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

E03 Othello Game ($\alpha - \beta$ pruning)

18340013 Conghao Chen

September 14, 2020

Contents

1 Othello **2**

2 Tasks **2**

3 Codes **3**

4 Results **20**

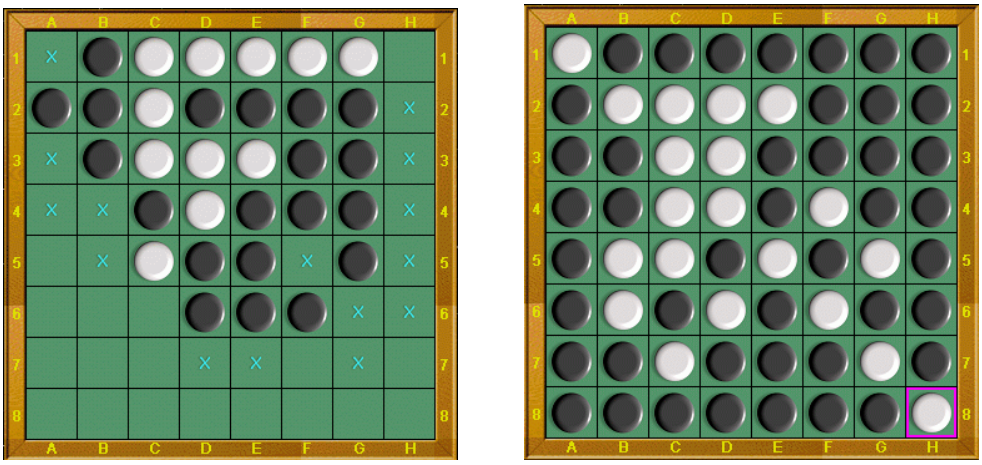

Figure 1: Othello Game

1 Othello

Othello (or Reversi) is a strategy board game for two players, played on an 8×8 uncheckered board. There are sixty-four identical game pieces called disks (often spelled "discs"), which are light on one side and dark on the other. Please see figure 1.

Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color.

The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

You can refer to `http://www.tothello.com/html/guideline_of_reversed_othello.html` for more information of guideline, meanwhile, you can download the software to have a try from `http://www.tothello.com/html/download.html`. The game installer `tothello_trial_setup.exe` can also be found in the current folder.

2 Tasks

1. In order to reduce the complexity of the game, we think the board is 6×6.

2. There are several evaluation functions that involve many aspects, you can turn to `http://www.cs.cornell.edu/~yuli/othello/othello.html` for help. In order to reduce the difficulty of the task, I have gaven you some hints of evaluation function in the file `Heuristic Function for Reversi (Othello).cpp`.

3. Please choose an appropriate evaluation function and use min-max and $\alpha - \beta$ prunning to implement the Othello game. The framework file you can refer to is `Othello.cpp`. Of course, I wish your program can beat the computer.

4. Write the related codes and take a screenshot of the running results in the file named E03_StudentNumber.pdf and send it to ai_2020@foxmail.com, the **deadline** is 2020.09.20 23:59:59.

3 Codes

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int const MAX = 65534;
int  depth = 12;          //最大搜索深度  （可调节）
//基本元素   棋子，颜色，数字变量
enum Option
{
  WHITE = -1, SPACE, BLACK  //是否能落子  //黑子
};
struct Do
{
  pair<int , int > pos;
  int score;
};
struct WinNum
{ enum Option color;
  int stable;                    // 此次落子赢棋个数
};

//主要功能    棋盘及关于棋子的所有操作，功能
struct Othello
{
  WinNum cell[6][6];                          //定义棋盘中有6*6个格子
  int whiteNum;                              //白棋数目
  int blackNum;                              //黑棋数目
  void Create(Othello *board);                        //初始化棋盘
  void Copy(Othello *boardDest, const  Othello *boardSource);     //复制棋盘
  void Show(Othello *board);                          //显示棋盘
  int Rule(Othello *board, enum Option player);            //判断落子是否符合规则
  int Action(Othello *board, Do *choice, enum Option player);     //落子,并修改棋盘
```

```
32    void Stable(Othello *board);                          //计算赢棋个数
33    int Judge(Othello *board, enum Option player);              //计算本次落子分数
34 };//主要功能
35
36
37
38
39 //我的AI用的剪枝算法
40 Do * Find1(Othello *board, enum Option player, int step, int min, int max, Do *choice)
       /*  step: 极大极小树的深度，从大往小递减  */
41 {
42    int i, j, k, num;
43    Do *allChoices;
44    choice->score = -MAX;
45    choice->pos.first = -1;
46    choice->pos.second = -1;
47
48    num = board->Rule(board, player);  /*  找出player可以落子的数量，对应于图像界面里面的
        '+' 的个数  */
49    if (!board->Rule(board, player))   /* 无处落子 */
50    {
51      if (board->Rule(board, (enum Option) - player))     /* 对方可以落子,让对方下.*/
52      {
53        Othello tempBoard;
54        Do nextChoice;
55        Do *pNextChoice = &nextChoice;
56        board->Copy(&tempBoard, board);
57        pNextChoice = Find1(&tempBoard, (enum Option) - player, step - 1, -max, -min,
        pNextChoice);
58        choice->score = -pNextChoice->score;
59        choice->pos.first = -1;
60        choice->pos.second = -1;
61        return choice;
62      }
63      else     /* 对方也无处落子,游戏结束. */
64      {
65        int value = WHITE*(board->whiteNum) + BLACK*(board->blackNum);
66        if (player*value>0)
67        {
68          choice->score = MAX - 1;
69        }
```

```
70        else if (player*value<0)
71        {
72          choice->score = -MAX + 1;
73        }
74        else
75        {
76          choice->score = 0;
77        }
78        return choice;
79      }
80    }
81    if (step <= 0)     /* 已经考虑到step步,直接返回得分 */
82    {
83      choice->score = board->Judge(board, player);
84      return choice;
85    }
86
87    /* 新建一个do*类型的数组，其中num即为玩家可落子的数量 */
88    allChoices = (Do *)malloc(sizeof(Do)*num);
89
90
91    /*
92      下面三个两重for循环其实就是分区域寻找可落子的位置，第67行代码 num = board->Rule(
        board, player)只返回了可落子的
93      数量，并没有返回可落子的位置，因此需要重新遍历整个棋盘去寻找可落子的位置。
94      下面三个for循环分别按照最外一圈、最中间的四个位置、靠里的一圈这三个顺序来寻找可落子
        的位置，如下图所示(数字
95      表示寻找的顺序)
96      1 1 1 1 1 1
97      1 3 3 3 3 1
98      1 3 2 2 3 1
99      1 3 2 2 3 1
100     1 3 3 3 3 1
101     1 1 1 1 1 1
102   */
103   k = 0;
104   for (i = 0; i<6; i++)   /* 在最外圈寻找可落子位置 */
105   {
106     for (j = 0; j<6; j++)
107     {
108       if (i == 0 || i == 5 || j == 0 || j == 5)
```

```
109         {
110             /* 可落子的位置需要满足两个条件：1、该位置上没有棋子，2、如果把棋子放在这个位置上可以吃掉对方的
111                 棋子(可以夹住对方的棋子)。stable记录的是可以吃掉对方棋子的数量，所以stable>0符合条件2
112             */
113             if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
114             {
115                 allChoices[k].score = -MAX;
116                 allChoices[k].pos.first = i;
117                 allChoices[k].pos.second = j;
118                 k++;
119             }
120         }
121     }
122 }
123
124 for (i = 0; i<6; i++)  // 分析同上
125 {
126     for (j = 0; j<6; j++)
127     {
128         if ((i == 2 || i == 3 || j == 2 || j == 3) && (i >= 2 && i <= 3 && j >= 2 && j <= 3))
129         {
130             if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
131             {
132                 allChoices[k].score = -MAX;
133                 allChoices[k].pos.first = i;
134                 allChoices[k].pos.second = j;
135                 k++;
136             }
137         }
138     }
139 }
140
141 for (i = 0; i<6; i++)  // 分析同上
142 {
143     for (j = 0; j<6; j++)
144     {
145         if ((i == 1 || i == 4 || j == 1 || j == 4) && (i >= 1 && i <= 4 && j >= 1 && j <= 4))
```

```
146         {
147             if (board->cell[i][j].color == SPACE && board->cell[i][j].stable)
148             {
149                 allChoices[k].score = -MAX;
150                 allChoices[k].pos.first = i;
151                 allChoices[k].pos.second = j;
152                 k++;
153             }
154         }
155     }
156 }
157
158 for (k = 0; k<num; k++)    /* 尝试在之前得到的num个可落子位置进行落子 */
159 {
160     Othello tempBoard;
161     Do thisChoice, nextChoice;
162     Do *pNextChoice = &nextChoice;
163     thisChoice = allChoices[k];
164     board->Copy(&tempBoard, board);  // 为了不影响当前棋盘，需要复制一份作为虚拟棋盘
165     board->Action(&tempBoard, &thisChoice, player);  // 在虚拟棋盘上落子
166     pNextChoice = Find1(&tempBoard, (enum Option) - player, step - 1, -max, -min,
    pNextChoice); // 递归调用 - 剪枝，得到对手的落子评分
167     thisChoice.score = -pNextChoice->score;
168
169     /* 使用Negamax算法代替minmax算法，实现 - 剪枝*/
170     // 其中，max 取上一层min的相反数，min取当前选择的score。
171     // 对每一层，我方行棋选择我方获益分数最大的，对手行棋选择我方获益分数最小的；
172     // 因此，实际上只需要将每一层的max min调换并取反即可；
173     // 故假设根节点为第0层，beta层的数值为负。
174     // 剪枝条件：beta <= alpha，即score >= max。
175
176     if (player == WHITE)
177     {
178         int alpha = -max, beta = -min;
179         if (thisChoice.score > -beta)
180         {
181             beta = -thisChoice.score;
182             choice->score = thisChoice.score;
183             choice->pos.first = thisChoice.pos.first;
184             choice->pos.second = thisChoice.pos.second;
185             min = -beta;
```

```
186         if (beta <= alpha) break;
187       }
188     }
189     else if(player == BLACK)
190     {
191       int alpha = min, beta = max;
192       if (thisChoice.score > alpha)
193       {
194         alpha = thisChoice.score;
195         choice->score = thisChoice.score;
196         choice->pos.first = thisChoice.pos.first;
197         choice->pos.second = thisChoice.pos.second;
198         min = alpha;
199         if (beta <= alpha) break;
200       }
201     }
202
203     // if (thisChoice.score>min && thisChoice.score<max)    /* 可以预计的更优值 */
204     // {
205     //   min = thisChoice.score;
206     //   choice->score = thisChoice.score;
207     //   choice->pos.first = thisChoice.pos.first;
208     //   choice->pos.second = thisChoice.pos.second;
209     // }
210     // else if (thisChoice.score >= max)    /* 好的超乎预计 */
211     // {
212     //   choice->score = thisChoice.score;
213     //   choice->pos.first = thisChoice.pos.first;
214     //   choice->pos.second = thisChoice.pos.second;
215     //   break;
216     // }
217     // /* 不如已知最优值 */
218   }
219   free(allChoices);
220   return choice;
221 }
222
223 int main()
224 {
225   Othello board;
226   Othello *pBoard = &board;
```

```
227   enum Option player , present ;
228   Do choice;
229   Do *pChoice = &choice;
230   int  num , result = 0;
231   char restart = ' ';
232
233 start:
234   player = SPACE;
235   present = BLACK;
236   num = 4;
237   restart = ' ';
238   cout << ">>>人机对战开始:  \n";
239
240   while (player != WHITE && player != BLACK)
241   {
242     cout << ">>>请选择执黑棋( ),或执白棋( ): 输入1为黑棋, -1为白棋" << endl;
243     scanf("%d", &player);
244     cout << ">>>黑棋行动:  \n";
245
246     if (player != WHITE && player != BLACK)
247     {
248       cout << "输入不符合规范, 请重新输入\n";
249     }
250   }
251
252   board.Create(pBoard);
253
254   while (num<36)                    // 棋盘上未下满36子
255   {
256     char *Player = "";
257     if (present == BLACK)
258     {
259       Player = "黑棋( )";
260     }
261     else if (present == WHITE)
262     {
263       Player = "白棋( )";
264     }
265
266     if (board.Rule(pBoard, present) == 0)    //未下满并且无子可下
267     {
```

```
268         if (board.Rule(pBoard, (enum Option) - present) == 0)
269         {
270           break;
271         }
272         cout << Player << "GAME OVER! \n";
273       }
274       else
275       {
276         int i, j;
277         board.Show(pBoard);
278
279         if (present == player)
280         {
281           cout << Player << ".....................";
282           pChoice = Find1(pBoard, present, depth, -MAX, MAX, pChoice);
283           i = pChoice->pos.first;
284           j = pChoice->pos.second;
285           system("cls");
286           cout << ">>>我的AI本手棋得分为      " << pChoice->score << endl;
287           board.Action(pBoard, pChoice, present);
288           num++;
289           cout << Player << ">>>我的AI于" << i + 1 << "," << j + 1<<"落子，该你了！";
290         /*  while (1)
291           {
292             cout << Player << " \n >>>请输入棋子坐标（空格相隔 如“3 5”代表第3行第5列）:\
    n";
293
294             cin >> i>> j;
295             i--;
296             j--;
297             pChoice->pos.first = i;
298             pChoice->pos.second = j;
299
300             if (i<0 || i>5 || j<0 || j>5 || pBoard->cell[i][j].color != SPACE || pBoard->
    cell[i][j].stable == 0)
301             {
302               cout <<">>>此处落子不符合规则，请重新选择   \n";
303               board.Show(pBoard);
304             }
305             else
306             {
```

```
307                 break;
308               }*
309             }*/
310             /*board.Show(pBoard);
311             system("cls");
312             cout << ">>>玩家 本手棋得分为      " << pChoice->score << endl;
313             system("pause");
314             cout << ">>>按任意键继续" << pChoice->score << endl;*/
315         }
316         else    //AI下棋
317         {
318             cout << Player << ".....................";
319             pChoice = Find1(pBoard, present, depth, -MAX, MAX, pChoice);
320             i = pChoice->pos.first;
321             j = pChoice->pos.second;
322             system("cls");
323             cout << ">>>电脑的AI本手棋得分为      " << pChoice->score << endl;
324
325             board.Action(pBoard, pChoice, present);
326             num++;
327             cout << Player << ">>>电脑的AI于" << i + 1 << "," << j + 1<<"落子，该你了！";
328         }
329     }
330
331     present = (enum Option) - present;     //交换执棋者
332 }
333
334
335 board.Show(pBoard);
336
337
338 result = pBoard->whiteNum - pBoard->blackNum;
339
340 if (result>0)
341 {
342     cout << "\n--------白棋()胜--------\n";
343 }
344 else if (result<0)
345 {
346     cout << "\n--------黑棋()胜--------\n";
347 }
```

```
348     else
349     {
350       cout << "\n----------平局----------\n";
351     }
352
353     cout << "\n ----------GAME OVER!----------\n";
354     cout << "\n";
355
356     while (restart != 'Y' && restart != 'N')
357     {
358       cout <<"|--------------------------------------|\n";
359       cout <<"|                                      | \n";
360       cout <<"|                                      |    \n";
361       cout <<"|>>>>>>>>>>>>>>>>Again?(Y,N)<<<<<<<<<<<<<<<<|\n";
362       cout <<"|                                      | \n";
363       cout <<"|                                      |   \n";
364       cout <<"|--------------------------------------|\n";
365       cout << "                                        \n";
366       cout << "                                        \n";
367       cout << "                                        \n";
368       cout << " ------                  ------        \n";
369       cout << " |  YES  |                |  NO  |      \n";
370       cout << " ------                  ------        \n";
371
372       cin >> restart;
373       if (restart == 'Y')
374       {
375         goto start;
376       }
377     }
378
379
380     return 0;
381 }
382
383
384
385
386
387
388 void Othello::Create(Othello *board)
```

```
389 {
390   int i, j;
391   board->whiteNum = 2;
392   board->blackNum = 2;
393   for (i = 0; i<6; i++)
394   {
395     for (j = 0; j<6; j++)
396     {
397       board->cell[i][j].color = SPACE;
398       board->cell[i][j].stable = 0;
399     }
400   }
401   board->cell[2][2].color = board->cell[3][3].color = WHITE;
402   board->cell[2][3].color = board->cell[3][2].color = BLACK;
403 }
404
405
406 void Othello::Copy(Othello *Fake, const  Othello *Source)
407 {
408   int i, j;
409   Fake->whiteNum = Source->whiteNum;
410   Fake->blackNum = Source->blackNum;
411   for (i = 0; i<6; i++)
412   {
413     for (j = 0; j<6; j++)
414     {
415       Fake->cell[i][j].color = Source->cell[i][j].color;
416       Fake->cell[i][j].stable = Source->cell[i][j].stable;
417     }
418   }
419 }
420
421 void Othello::Show(Othello *board)
422 {
423   int i, j;
424   cout << "\n  ";
425   for (i = 0; i<6; i++)
426   {
427     cout << "   " << i + 1;
428   }
429   cout << "\n                \n";
```

```
430   for (i = 0; i<6; i++)
431   {
432     cout << i + 1 << "-- ";
433     for (j = 0; j<6; j++)
434     {
435       switch (board->cell[i][j].color)
436       {
437         case BLACK:
438           cout << "  ";
439           break;
440         case WHITE:
441           cout << "  ";
442           break;
443         case SPACE:
444           if (board->cell[i][j].stable)
445           {
446             cout << " + ";
447           }
448           else
449           {
450             cout << "    ";
451           }
452           break;
453         default:     /* 棋子颜色错误 */
454           cout << "*  ";
455       }
456     }
457     cout << "\n                \n";
458   }
459
460   cout << ">>>白棋( )个数为:" << board->whiteNum << "          ";
461   cout << ">>>黑棋( )个数为:" << board->blackNum << endl << endl << endl;
462 }
463
464 int Othello::Rule(Othello *board, enum Option player)
465 {
466   int i, j;
467   unsigned num = 0;
468   for (i = 0; i<6; i++)
469   {
470     for (j = 0; j<6; j++)
```

```
471     {
472       if (board->cell[i][j].color == SPACE)
473       {
474         int x, y;
475         board->cell[i][j].stable = 0;
476         for (x = -1; x <= 1; x++)
477         {
478           for (y = -1; y <= 1; y++)
479           {
480             if (x || y)     /* 8个方向 */
481             {
482               int i2, j2;
483               unsigned num2 = 0;
484               for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2
    += x, j2 += y)
485               {
486                 if (board->cell[i2][j2].color == (enum Option) - player)
487                 {
488                   num2++;
489                 }
490                 else if (board->cell[i2][j2].color == player)
491                 {
492                   board->cell[i][j].stable += player*num2;
493                   break;
494                 }
495                 else if (board->cell[i2][j2].color == SPACE)
496                 {
497                   break;
498                 }
499               }
500             }
501           }
502         }
503
504         if (board->cell[i][j].stable)
505         {
506           num++;
507         }
508       }
509     }
510   }
```

```
511     return num;
512 }
513
514
515 int Othello::Action(Othello *board, Do *choice, enum Option player)
516 {
517     int i = choice->pos.first, j = choice->pos.second;
518     int x, y;
519
520     /* 要准备落子的位置上已经有棋子，或者在这个位置落子不能吃掉对方任何棋子的话，说明这个
        action不合理，直接返回 */
521     if (board->cell[i][j].color != SPACE || board->cell[i][j].stable == 0 || player ==
        SPACE)
522     {
523         return -1;
524     }
525
526
527     board->cell[i][j].color = player;
528     board->cell[i][j].stable = 0;
529
530
531     if (player == WHITE)
532     {
533         board->whiteNum++;
534     }
535     else if (player == BLACK)
536     {
537         board->blackNum++;
538     }
539
540
541
542     for (x = -1; x <= 1; x++)
543     {
544         for (y = -1; y <= 1; y++)
545         {
546
547             //需要在每个方向（8个）上检测落子是否符合规则（能否吃子）
548
549
```

```
550       if (x || y)
551       {
552         int i2, j2;
553         unsigned num = 0;
554         for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2 += x,
      j2 += y)
555         {
556           if (board->cell[i2][j2].color == (enum Option) - player)
557           {
558             num++;
559           }
560           else if (board->cell[i2][j2].color == player)
561           {
562             board->whiteNum += (player*WHITE)*num;
563             board->blackNum += (player*BLACK)*num;
564
565             for (i2 -= x, j2 -= y; num>0; num--, i2 -= x, j2 -= y)
566             {
567               board->cell[i2][j2].color = player;
568               board->cell[i2][j2].stable = 0;
569             }
570             break;
571           }
572           else if (board->cell[i2][j2].color == SPACE)
573           {
574             break;
575           }
576         }
577       }
578     }
579   }
580   return 0;
581 }
582
583
584 void Othello::Stable(Othello *board)
585 {
586   int i, j;
587   for (i = 0; i<6; i++)
588   {
589     for (j = 0; j<6; j++)
```

```
590     {
591       if (board->cell[i][j].color != SPACE)
592       {
593         int x, y;
594         board->cell[i][j].stable = 1;
595 
596         for (x = -1; x <= 1; x++)
597         {
598           for (y = -1; y <= 1; y++)
599           {
600             /* 4个方向 */
601             if (x == 0 && y == 0)
602             {
603               x = 2;
604               y = 2;
605             }
606             else
607             {
608               int i2, j2, flag = 2;
609               for (i2 = i + x, j2 = j + y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2
    += x, j2 += y)
610               {
611                 if (board->cell[i2][j2].color != board->cell[i][j].color)
612                 {
613                   flag--;
614                   break;
615                 }
616               }
617 
618               for (i2 = i - x, j2 = j - y; i2 >= 0 && i2 <= 5 && j2 >= 0 && j2 <= 5; i2
    -= x, j2 -= y)
619               {
620                 if (board->cell[i2][j2].color != board->cell[i][j].color)
621                 {
622                   flag--;
623                   break;
624                 }
625               }
626 
627               if (flag)     /* 在某一条线上稳定 */
628               {
```

```
629                 board->cell[i][j].stable++;
630               }
631             }
632           }
633         }
634       }
635     }
636   }
637 }
638
639 int Othello::Judge(Othello *board, enum Option player)
640 {
641   int value = 0;
642   int i, j;
643   Stable(board);
644
645   // 对稳定子给予奖励
646   for (i = 0; i<6; i++)
647   {
648     for (j = 0; j<6; j++)
649     {
650       value += (board->cell[i][j].color)*(board->cell[i][j].stable);
651     }
652   }
653
654    int V[6][6] = {{ 20,  -8,  11,  11,  -8,  20},
655           { -8, -15,  -4,  -4, -15,  -8},
656           { 11,  -4,   2,   2,  -4,  11},
657           { 11,  -4,   2,   2,  -4,  11},
658           { -8, -15,  -4,  -4, -15,  -8},
659           { 20,  -8,  11,  11,  -8,  20}};
660
661    for (int i = 0; i < 6; ++i)
662    {
663     for (int j = 0; j < 6; ++j)
664     {
665       value += V[i][j] * board->cell[i][j].color;
666     }
667    }
668
669   // 行动力计算
```

```
670     int my_mov, opp_mov, mov = 0;
671     my_mov = Rule(board, player);
672     opp_mov = Rule(board, (enum Option) - player);
673     if(my_mov > opp_mov)
674         value += 78.922 * (100.0 * my_mov)/(my_mov + opp_mov);
675     else if(my_mov < opp_mov)
676         value += 78.922 * -(100.0 * opp_mov)/(my_mov + opp_mov);
677 
678     return value*player;
679 }
```

4 Results

从图可以看到，最终是执黑棋的我的 AI 击败了执白棋的电脑的 AI。代码主要修改了两部分，一部分是 Judge 函数里的 Evaluation function，还有就是剪枝算法做了修改：使用自己写的 Negamax 算法，而不是原文件内的算法。不过这个程序要执行好久，大约 7-10min 左右才能看到结果。运行后不用进行任何操作，自己会交替显示双方的情况。

Figure 2: Result