不围棋 AI 设计展示

董海辰 518030910417

2018 年 11 月 13 日

目录

1 基本信息

AI 代号 White Album 2(下简称 WA2)

代码长度 9.38KB

关键算法 Monte Carlo 树搜索及其优化

2 设计思路

在传统的 Monte Carlo 树搜索 (MCTS) 中, 将棋盘状态以落子转移保存为树结构, 对于一个节点进行一定次数的随机模拟, 来判断一个节点及其父节点的价值 (获胜概率).

对于一个局面随机模拟的庞大计算量限制了模拟次数, 导致在每次估价过程中没有足够的数据来判断价值, 成为 MCTS 算法的主要瓶颈, 因而 WA2 的主要设计思路即是增加每个节点的样本数量.

3 具体实现

3.1 完整代码

```
1  //
2  //  NoGo.cpp
3  //  AI_NoGo
4  //
5  //  Created by Haichen Dong on 2018/10/23.
6  //  Copyright © 2018 Haichen Dong. All rights reserved.
7  //
8  //  Version ONLINE_JUDGE with TL=1.5s
9  //
10
11 #pragma GCC optimize ("O3")
12
13 #include "submit.h"
14 #include <bits/stdc++.h>
```

```
15 using namespace std;
16
17 extern int ai_side;
18 std::string ai_name = "董海辰";
19 int PRINTFLAG;
20 const int TIMELIMIT = CLOCKS_PER_SEC / 20 * 25;
21 const double K = 1000.0;
22
23 struct Status {
24     unsigned long long a[3];
25     int color, exd, mvx, mvy, lson, rson, fa;
26     int n, n1, w, w1;
27     map<short,int> son;
28     inline Status() {
29         son.clear();
30         exd = 0;
31     }
32     inline void getBoard (int board[9][9]) {
33         unsigned long long _a[3];
34         for (int i = 0; i < 3; i++) _a[i]=a[i];
35         for (int k = 0; k < 3; k++) {
36             for (int i = 0; i < 3; i++) {
37                 for (int j = 0; j < 9; j++) {
38                     board[i + k*3][j] = _a[k] & 3;
39                     _a[k] >>= 2;
40                 }
41             }
42         }
43     }
44     inline void init (int _color,int board[9][9]) {
45         color = _color;
46         a[0] = a[1] = a[2] = 0;
47         for (int k = 2; k >= 0; k--) {
48             for (int i = 2; i >= 0; i--) {
```

```
                for (int j = 8; j >=0 ; j--) {
                    a[k] <<= 2;
                    a[k] += board[i + k*3][j];
                }
            }
        }
    }
} sTree[(int)1e6];
const int dx[]= {0, 0, 1, -1}, dy[]= {1, -1, 0, 0};
queue< pair<int,int> > q, qq, qqq;
int bk[9][9], ok[9][9], a[9][9], qc[9][9], tmp[9][9], q1, q2;
inline void bfs (int sx, int sy, int type) {
    int qiCnt = 0, qix = 0, qiy = 0;
    bk[sx][sy] = 1;
    while (!q.empty())
        q.pop();
    while (!qq.empty())
        qq.pop();
    while (!qqq.empty())
        qqq.pop();
    q.push(make_pair(sx, sy));
    while (!q.empty()) {
        int cux = q.front().first, cuy = q.front().second;
        qq.push(q.front());
        q.pop();
        for (int k = 0; k < 4; k++) {
            int nex = cux + dx[k], ney = cuy + dy[k];
            if (nex < 0 || nex > 8 || ney < 0 || ney > 8)
                continue;
            if (!a[nex][ney]) {
                if (!tmp[nex][ney]) {
                    tmp[nex][ney] = 1;
                    qiCnt++, qix = nex, qiy = ney;
                    qqq.push(make_pair(nex, ney));
```

```
                }
            } else if (a[nex][ney] == a[sx][sy]) {
                if (!bk[nex][ney]) {
                    bk[nex][ney] = 1;
                    q.push(make_pair(nex, ney));
                }
            }
        }
    }
    if (type) {
        q2 += qiCnt;
        if (qiCnt == 1)
            ok[qix][qiy] = 0;
    } else {
        q1 += qiCnt;
        while (!qq.empty()) {
            qc[qq.front().first][qq.front().second] = qiCnt;
            qq.pop();
        }
    }
    while (!qqq.empty()) {
        tmp[qqq.front().first][qqq.front().second] = 0;
        qqq.pop();
    }
}
vector< pair<int,int> > possibleVec, tmpvec;
inline pair<int,int> findPossiblePos (int color, vector< pair<int,
    int> > &vcr = possibleVec) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            bk[i][j] = 0;
            qc[i][j] = 0;
            if (!a[i][j])
                ok[i][j] = 1;
```

```
116             else
117                 ok[i][j] = 0;
118         }
119     }
120     for (int i = 0; i < 9; i++) {
121         for (int j = 0; j < 9; j++) {
122             if (a[i][j] && !bk[i][j]) {
123                 bfs(i, j, a[i][j] == 3-color);
124             }
125         }
126     }
127     for (int i = 0; i < 9; i++) {
128         for (int j = 0; j<9; j++) {
129             if (!a[i][j]) {
130                 int qcu = 0;
131                 for (int k = 0; k < 4; k ++) {
132                     int nex = i + dx[k], ney = j + dy[k];
133                     if (nex < 0 || nex > 8 || ney < 0 || ney > 8)
134                         continue;
135                     if (a[nex][ney] == color) {
136                         qcu = qcu + qc[nex][ney] - 1;
137                     } else if (!a[nex][ney]) {
138                         qcu = 100;
139                     }
140                 }
141                 if (!qcu)
142                     ok[i][j] = 0;
143             }
144         }
145     }
146     vcr.clear();
147     for (int i = 0; i < 9; i++) {
148         for (int j = 0; j < 9; j++) {
149             if (ok[i][j])
```

```
150                 vcr.push_back(make_pair(i, j));
151         }
152     }
153     if (!vcr.size())
154         return make_pair(-1, -1);
155     swap(vcr[rand() % vcr.size()], vcr[0]);
156     return vcr[0];
157 }
158
159 inline int simulate (Status s) {
160     int curColor = 3 - s.color;
161     s.getBoard(a);
162     while (1) {
163         pair<int,int> pos = findPossiblePos(curColor);
164         if (pos.first == -1) {
165             return 3 - curColor;
166         }
167         a[pos.first][pos.second] = curColor;
168         curColor = 3 - curColor;
169     }
170 }
171
172 double _beta[200005];
173 inline double cal (int k, int flag) {
174     double beta = _beta[sTree[k].n];
175     return (1.0 - beta) * sTree[k].w / sTree[k].n + beta * sTree[k
           ].w1 / sTree[k].n1;
176 }
177 inline int getBestSon (int k, int flag) {
178     double ma = 0;
179     int mapo = sTree[k].lson;
180     for (int i = sTree[k].lson, rs = sTree[k].rson; i <= rs; i++) {
181         double cu = cal(i, flag);
182         if (cu > ma)
```

```
183             ma = cu, mapo = i;
184         if (sTree[i].n < 2)
185             return i;
186     }
187     return mapo;
188 }
189
190 int tot = 1, CNT = 0;
191 unsigned int startClock;
192 vector<int> vs;
193 vector< pair<pair<int,int>,int> > va;
194 pair<int,int> search (Status s0) {
195     sTree[1] = s0;
196     tot = 1;
197     while (clock()-startClock<TIMELIMIT) {
198         CNT++;
199         int cur = 1, T = 0, t = 0, win = 0;
200         vs.clear();
201         va.clear();
202         while (1) {
203             if (cur == 0)
204                 break;
205             if (!sTree[cur].exd) {
206                 sTree[cur].getBoard(a);
207                 findPossiblePos(3 - sTree[cur].color, possibleVec);
208                 if (!possibleVec.size()) {
209                     win = sTree[cur].color;
210                     break;
211                 }
212                 sTree[cur].exd = 1;
213                 sTree[cur].lson = tot + 1;
214                 for (int i = 0, sz = possibleVec.size(); i < sz; i
                        ++) {
```

```
215                     int nx = possibleVec[i].first, ny = possibleVec
                            [i].second;
216                     sTree[++tot] = Status();
217                     sTree[cur].son.insert(make_pair((short)nx * 9 +
                             ny, tot));
218                     sTree[tot].fa = cur;
219                     sTree[tot].mvx = nx;
220                     sTree[tot].mvy = ny;
221                     a[nx][ny] = 3 - sTree[cur].color;
222                     sTree[tot].init(3 - sTree[cur].color, a);
223                     q1 = q2 = 0;
224                     findPossiblePos(sTree[cur].color, tmpvec);
225                     sTree[tot].n = sTree[tot].n1 = q1 + q2 + tmpvec
                            .size() / 4;
226                     sTree[tot].w = sTree[tot].w1 = q2;
227                     a[nx][ny] = 0;
228                 }
229                 sTree[cur].rson = tot;
230                 break;
231             } else {
232                 vs.push_back(cur);
233                 cur = getBestSon(cur, 0);
234                 va.push_back(make_pair(make_pair(sTree[cur].mvx,
                        sTree[cur].mvy), sTree[cur].color));
235                 T++;
236             }
237         }
238         t = T;
239         int curColor = 3 - sTree[cur].color;
240         sTree[cur].getBoard(a);
241         while (1) {
242             pair<int,int> pos = findPossiblePos(curColor);
243             if (pos.first == -1) {
244                 win = 3 - curColor;
```

```
245                 break;
246             }
247             va.push_back(make_pair(make_pair(pos.first, pos.second)
                    , curColor));
248             t++;
249             a[pos.first][pos.second] = curColor;
250             curColor = 3 - curColor;
251         }
252 
253         for (int i = 0; i < T; i++) {
254             int uu = vs[i];
255             sTree[uu].n++;
256             if (sTree[uu].color == win)
257                 sTree[uu].w++;
258             int ff = sTree[uu].fa;
259             if (i == 0)
260                 continue;
261             for (int j = i - 1; j < t; j++) {
262                 if (sTree[ff].color != va[j].second) {
263                     int k = sTree[ff].son[(short)(va[j].first.first
                            * 9 + va[j].first.second)];
264                     if (k) {
265                         sTree[k].n1++;
266                         if (sTree[k].color == win)
267                             sTree[k].w1++;
268                     }
269                 }
270             }
271             if (clock() - startClock > TIMELIMIT)
272                 break;
273         }
274     }
275     int bss = getBestSon(1, 1);
276     int rex = sTree[bss].mvx, rey = sTree[bss].mvy;
```

```
277     for (int i = 1; i <= tot; i++)
278         sTree[i] = Status();
279     return make_pair(rex, rey);
280 }
281
282 void init() {
283     srand(time(NULL));
284     for (int i = 0; i <= 200000; i++)
285         _beta[i] = sqrt(K / (K + 3 * i));
286 }
287
288 int bd[9][9];
289 void GetUpdate (std::pair<int, int> location) {
290     bd[location.first][location.second] = 2 - ai_side;
291 }
292
293 std::pair<int, int> Action() {
294     startClock = clock();
295     Status ss = Status();
296     ss.init(2 - ai_side, bd);
297     ss.getBoard(a);
298     findPossiblePos(ai_side + 1);
299     for (int i = 0, sz = possibleVec.size(); i < sz; i++) {
300         if (possibleVec[i].first == 0 && possibleVec[i].second ==
              0)
301             return bd[0][0] = ai_side + 1, make_pair(0, 0);
302         if (possibleVec[i].first == 8 && possibleVec[i].second ==
              0)
303             return bd[8][0] = ai_side + 1, make_pair(8, 0);
304         if (possibleVec[i].first == 0 && possibleVec[i].second ==
              8)
305             return bd[0][8] = ai_side + 1, make_pair(0, 8);
306         if (possibleVec[i].first == 8 && possibleVec[i].second ==
              8)
```

```
307             return bd[8][8] = ai_side + 1, make_pair(8, 8);
308     }
309     CNT = 0;
310     pair<int,int> ret = search(ss);
311     bd[ret.first][ret.second] = ai_side + 1;
312     return ret;
313 }
```

3.2 随机模拟

3.3 更新节点