2021 年春季学期
数据结构课程设计 A3 题实验报告

张津赫 *[1], 曹伟[1], 张天浩[1],

软件学院 2019 级 8 班

摘要

利用极大极小树与 alpha-beta 剪枝算法结合, 限定层数为 6 层，并采用 $c++$ STL 类的 list 类型，对蛇体信息进行存储, 该算法对比贪心方法，简单评估函数有很大提升。

关键词（alpha-beta 剪枝，缓存增强试探法，散列表）

目录

第一章　分工与合作

1. 张津赫：参与实现 alpha-beta 剪枝函数，细节修改，mtdf 实现，bug 调试。

2. 张天浩：主要负责 alpha-beta 剪枝函数实现，bug 调试。

3. 曹伟：参与实现 alpha-beta 剪枝函数，负责评估函数的实现，bug 调试。

- 4.17 讨论简单的贪心算法，算法局限性较大，难以提高效率。
- 4.18 转而简单评估函数，胜率较低。
- 4.20 选择 alpha-beta 剪枝 + 缓存增强试探法 + 散列表，进行研讨，开始着手进行代码编写。
- 成功完成代码，并成功运行。

第二章　算法思想

2.1　总体思路

本题程序主要使用了 Alpha-Beta 剪枝算法，Alpha-Beta 剪枝算法是对 Minimax 方法的优化，它们产生的结果是完全相同的，只不过运行效率不一样。Alpha-Beta 只能用递归实现。这个思想是在搜索中传递两个值，第一个值是 Alpha，即搜索到的最好值，任何比它更小的值就没用了，因为策略就是知道 Alpha 的值，任何小于或等于 Alpha 的值都不会有所提高。

如果某个着法的结果大于 Alpha 但小于 Beta，那么这个着法就是走棋一方可以考虑走的，除非以后有所变化。因此 Alpha 会不断增加以反映新的情况。有时候可能一个合理着法也不超过 Alpha，这在实战中是经常发生的，此时这种局面是不予考虑的，因此为了避免这样的局面，我们必须在博弈树的上一个层局面选择另外一个着法。

这个算法严重依赖于着法的寻找顺序。如果你总是先去搜索最坏的着法，那么 Beta 截断就不会发生，因此该算法就如同最小-最大一样，效率非常低。该算法最终会找遍整个博弈树，就像最小-最大算法一样。在 AlphaBeta 剪枝算法中，因为存在剪枝，搜索过程并不完整，所以剪枝算法所得到的估值不一定是准确值。由此可看出，alpha-beta 剪枝算法重点关注的是 (alpha,beta) 区间，它也称为搜索窗口。相对于 alpha 节点和 beta 节点而言，程序搜索 PV 节点的过程是比较费时的。如果把搜索窗口的宽度减小，发生剪枝的可能性就会加大，程序搜索的时间也会短些。在选择每一次的试探值时，通常的想法是采用二分法，即每次都选择估值区间的中点进行试探。采用二分法的效率确实比较高，但是荷兰计算机科学家 Aske Plaat 经研究发现，在多次搜索试探过程中，局面估值往往会出现在前一次搜索的返回值附近，而机械地选取中点做为新的试探值并不是最高效的。于是他提出了一种选取前一次搜索返回值做为新试探值的 MTD 算法（Memory-enhanced Test Driver，缓存增强试探法）——MTD(f)。

增加可搜索的节点数量。蛇的位置采用 c++ STL 类的 list 类型进行实时更新。该算法从网站得到基本的结构，随后根据网站内呈现的基本结构逐步扩充代码，完成了蛇的移动与溯源，估值函数等功能，最后完成代码的编写。

2.2　所用方法的特别、新颖或创新之处

无

2.3 算法流程图

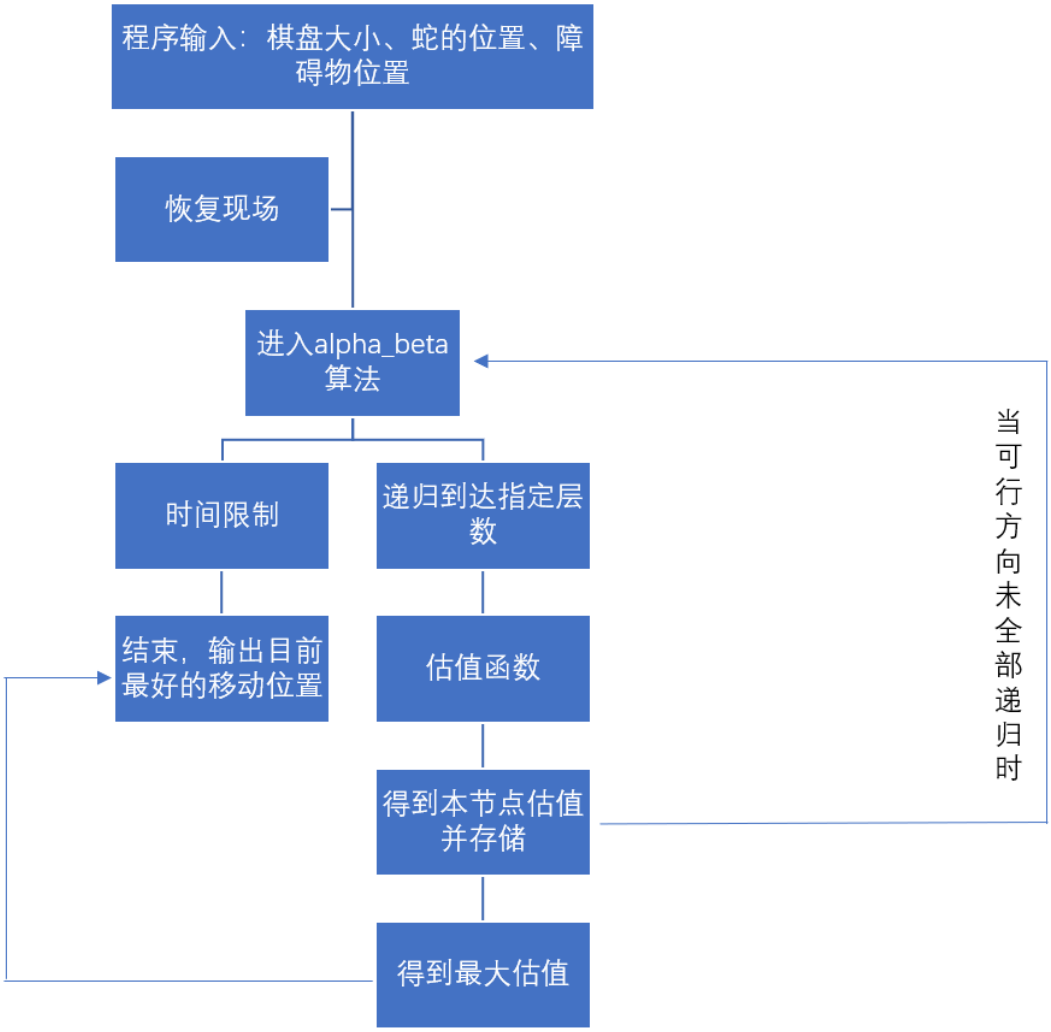

图 2.1: 总图

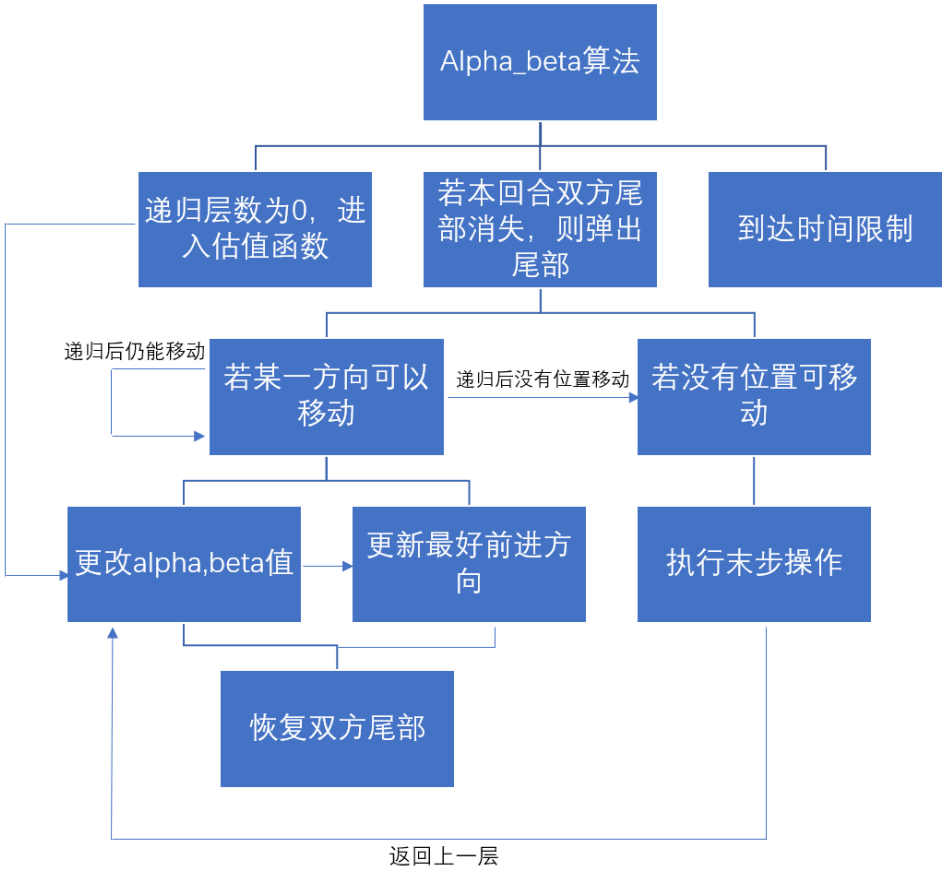

图 2.2: alpha-beta 算法分图

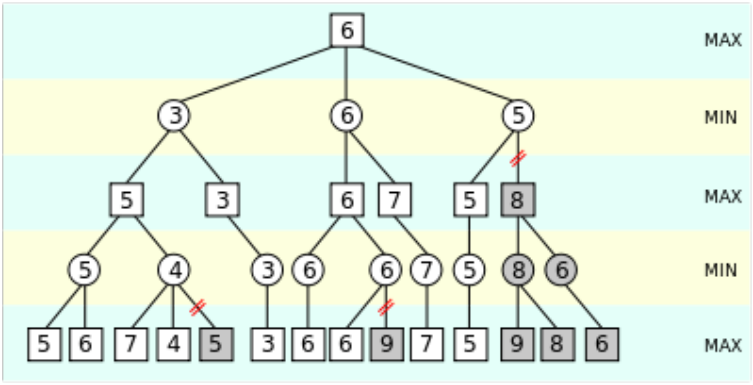

图 2.3: 算法示意

2.4 算法运行时间复杂度分析

Alpha-Beta 剪枝算法用于减小极大极小算法所搜索的节点数目，Alpha-Beta 剪枝算法的效率很大依赖于节点的排列，在理想的排序下, 由

$$O\left(\left(\left(b-1+\sqrt{b^2+14b+1}\right)/4\right)^d\right)$$

算法复杂度为 $O\left(b^{d/2}\right)$, 可以使搜索节点的数量减小一半，从而使在相同时间下的搜索深度增加一倍。在随机排序下，算法复杂度平均为 $O\left(b^{d*3/4}\right)$ 最坏情况下，没有节点会被剪枝，整个树都会被检视，每个节点会检视 b^2 个子节点，最好情况下，每个结点都会检视 $2b-1$ 个子节点。随意总的算法将会检验 $O\left(b^{d/2}\right)$ 个节点。

d: 树的深度

b: 分支因子，是每个结点下的子结点数，即出度。

第三章　程序代码说明

3.1　数据结构说明

主要数据结构：

1. 采用了 $c++$ STL 类的 list 类型，对蛇体信息进行存储。

2. 用若干二维数组保存地图信息。

3. 用递归函数及回溯操作模拟博弈树，对博弈情况进行模拟和评估。

3.2　函数说明

```
1. int evaluate(int total):
    int evaluate(int total)
{
    int dist1[25][25] = { 0 }, dist2[25][25] = { 0 }, temp_map[25][25];
    int st[25][25] = { 0 };

    for(int i=0;i<25;i++)
        for (int j = 0; j < 25; j++)
        {
            dist1[i][j] = 9999999;
            dist2[i][j] = 9999999;
            st[i][j] = 0;
        }
//对数组进行初始化st[数组]存放当前位置的点（不为零），表示蛇身体某点再经过几回合可以空出来。（存放回合数），dist1存放我方蛇局面评估信息，表示蛇头走到某点的回合数（dist2对方同理），dist1/2初始化为无穷大的目的：距离蛇头较近的点不一定先走，所以初始化每个点都无法达到。
    list<point>::iterator it0 = snake[0].end();
    list<point>::iterator it1 = snake[1].end();
    it0--;
    it1--;
    int k = 0;
    for (k = total - snake[0].size() + 1; k <= total; k++)
//k表示当前是第几个不增长的回合。
    {

```

```
24          st[it0->x][it0->y] = disteps[k] - total;
25  //disteps相当于查表，disteps[i]内存放第i个不增长的回合
26          st[it1->x][it1->y] = disteps[k] - total;
27          if (it0 != snake[0].begin())
28              it0--;
29          if (it1 != snake[1].begin())
30              it1--;
31
32      }
33  //st[i][j]存放蛇身点（i,j）经过几回合后可以被使用（走）。
34      general_map(temp_map);
35  //形成地图信息
36      for (int id = 0; id <= 1; id++)
37      {
38          point head(snake[id].front());
39
40          int visit[25][25] = { 0 };
41
42          visit[head.x][head.y] = 1;
43          if (id == 0)
44              dist1[snake[id].front().x][snake[id].front().y] = 0;
45          else
46              dist2[snake[id].front().x][snake[id].front().y] = 0;
47
48          deque<point> now_try;
49  //使用队列完成对于所有待评估点的遍历
50          now_try.push_back(head);
51
52
53          if (id == 0)
54  //对我方局面进行评估
55          {
56              while (!now_try.empty())
57              {
58                  point new_p(*now_try.begin());
59                  now_try.pop_front();
60                  for (int d = 0; d < 4; d++)
61                  {
62                      point p1;
63                      p1.x = new_p.x + dx[d];
64                      p1.y = new_p.y + dy[d];
65
66  if (dist1[p1.x][p1.y] > dist1[new_p.x][new_p.y] + 1 && temp_map[p1.x][p1.y] != 1 && st[p1.x][p1.y] <= dist1[new_p.x][
        new_p.y] + 1)//
67  //st[p1.x][p1.y] <= dist1[new_p.x][new_p.y] + 1 表示当前蛇身的某点在经过dist[][]+1回合后空出来，可以使用。
68                      {
69                          now_try.push_back(p1);
70                          dist1[p1.x][p1.y] = dist1[new_p.x][new_p.y] + 1;
71
72                      }
73                  }
74
75              }
76          }
77
```

```
78          if (id == 1)
79          {
80              while (!now_try.empty())
81              {
82                  point new_p(*now_try.begin());
83                  now_try.pop_front();
84                  for (int d = 0; d < 4; d++)
85                  {
86                      point p1;
87                      p1.x = new_p.x + dx[d];
88                      p1.y = new_p.y + dy[d];
89
90  if (dist2[p1.x][p1.y] > dist2[new_p.x][new_p.y] + 1 && temp_map[p1.x][p1.y] != 1 && st[p1.x][p1.y] <= dist2[new_p.x][
        new_p.y] + 1)//
91                      {
92                          now_try.push_back(p1);
93                          dist2[p1.x][p1.y] = dist2[new_p.x][new_p.y] + 1;
94
95                      }
96                  }
97
98              }
99          }
100
101
102     }
103
104
105
106
107
108     int value1 = 0, value2 = 0;
109     for (int i = 1; i <= n; i++)
110         for (int j = 1; j <= m; j++)
111         {
112             if (dist1[i][j] < dist2[i][j])
113 //我方优势点
114                 value1 += dist_value[dist1[i][j]];//累计优势值
115             else if (dist2[i][j] < dist1[i][j])
116                 value2 += dist_value[dist2[i][j]];
117
118         }
119     return value1 - value2;
120 ////效益数组，越后走到的回合效益值越低
121
122
123
124 };
125
126 2. int alpha_beta(int depth, int id, int alpha, int beta, int total, bool firstpoint, int hash_value)
127 int alpha_beta(int depth, int id, int alpha, int beta, int total, bool firstpoint, int hash_value) {
128 //firstpoint除了调用时赋值一，递归时赋值零，以标记第一层
129     ret["response"]["total"] = total;
130
131
```

```
132     if ((clock() - Time) >= timeout) //时间限制，在超时前结束
133
134     {
135         output(returnvalue);
136         exit(0);
137
138     }
139
140     int now = -1000000000, bestmove = 0;
141     int i, j;
142     bool isdeletetail = 0;
143     if (depth == 0) {//当递归剩余层数为0，进入估值函数
144         return evaluate(total);
145     }
146     //---------------------------------------------
147     int best = hash_move[hash_value & (HASH_SIZE - 1)];
148      //从哈希表里取得最好移动方向
149
150     /////
151
152     point p0, p1;
153     if (!whetherGrow(total)) {//尾部是否消失
154         p0 = snake[0].back();//
155         p1 = snake[1].back();//保存双方的尾部
156     }
157
158     int map[25][25];
159     general_map(map);
160     bool endflag = 1;//如果最后还是一，代表无路可走，调用末步函数
161     if (!id)
162     {
163
164         if (whetherGrow(total) == 0)//若本回合双方尾巴消失
165         {
166             snake[0].pop_back();
167             snake[1].pop_back();//消除尾部
168         }
169     }
170     for (int i = best, j = 0; j < 4; i = (i + 1) & 3, j++)
171     {
172         point p = *(snake[id].begin());//读取目前前进的蛇最近移动的位置
173
174         int snakex = p.x + dx[i];
175         int snakey = p.y + dy[i];
176
177         if (map[snakex][snakey] != 0) continue;//遇到障碍，不可前进到此处
178
179         total = move1(id, i, total);//蛇移动
180         int idnext = (id == 1 ? 0 : 1);
181         endflag = 0;//目前还有某一方向可以走
182
183         int score = -alpha_beta(depth - 1, idnext, -beta, -alpha, total, 0, hash_value * 20011 + i);//递归
184
185         ret["response"]["score"] = score;
186
```

```
187
188         if (id == 1) {//恢复模拟前的回合数
189             total--;
190         }
191         snake[id].pop_front();//弹出模拟时放入的头部
192
193         if (score > now) {
194             bestmove = i;
195             now = score;
196             ret["response"]["now"] = now;
197             if (alpha < score) {//剪枝
198                 alpha = score;
199                 if (firstpoint == 1) {
200                     returnvalue = i;
201                 }
202             }
203             if (score > beta) {//剪枝
204                 break;
205             }
206         }
207     }
208     if (endflag) {//末步函数
209         now = noplace(id);//无路可走\成功处理
210     }
211
212     if (!id) {
213
214         if (whetherGrow(total) == 0)//本回合消除了尾部
215         {
216             snake[0].push_back(p0);
217             snake[1].push_back(p1);//恢复尾部
218
219         }
220     }
221
222
223     hash_move[hash_value & (HASH_SIZE - 1)] = bestmove;//hash
224     return now;
225 }
226
227
228
229
230
231
232
233 /*需要一个初始试探值，一般可以简单地选取初始搜索区间的中点，当然也可以根据具体情况选择对搜索更有利的值。
234 由于该算法必须采用散列表，散列表的性能对于整体算法的效率也是至关重要的。
235 另外，估值的取值范围大小，也会影响MTD(f)算法的搜索速度。
236 一般来说，估值越粗略（即取值范围较小），搜索速度会越快；这是由于试探的次数较少所致。
237 如果算法还涉及最佳棋步的求解，那么最佳棋步的取舍问题应引起注意。
238 虽然MTD(f)每进行一次零宽窗口试探，都会得到一个新的最佳棋步，但对于alpha节点（best_value < test时）。
239 我们所得到的“最佳棋步”只是满足估值上限的棋步，而非真正最佳的棋步。因此，我们应丢弃这种虚假的最佳棋步，而保留上一次搜索结果
240 */
241 3. int mtd(int alpha, int beta, int depth, int test, int total)
```

```
242 {
243     int best_value;
244     do {
245         // 进行零宽窗口试探
246         best_value = alpha_beta(depth, 0, test - 1, test, total,1, 0);
247         // 如果是alpha节点
248         if (best_value < test) {
249             // 更新估值上限，并将此做为新的试探值
250             test = beta = best_value;
251             // 否则是beta节点
252         }
253         else {
254             // 更新估值下限
255             alpha = best_value;
256             // 新的试探值
257             test = best_value + 1;
258         }
259     } while (alpha < beta);
260     return best_value;
261 }
```

3.3 程序限制

若棋盘足够大回合足够多，例如出现某步可能需要蛇头部运行 400 步才能到达时，评估函数由于查表数组大小的限制，会出现失效。

第四章　实验结果

4.1　测试数据

将 json 格式字符串直接输入程序，进行单步执行。

4.2　结果分析

- alpha-beta 剪枝操作（主要时间开销）
- 缓存增强试探：大大提高了算法效率
- 局面评估函数：不仅针对当前场上可用的点进行评估，对于蛇体运动中即将空闲出来的点也有相应的评估。可实现预判蛇尾轨迹进行追尾部操作，使本方能获得更多的可能活动空间

4.3　经典战局

此算法程序在对战普通的未剪枝博弈树或普通剪枝博弈树或普通贪心或评估算法，具有较大优势，胜率较高。但若遇到完备的 MCT 算法的程序，则胜率大大下降。

总结

这次课设让我们实践了数据结构课程的理论知识，并增强了我们对各种算法的理解与融会贯通。在网上查阅资料并相互讨论的过程中，我们接触到了新的方法如缓存增强试探法，散列表等，在这些方法的帮助下，提高效率，存储信息等问题得到了解决。但本算法仍然有改进空间，算法若采用全局的地图信息，则不必每次递归都形成一个新的地图状态，可微弱减少空间开销和时间开销。其次在团队协作编程时，务必要注意及时统一版本，避免出现重复劳动和代码版本混乱。在此次小组合作中，我们组内气氛融洽，小组成员都负责认真地完成了自己的任务，并充分与他人交流完成了代码对接，在交流沟通方面大家都获得了较大提升。

第五章　参考文献

https://www.xqbase.com/computer/search$_{a}lphabeta.htm$

http://www.soongsky.com/othello/computer/mtdf.php

http://www.soongsky.com/othello/computer/alpha$_{b}eta.php$

http://www.soongsky.com/othello/computer/hashtable.php