复杂度的问题，我用maps构造了一个结构体为元素的二维函数board，分别表示每一个位置的值，坐标，是否赋值，还有上下左右有无不等号约束以便于分约束检测。原本的约束通过对lrud四个变量的不同取值来代表不同约束。

4 Codes

```
/*
*    姓名：孙新梦
*    学号：18340149
*    作业：人工智能作业四Futoshiki
*    程序编译：gcc -o futoshiki futoshiki.c
*    运行：./futoshiki
*    知识点：算法FC
*/
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;
```

```
13
14  const int SIZE = 9;
15  static int nodes = 0;
16
17  //初始地图
18  int maps[9][9] = { {0, 0, 0, 7, 3, 8, 0, 5, 0},
19          {0, 0, 7, 0, 0, 2, 0, 0, 0},
20          {0, 0, 0, 0, 0, 9, 0, 0, 0},
21          {0, 0, 0, 4, 0, 0, 0, 0, 0},
22          {0, 0, 1, 0, 0, 0, 6, 4, 0},
23          {0, 0, 0, 0, 0, 0, 2, 0, 0},
24          {0, 0, 0, 0, 0, 0, 0, 0, 0},
25          {0, 0, 0, 0, 0, 0, 0, 0, 0},
26          {0, 0, 0, 0, 0, 0, 0, 0, 6} };
27
28  //定义结构体表示操作
29  struct Do {
30      int val; // value
31      int row, col; // position
32      int l, r, u, d; // [-1:<] | [0:\] | [1:>]
33      bool curdom[SIZE]; // 1~SIZE's availablity
34      bool assigned;
35  };
36
37  struct futoshiki
38  {
39      Do board[SIZE][SIZE];
40      void initial(void);
41      bool RowCheck(futoshiki* board, Do* m);
42      bool ColCheck(futoshiki* board, Do* m);
43      bool NeiCheck(futoshiki* board, Do* m);
44      int CDcount(Do* m);
45      bool Goal(futoshiki* board);
```

```
46      bool FCCheck(futoshiki* board, int c, Do* m);
47      void Copyboard(futoshiki* dest, const futoshiki* src);
48      Do* heuristicpick(futoshiki* board);
49      void propagete(futoshiki* board, Do* m);
50      void addConstraints(int x, int y, int x1, int y1);
51  };
52
53  bool FC(futoshiki* board, int level);
54  void propagate(futoshiki* board, Do* m);
55  void display(futoshiki* board);
56
57
58  int main() {
59      futoshiki FTSK;
60      futoshiki* ptr = &FTSK;
61      ptr->initial();
62
63      display(ptr);
64      FC(ptr, 0);
65      cout << nodes << endl;
66      return 0;
67  }
68
69  /*
        ##########################################################################
        */
70
71  void futoshiki::addConstraints(int x, int y, int x1, int y1)
72  {
73
74      if (x == x1)
75      {
76          if (y1 < y)
```

```
77          {
78              board[x][y].l = -1;
79              board[x1][y1].r = 1;
80          }
81          else
82          {
83              board[x][y].r = -1;
84              board[x1][y1].l = 1;
85          }
86      }
87      else
88      {
89          if (x < x1)
90          {
91              board[x][y].u = -1;
92              board[x1][y1].d = 1;
93          }
94          else
95          {
96              board[x][y].u = 1;
97              board[x1][y1].d = -1;
98          }
99      }
100
101 }
102 void futoshiki::initial(void)
103 {
104     for (int i = 0; i < SIZE; i++) {
105         for (int j = 0; j < SIZE; j++) {
106             board[i][j].val = maps[i][j];
107             board[i][j].row = i;
108             board[i][j].col = j;
109             board[i][j].u = 0;
```

```
110                 board[i][j].d = 0;
111                 board[i][j].l = 0;
112                 board[i][j].r = 0;
113                 board[i][j].assigned = maps[i][j]==0? false:true;
114                 memset(board[i][j].curdom, 0, sizeof(board[i][j].curdom));
115             }
116         }
117         //添加限制
118         addConstraints(0, 0, 0, 1);
119         addConstraints(0, 3, 0, 2);
120         addConstraints(1, 3, 1, 4);
121         addConstraints(1, 6, 1, 7);
122         addConstraints(2, 6, 1, 6);
123         addConstraints(2, 1, 2, 0);
124         addConstraints(2, 2, 2, 3);
125         addConstraints(2, 3, 3, 3);
126         addConstraints(3, 3, 3, 2);
127         addConstraints(3, 5, 3, 4);
128         addConstraints(3, 5, 3, 6);
129         addConstraints(3, 8, 3, 7);
130         addConstraints(4, 1, 3, 1);
131         addConstraints(4, 5, 3, 5);
132         addConstraints(4, 0, 4, 1);
133         addConstraints(5, 4, 4, 4);
134         addConstraints(5, 8, 4, 8);
135         addConstraints(5, 1, 5, 2);
136         addConstraints(5, 4, 5, 5);
137         addConstraints(5, 7, 5, 6);
138         addConstraints(5, 1, 6, 1);
139         addConstraints(6, 6, 5, 6);
140         addConstraints(6, 8, 5, 8);
141         addConstraints(6, 3, 6, 4);
142         addConstraints(7, 7, 6, 7);
```

```
143     addConstraints(7, 1, 8, 1);
144     addConstraints(8, 2, 7, 2);
145     addConstraints(7, 5, 8, 5);
146     addConstraints(8, 8, 7, 8);
147     addConstraints(8, 5, 8, 6);
148 
149 }
150 bool RowCheck(futoshiki* board, Do* m) {
151     // return false: constraint falsified;
152     //         true: NO falsification.
153     int Row[SIZE];
154     for (int i = 0; i < SIZE; i++) {
155         Row[i] = board->board[m->row][i].val;
156     }
157     // Check Constraints
158     sort(Row, Row + SIZE);
159     for (int i = 0; i < SIZE - 1; i++) {
160         if (!Row[i]) continue;
161         if (Row[i] == Row[i + 1]) return false;
162     }
163     return true;
164 }
165 
166 bool ColCheck(futoshiki* board, Do* m) {
167     // return false: constraint falsified;
168     //         true: NO falsification.
169     int Col[SIZE];
170     for (int i = 0; i < SIZE; i++) {
171         Col[i] = board->board[i][m->col].val;
172     }
173     // Check Constraints
174     sort(Col, Col + SIZE);
175     for (int i = 0; i < SIZE - 1; i++) {
```

```
176             if (!Col[i]) continue;
177             if (Col[i] == Col[i + 1]) return false;
178         }
179         return true;
180     }
181
182     bool NeiCheck(futoshiki* board, Do* m) {
183         int v = m->val;
184         // Check Constraints
185         // UP
186         if (m->u && board->board[m->row - 1][m->col].assigned) {
187             if (m->u == -1 && v > board->board[m->row - 1][m->col].val)
                    return false;
188             else if (m->u == 1 && v < board->board[m->row - 1][m->col].val)
                        return false;
189         }
190         // DOWN
191         if (m->d && board->board[m->row + 1][m->col].assigned) {
192             if (m->d == -1 && v > board->board[m->row + 1][m->col].val)
                    return false;
193             else if (m->d == 1 && v < board->board[m->row + 1][m->col].val)
                        return false;
194         }
195         // LEFT
196         if (m->l && board->board[m->row - 1][m->col].assigned) {
197             if (m->l == -1 && v > board->board[m->row][m->col - 1].val)
                    return false;
198             else if (m->l == 1 && v < board->board[m->row][m->col - 1].val)
                        return false;
199         }
200         // RIGHT
201         if (m->r && board->board[m->row - 1][m->col].assigned) {
```

```
202         if (m->r == -1 && v > board->board[m->row][m->col + 1].val)
                return false;
203         else if (m->r == 1 && v < board->board[m->row][m->col + 1].val)
                return false;
204     }
205     return true;
206 }
207
208 int CDcount(Do* m) {
209     int res = 0;
210     for (int i = 0; i < SIZE; i++) {
211         res += m->curdom[i];
212     }
213     return res;
214 }
215
216 bool CheckCons(futoshiki* board, int c, Do* m) {
217     int R = m->row, C = m->col, tot = 0;
218     // c == 0 >>> row
219     if (c == 0) {
220         for (int i = 0; i < SIZE; i++) {
221             tot += board->board[R][i].assigned;
222         }
223         return (tot == SIZE - 1);
224     }
225     // c == 1 >>> col
226     else if (c == 1) {
227         for (int i = 0; i < SIZE; i++) {
228             tot += board->board[i][C].assigned;
229         }
230         return (tot == SIZE - 1);
231     }
232     // c == 2 >>> neighbour
```

```
233     else if (c == 2) {
234         tot += (R == 0) ? 1 : board->board[R - 1][C].assigned;
235         tot += (R == SIZE - 1) ? 1 : board->board[R + 1][C].assigned;
236         tot += (C == 0) ? 1 : board->board[R][C - 1].assigned;
237         tot += (C == SIZE - 1) ? 1 : board->board[R][C + 1].assigned;
238         return (tot == SIZE - 1);
239     }
240     return false;
241 }
242
243 bool FCCheck(futoshiki* board, int c, Do* m) {
244     // c == 0 >>> row
245     if (c == 0) for (int i = 0; i < SIZE; i++) {
246         if (!m->curdom[i]) {
247             m->curdom[i] = 1;
248             if (RowCheck(board, m)) m->curdom[i] = 0; // No
                falsification
249         }
250     }
251     // c == 1 >>> col
252     else if (c == 1) for (int i = 0; i < SIZE; i++) {
253         if (!m->curdom[i]) {
254             m->curdom[i] = 1;
255             if (ColCheck(board, m)) m->curdom[i] = 0; // No
                falsification
256         }
257     }
258     // c == 2 >>> neighbour
259     else if (c == 2) for (int i = 0; i < SIZE; i++) {
260         if (!m->curdom[i]) {
261             m->curdom[i] = 1;
262             if (NeiCheck(board, m)) m->curdom[i] = 0; // No
                falsification
```

```
263             }
264         }
265         if (CDcount(m) == SIZE) return false;
266         else return true;
267     }
268
269     bool Goal(futoshiki* board) {
270         for (int i = 0; i < SIZE; i++) {
271             for (int j = 0; j < SIZE; j++) {
272                 if (!board->board[i][j].assigned) return false;
273             }
274         }
275         return true;
276     }
277
278     Do* heuristicpick(futoshiki* board) {
279         // MRV
280         Do* maxi = &board->board[0][0];
281         for (int i = 0; i < SIZE; i++) {
282             for (int j = 0; j < SIZE; j++) {
283                 if (board->board[i][j].assigned) continue;
284                 if (CDcount(maxi) < CDcount(&board->board[i][j]) || maxi->
                    assigned) {
285                     maxi = &board->board[i][j];
286                     if (CDcount(maxi) == SIZE - 1) return maxi;
287                 }
288             }
289         }
290         return maxi;
291     }
292
293     void propagate(futoshiki* board, Do* m) {
294         // cout<<"P: "<<m->val<<endl;
```

```
295     for (int i = 0; i < SIZE; i++) {//同行同列不能取
296         board->board[m->row][i].curdom[m->val - 1] = 1;
297         board->board[i][m->col].curdom[m->val - 1] = 1;
298     }
299     if (m->r == -1) {//约束下的不能取
300         for (int i = 0; i < m->val - 1; i++) {
301             board->board[m->row][m->col + 1].curdom[i] = 1;
302         }
303     }
304     else if (m->r == 1) {
305         for (int i = m->val; i < SIZE; i++) {
306             board->board[m->row][m->col + 1].curdom[i] = 1;
307         }
308     }
309     if (m->u == -1) {
310         for (int i = 0; i < m->val - 1; i++) {
311             board->board[m->row - 1][m->col].curdom[i] = 1;
312         }
313     }
314     else if (m->u == 1) {
315         for (int i = m->val; i < SIZE; i++) {
316             board->board[m->row - 1][m->col].curdom[i] = 1;
317         }
318     }
319     if (m->l == -1) {
320         for (int i = 0; i < m->val - 1; i++) {
321             board->board[m->row][m->col - 1].curdom[i] = 1;
322         }
323     }
324     else if (m->l == 1) {
325         for (int i = m->val; i < SIZE; i++) {
326             board->board[m->row][m->col - 1].curdom[i] = 1;
327         }
```

```
328     }
329     if (m->d == -1) {
330         for (int i = 0; i < m->val - 1; i++) {
331             board->board[m->row + 1][m->col].curdom[i] = 1;
332         }
333     }
334     else if (m->d == 1) {
335         for (int i = m->val; i < SIZE; i++) {
336             board->board[m->row + 1][m->col].curdom[i] = 1;
337         }
338     }
339 }
340 
341 void Copyboard(futoshiki* dest, const futoshiki* src) {
342     memcpy(dest, src, sizeof(futoshiki));
343 }
344 
345 bool FC(futoshiki* board, int level)
346 {
347     nodes++;
348     if (Goal(board))
349     {//找到目标
350         display(board);
351         return true;
352     }
353     Do* v = heuristicpick(board); // Pick with MRV
354     v->assigned = true;
355 
356     bool dwo = false;
357     int pos = 0;
358     for (int i = 0; i < SIZE; i++) if (!v->curdom[i]) {
359         futoshiki boardcopy;
360         Copyboard(&boardcopy, board);
```

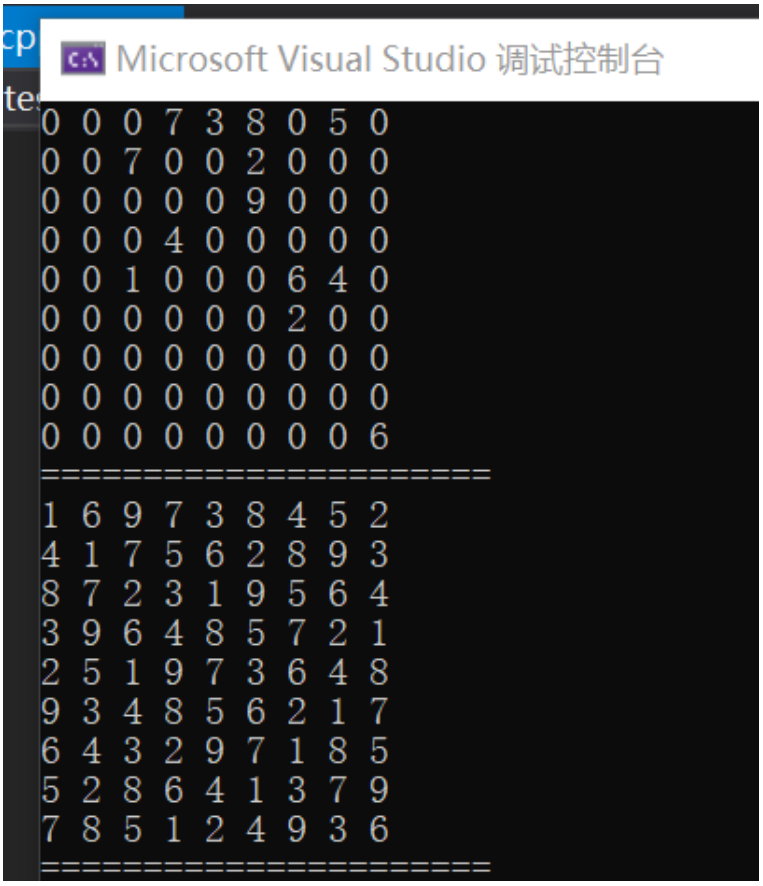
```
361             v->val = i + 1;
362             propagate(board, v);
363             dwo = false;
364             // row constraint
365             if (!dwo && CheckCons(board, 0, v)) {
366                 for (int i = 0; i < SIZE; i++) if (!board->board[v->row][i
                        ].assigned) {
367                     dwo = !FCCheck(board, 0, &board->board[v->row][i]);
368                 }
369             }
370             // col constraint
371             if (!dwo && CheckCons(board, 1, v)) {
372                 for (int i = 0; i < SIZE; i++) if (!board->board[i][v->col
                        ].assigned) {
373                     dwo = !FCCheck(board, 1, &board->board[i][v->col]);
374                 }
375             }
376             // neighbour constraint
377             if (!dwo && CheckCons(board, 2, v)) {
378                 if (v->row && board->board[v->row - 1][v->col].assigned) {
379                     dwo = !FCCheck(board, 2, &board->board[v->row - 1][v->
                            col]);
380                 }
381                 else if (v->row != SIZE - 1 && board->board[v->row + 1][v->
                        col].assigned) {
382                     dwo = !FCCheck(board, 2, &board->board[v->row + 1][v->
                            col]);
383                 }
384                 else if (v->col && board->board[v->row][v->col - 1].
                        assigned) {
385                     dwo = !FCCheck(board, 2, &board->board[v->row][v->col -
                            1]);
386                 }
```

```
387             else if (v->col != SIZE - 1 && board->board[v->row][v->col
                    + 1].assigned) {
388                 dwo = !FCCheck(board, 2, &board->board[v->row][v->col +
                        1]);
389             }
390         }
391         if (!dwo && FC(board, level + 1)) return true;
392         Copyboard(board, &boardcopy);
393     }
394     v->assigned = false;
395     return false;
396 }
397
398 void display(futoshiki* board) {
399     for (int i = 0; i < SIZE; i++) {
400         cout << "\t";
401         for (int j = 0; j < SIZE; j++) {
402             if (board->board[i][j].r == 1) cout << " " << board->board[
                    i][j].val << " >";
403             else if (board->board[i][j].r == -1) cout << " " << board->
                    board[i][j].val << " <";
404             else cout << " " << board->board[i][j].val << "  ";
405         }
406         cout << endl << "\t";
407         if (i != SIZE - 1) {
408             for (int j = 0; j < SIZE; j++) {
409                 if (board->board[i + 1][j].u == 1) {
410                     cout << " ^  ";
411                 }
412                 else if (board->board[i + 1][j].u == -1) {
413                     cout << " v  ";
414                 }
415                 else cout << "    ";
```

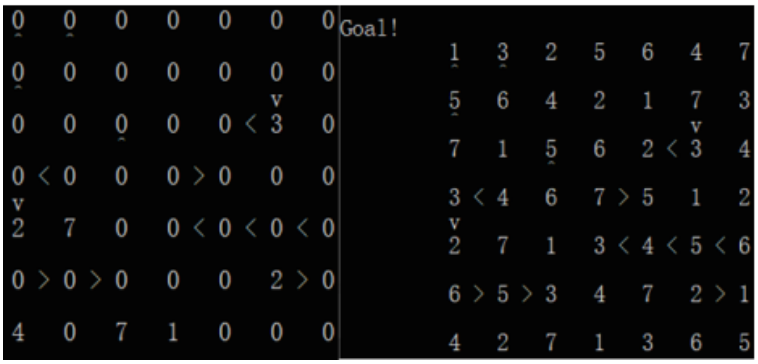
```
416                 }
417             }
418             cout << endl;
419         }
420 }
```

5 Results

这里可以看到当SIZE为9，也就是参考代码给出的puzzle，结果为：



加入对大于等于号的输入，可以有更人性化的输出。



6 What I learnt

1. 首先是对于回溯算法的实现有了更深的理解，而约束传播的两种方式，一个是FC向前检测，一个是GAC算法。这回实现的FC算法是通过在赋值之前先把不可能的值排除，可以缩减搜索树的规模

2. 值得注意的是，每一次需要保存赋值之前的board的状态，否则在回溯的时候无法恢复到原来的样子是不可取的。回溯的思想是推翻之前重来，如果回不到之前的状态会有错误的删减值域。

3. tradeoff：当我们用检测缩减值域的时候，虽然减少搜索空间，但是搜索速度是变慢了的，原因就在于对domain的维护开销