P02 CSP and KRR

学 号	姓 名	专业 (方向)
18340052	何 泽	计算机科学与技术（超级计算方向）
18340032	邓俊锋	计算机科学与技术（大数据与人工智能方向）

I Futoshiki (GAC, C++/Python)

1. Description

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), please see Figure 1.

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/.

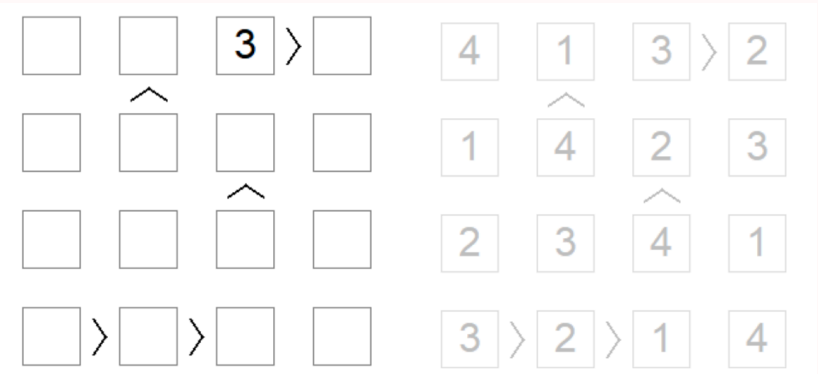


Figure 1: Futoshiki Puzzles

2. Tasks

(1)

Describe with sentences the main ideas of the GAC algorithm and the main differences between the GAC and the forward checking (FC) algorithm.

- GAC：限制C(V1,V2,...,Vn)是关于Vi边一致的， 当且仅当∀Vi, ∃V1, . . . , Vi−1, Vi+1, . . . , Vn满足C。限制C是GAC的当且仅当对于每一变量都是GAC的，一 个CSP是GAC的当且仅当它的限制都是GAC的，在Vi = d下，没有其他变量赋值能够满足该限制，则d是边不一致的，进而Vi = d可以被剪枝，并采用队列的方式，不断将需要检测边一致性的限制添加直到队列为空。
- GAC和FC的区别：FC只考虑当前一个限制下某一变量赋值带来的后果，而GAC则考虑当前变量赋值在所有限制下所带来的后果。

(2)

The GAC Enforce procedure from class acts as follows: when removing d from CurDom[V], push all constraints C' such that V ∈ scope(C') and C' ∉ GACQueue onto GACQueue. What's the reason behind this operation? Can it be improved and how?

- 因为当从论域中移除一个值时可能导致其他论域的新的不一致；即后续的剪枝可能会影响已进行的约束剪枝，所以需要采用队列的方式，不断将需要检测的限制添加进去，直

到所有约束都满足GAC。

- 针对课件中给出的GAC算法，当前约束可以不添加至队列中，以此改善性能

(3)

> Use the GAC algorithm to implement a Futoshiki solver by C++ or Python.

- 使用的全局变量与结构体定义：

```
const int max_size = 9;
int size;
static int nodes = 0;
clock_t start_time,end_time;
double node_time;

struct Position{
    int row, col;
    Position() {}
    Position(int a, int b): row(a), col(b) {}
};

struct Relation {
    Position p;
    int r;
    Relation() {}
    Relation(Position a, int re): p(a), r(re) {}
};

struct MultiRelation {
    Position x, y;
    int r;
    MultiRelation() {}
    MultiRelation(Position a, Position b, int re): x(a),
y(b), r(re) {}
};

struct item {
```

```
    int val;
    Position pos;
    int curdom[max_size+1];
    bool assigned;
    vector<Relation> relation;
    item() {}
};

struct futoshiki {
    item board[max_size+1][max_size+1];
    vector<MultiRelation> multirelation;
    futoshiki() {}
};
```

- 检查某一行、某一列是否有重复数字以及与相邻元素的大小关系是否符合规则：

```
bool RowCheck(futoshiki* f, item* m) {
    int row = m->pos.row;
    int val = m->val;
    for(int i = 1;i≤size;i++) {
        item* current_item = &f->board[row][i];
        if (current_item->assigned)
            continue;
        if (!current_item->curdom[val]) {
            current_item->curdom[val] = 1;
            current_item->curdom[0] = CDcount(&f->board[row][i]);
        }
        if(current_item->curdom[0] == size)
            return false;
    }
    return true;
}

bool ColCheck(futoshiki* f, item* m) {
    int col = m->pos.col;
    int val = m->val;
    for(int i = 1;i≤size;i++) {
        item* current_item = &f->board[i][col];
```

```
        if (current_item->assigned)
            continue;
        if (!current_item->curdom[val]) {
            current_item->curdom[val] = 1;
            current_item->curdom[0] = CDcount(&f->board[i]
[col]);
        }
        if(current_item->curdom[0] == size)
            return false;
    }
    return true;
}

bool neighbour_check(futoshiki* f) {
    bool flag = true;
    for (size_t m = 0;m<f->multirelation.size();m++) {
        int x_row = f->multirelation[m].x.row;
        int x_col = f->multirelation[m].x.col;
        int y_row = f->multirelation[m].y.row;
        int y_col = f->multirelation[m].y.col;
        item* X = &f->board[x_row][x_col];
        item* Y = &f->board[y_row][y_col];
        int re = f->multirelation[m].r;
        if (X->assigned && Y->assigned)
            continue;
        else if (X->assigned) {
            int v = X->val;
            if (re == 1) {
                for (int j = v;j≤size;j++) {
                    if (!Y->curdom[j]) {
                        Y->curdom[j] = 1;
                        Y->curdom[0] = CDcount(Y);
                    }
                }
                if(Y->curdom[0] == size)
                    flag = false;
            }
            else if (re == -1)  {
                for (int j = v;j>0;j--){
```

```
                    if (!Y->curdom[j]) {
                        Y->curdom[j] = 1;
                        Y->curdom[0] = CDcount(Y);
                    }
                }
                if(Y->curdom[0] == size)
                    flag = false;
            }
        }
        else if (Y->assigned) {
            int v = Y->val;
            if (re == 1) {
                for (int j = v;j>0;j--) {
                    if (!X->curdom[j]) {
                        X->curdom[j] = 1;
                        X->curdom[0] = CDcount(X);
                    }
                }
                if(X->curdom[0] == size)
                    flag = false;
            }
            else if (re == -1)  {
                for (int j = v;j≤size;j++){
                    if (!X->curdom[j]) {
                        X->curdom[j] = 1;
                        X->curdom[0] = CDcount(X);
                    }
                }
                if(X->curdom[0] == size)
                    flag = false;
            }
        }
        else {
            if (re == 1) {
                for (int j = 1;j≤size;j++){
                    if (!X->curdom[j]) {
                        int t = 1;
                        while (Y->curdom[t++]) {
                            if(t == j) {
```

```
                                X->curdom[j] = 1;
                                X->curdom[0] = CDcount(X);
                            }
                        }
                    }
                    if(X->curdom[0] == size)
                        flag = false;
                }
                for (int j = 1;j<=size;j++){
                    if (!Y->curdom[j]) {
                        int t = j + 1;
                        while (X->curdom[t++]) {
                            if(t == size + 1) {
                                Y->curdom[j] = 1;
                                Y->curdom[0] = CDcount(Y);
                            }
                        }
                        if(Y->curdom[0] == size)
                            flag = false;
                    }
                }
            }
            else if (re == -1)  {
                for (int j = 1;j<=size;j++){
                    if (!X->curdom[j]) {
                        int t = j + 1;
                        while (Y->curdom[t++]) {
                            if(t == size + 1) {
                                X->curdom[j] = 1;
                                X->curdom[0] = CDcount(X);
                            }
                        }
                        if(X->curdom[0] == size)
                            flag = false;
                    }
                }
                for (int j = 1;j<=size;j++) {
                    if (!Y->curdom[j]) {
                        int t = 1;
```

```
                        while (X->curdom[t++]) {
                            if(t == j) {
                                Y->curdom[j] = 1;
                                Y->curdom[0] = CDcount(Y);
                            }
                        }
                        if(Y->curdom[0] == size)
                            flag = false;
                    }
                }
            }
        }
    }
    return flag;
}
```

- main函数：

  分别输入维数、大小关系和初始值，后两个都以全0结束，例如测例1的输入如下：

```
5

1 1 > 1 2
2 1 > 1 1
2 2 < 2 3
2 3 < 2 4
2 4 < 2 5
3 2 < 3 3
5 1 > 5 2
0 0 0 0 0

5 5 4
0 0 0
```

  代码如下：

```
int main() {
    cout << "请输入维数: ";
    cin >> size;
```

```
    futoshiki f;
    futoshiki* ptr = &f;
    for (int i = 1;i≤size;i++) {
        for (int j = 1;j≤size;j++) {
            ptr->board[i][j].val = 0;
            ptr->board[i][j].pos.row = i;
            ptr->board[i][j].pos.col = j;
            ptr->board[i][j].assigned = 0;
            memset(ptr->board[i][j].curdom, 0, sizeof(ptr-
>board[i][j].curdom));
        }
    }
    cout << "请输入大小关系（以0 0 0 0 0结束）：" << endl;
    while(1) {
        Position x, y;
        char c;
        cin >> x.row >> x.col >> c >> y.row >> y.col;
        if (c≠'<' && c≠'>')
            break;
        Relation tmp1(y, ((c == '>') ? 1 : -1));
        Relation tmp2(x, ((c == '>') ? -1 : 1));
        MultiRelation tmp(x, y, ((c == '>') ? 1 : -1));
        ptr->multirelation.push_back(tmp);
        ptr->board[x.row][x.col].relation.push_back(tmp1);
        ptr->board[y.row][y.col].relation.push_back(tmp2);
    }
    cout << "请输入某些元素初始值（以0 0 0结束）：" << endl;
    while(1) {
        int a, b, v;
        cin >> a >> b >> v;
        if (a+b+v == 0)
            break;
        ptr->board[a][b].val = v;
        ptr->board[a][b].assigned = 1;
        GAC_Enforce(ptr, &ptr->board[a][b]);
    }
    cout << "初始Futoshiki：" << endl;
    display(ptr);
    start_time=clock();
```

```
    GAC(ptr);
    return 0;
}
```

- GAC检测：

```
bool GAC_Enforce(futoshiki* f, item* v) {
    bool flag_row = 0, flag_col = 0, flag_compare = 0;
    flag_row = RowCheck(f, v);
    flag_col = ColCheck(f, v);
    flag_compare = neighbour_check(f);
    return (flag_row && flag_col && flag_compare);
}

void GAC(futoshiki* f) {
    clock_t clock1=clock();
    nodes++;
    if (finish(f)) {
        end_time=clock();
        double time=(double)(end_time-start_time)/CLOCKS_PER_SEC;
        cout << "结果Futoshiki: " << endl;
        display(f);
        cout << "运行总时间: " << time*1000 << " ms" << endl;
        cout << "搜索的节点数: " << nodes << "次" << endl;
        cout << "每个节点平均GAC时间: " << (double)(node_time*1000.00/(nodes-1)) << "ms" << endl;
        exit(0);
    }
    item* v = heuristicpick(f);
    v->assigned = true;
    for (int i = 1;i≤size;i++){
        if (!v->curdom[i]) {
            v->val = i;
            futoshiki boardcopy;
            Copyboard(&boardcopy, f);
            for (int j = 1;j≤size;j++){
                if (j ≠ i && !v->curdom[j]) {
                    v->curdom[j] = 1;
```

```
                    v->curdom[0] = CDcount(v);
                }
            }
            if (GAC_Enforce(f, v) != false) {
                GAC(f);
            }
            Copyboard(f, &boardcopy);
        }
    }
    v->assigned = false;
    clock_t clock2=clock();
    double time_per_node=(double)(clock2-clock1)/CLOCKS_PER_SEC;
    node_time+=time_per_node;
}
```

(4)

> Explain any ideas you use to speed up the implementation.

每次选点的时候使用MRV启发式函数：

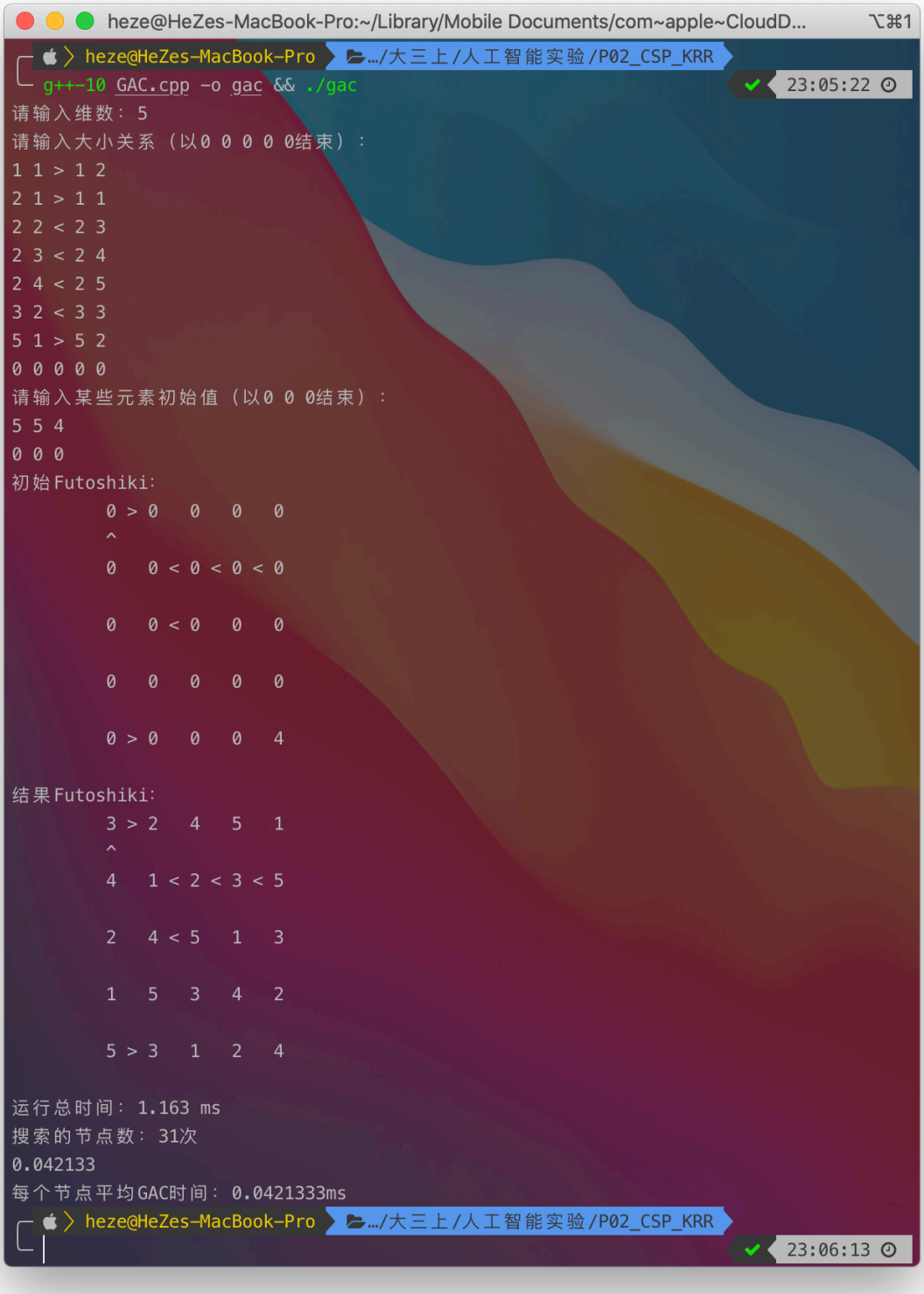
```
item* heuristicpick(futoshiki* f) {
    // MRV
    item* maxi = &f->board[1][1];
    for (int i = 1;i≤size;i++) {
        for (int j = 1;j≤size;j++) {
            if(f->board[i][j].assigned)
                continue;
            if(maxi->curdom[0] < f->board[i][j].curdom[0] || maxi->assigned) {
                maxi = &f->board[i][j];
                if (maxi->curdom[0] == size-1)
                    return maxi;
            }
        }
    }
```

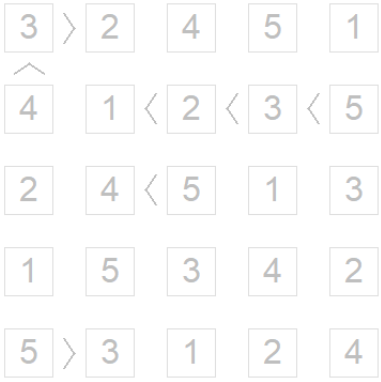
```
15      return maxi;
16  }
```

(5)

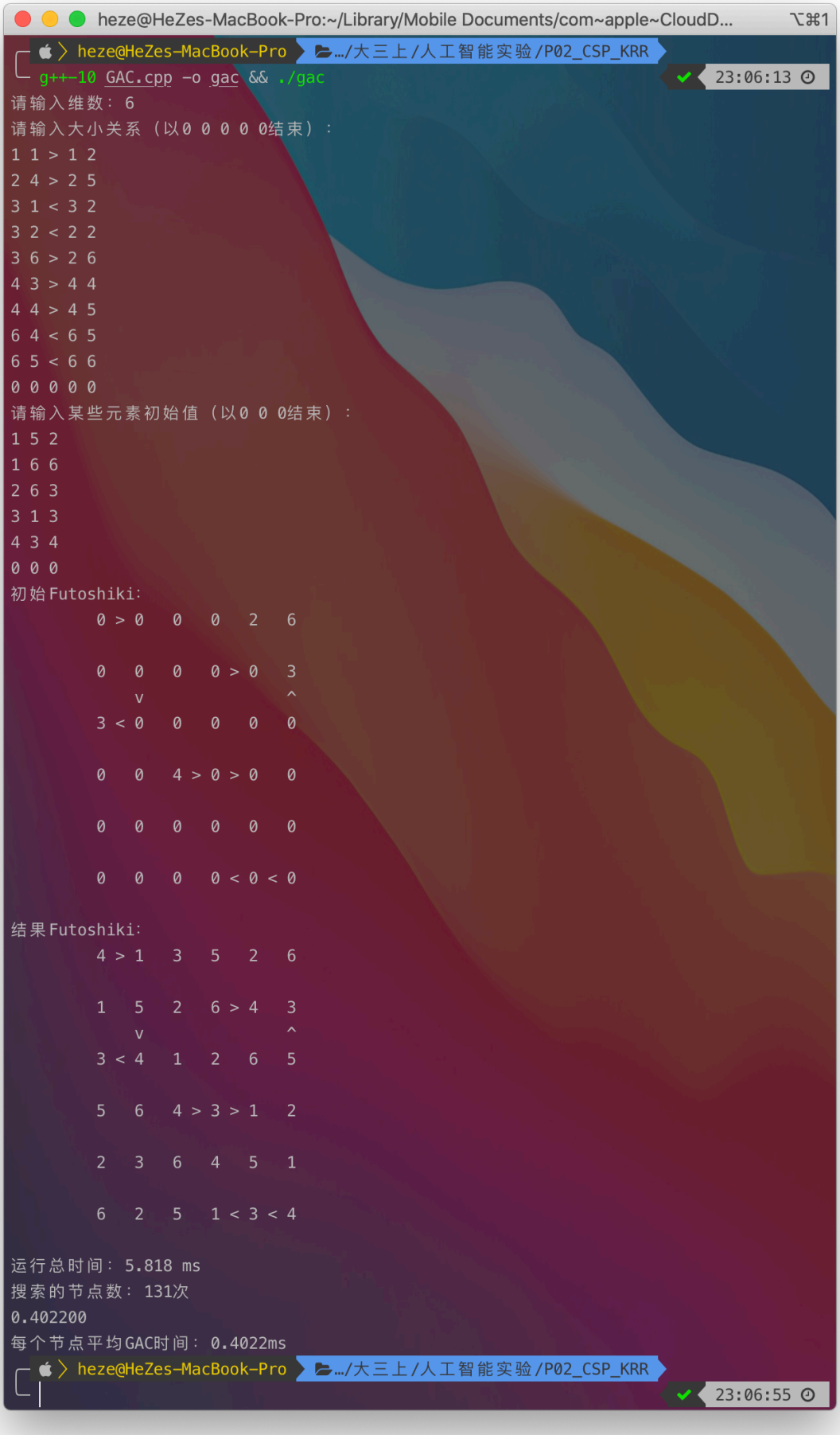
> Run the following 5 test cases to verify your solver's correctness. We also provide test file "datai.txt" for every test case i. Refer to the "readme.txt" for more details.

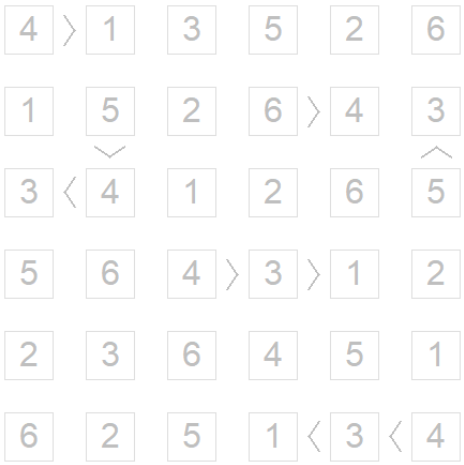
- 1 :


heze@HeZes-MacBook-Pro:~/Library/Mobile Documents/com~apple~CloudD...
heze@HeZes-MacBook-Pro
…/大三上/人工智能实验/P02_CSP_KRR
g++-10 GAC.cpp -o gac && ./gac
23:05:22
请输入维数： 5
请输入大小关系（以0 0 0 0 0结束）：
1 1 > 1 2
2 1 > 1 1
2 2 < 2 3
2 3 < 2 4
2 4 < 2 5
3 2 < 3 3
5 1 > 5 2
0 0 0 0 0
请输入某些元素初始值（以0 0 0结束）：
5 5 4
0 0 0
初始Futoshiki:
0 > 0 0 0 0
^
0 0 < 0 < 0 < 0
0 0 < 0 0 0
0 0 0 0 0
0 > 0 0 0 4
结果Futoshiki:
3 > 2 4 5 1
^
4 1 < 2 < 3 < 5
2 4 < 5 1 3
1 5 3 4 2
5 > 3 1 2 4
运行总时间： 1.163 ms
搜索的节点数： 31次
0.042133
每个节点平均GAC时间： 0.0421333ms
heze@HeZes-MacBook-Pro
…/大三上/人工智能实验/P02_CSP_KRR
23:06:13

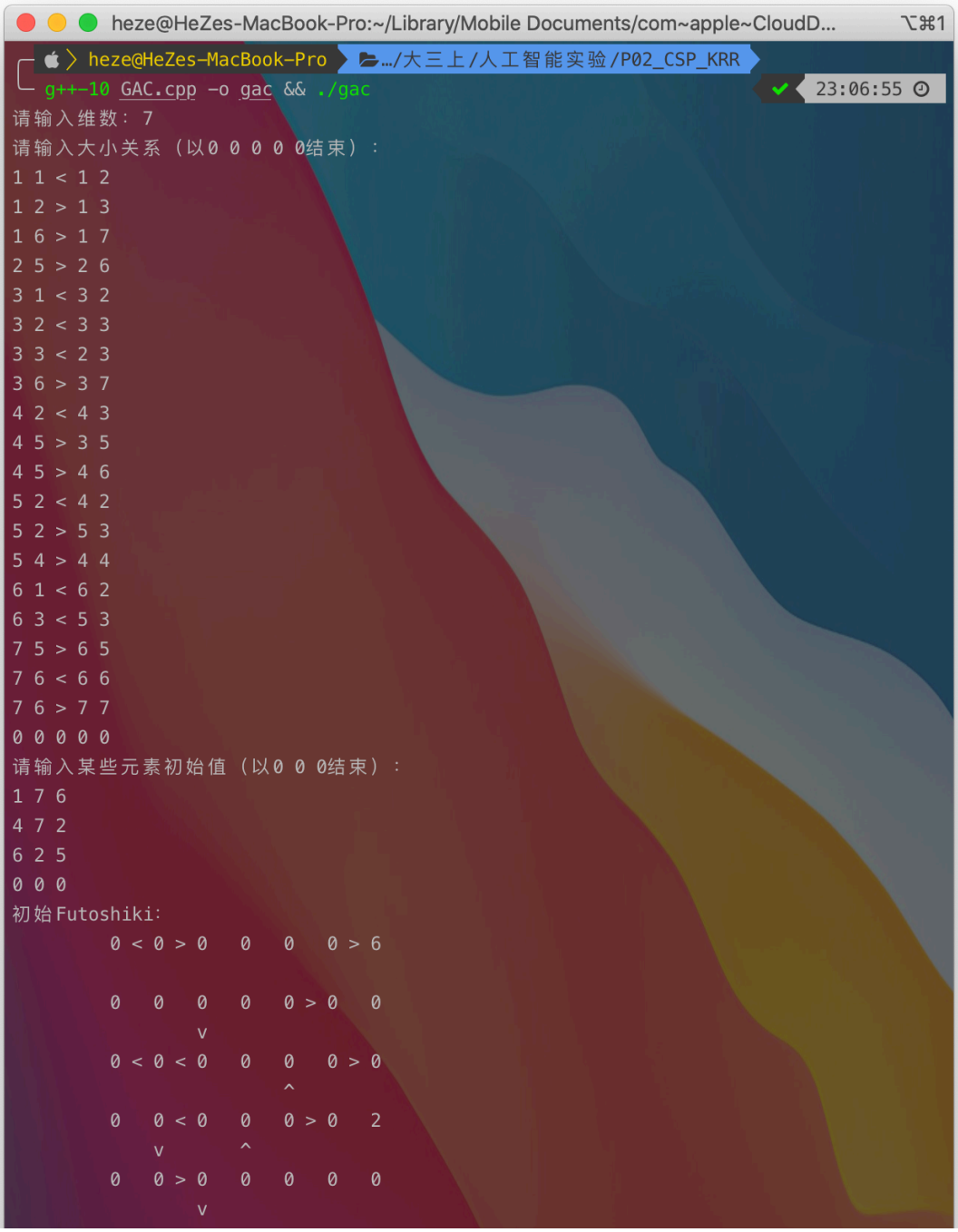


- 2:


heze@HeZes-MacBook-Pro:~/Library/Mobile Documents/com~apple~CloudD...
heze@HeZes-MacBook-Pro …/大三上/人工智能实验/P02_CSP_KRR
g++-10 GAC.cpp -o gac && ./gac
23:06:13
请输入维数： 6
请输入大小关系（以0 0 0 0 0结束）：
1 1 > 1 2
2 4 > 2 5
3 1 < 3 2
3 2 < 2 2
3 6 > 2 6
4 3 > 4 4
4 4 > 4 5
6 4 < 6 5
6 5 < 6 6
0 0 0 0 0
请输入某些元素初始值（以0 0 0结束）：
1 5 2
1 6 6
2 6 3
3 1 3
4 3 4
0 0 0
初始Futoshiki:
0 > 0 0 0 2 6
0 0 0 0 > 0 3
v ^
3 < 0 0 0 0 0
0 0 4 > 0 > 0 0
0 0 0 0 0 0
0 0 0 0 < 0 < 0
结果Futoshiki:
4 > 1 3 5 2 6
1 5 2 6 > 4 3
v ^
3 < 4 1 2 6 5
5 6 4 > 3 > 1 2
2 3 6 4 5 1
6 2 5 1 < 3 < 4
运行总时间： 5.818 ms
搜索的节点数： 131次
0.402200
每个节点平均GAC时间： 0.4022ms
heze@HeZes-MacBook-Pro …/大三上/人工智能实验/P02_CSP_KRR
23:06:55


4 > 1 3 5 2 6
1 5 2 6 > 4 3
3 < 4 1 2 6 5
5 6 4 > 3 > 1 2
2 3 6 4 5 1
6 2 5 1 < 3 < 4

- 3:


heze@HeZes-MacBook-Pro:~/Library/Mobile Documents/com~apple~CloudD...
heze@HeZes-MacBook-Pro …/大三上/人工智能实验/P02_CSP_KRR
g++-10 GAC.cpp -o gac && ./gac
23:06:55
请输入维数：7
请输入大小关系（以0 0 0 0 0结束）：
1 1 < 1 2
1 2 > 1 3
1 6 > 1 7
2 5 > 2 6
3 1 < 3 2
3 2 < 3 3
3 3 < 2 3
3 6 > 3 7
4 2 < 4 3
4 5 > 3 5
4 5 > 4 6
5 2 < 4 2
5 2 > 5 3
5 4 > 4 4
6 1 < 6 2
6 3 < 5 3
7 5 > 6 5
7 6 < 6 6
7 6 > 7 7
0 0 0 0 0
请输入某些元素初始值（以0 0 0结束）：
1 7 6
4 7 2
6 2 5
0 0 0
初始Futoshiki:
0 < 0 > 0 0 0 0 > 6
0 0 0 0 0 > 0 0
v
0 < 0 < 0 0 0 0 > 0
^
0 0 < 0 0 0 > 0 2
v ^
0 0 > 0 0 0 0 0
v

```
        0 < 5   0   0   0   0   0
                        ^   v
        0   0   0   0   0   0 > 0

结果Futoshiki:
        2 < 4 > 3   5   1   7 > 6

        4   1   6   2   7 > 5   3
                v
        1 < 2 < 5   7   3   6 > 4
                        ^
        5   6 < 7   1   4 > 3   2
            v       ^
        7   3 > 2   4   6   1   5
                v
        3 < 5   1   6   2   4   7
                        ^   v
        6   7   4   3   5   2 > 1

运行总时间： 2542.65 ms
搜索的节点数： 81556次
0.670962
每个节点平均GAC时间： 0.670962ms
 heze@HeZes-MacBook-Pro …/大三上/人工智能实验/P02_CSP_KRR
                                                        23:07:39
```

- 4:


heze@HeZes-MacBook-Pro:~/Library/Mobile Documents/com~apple~CloudD... ⌥⌘1
初始Futoshiki:
0 0 > 0 > 0 0 < 0 < 0 < 0
0 < 0 0 0 6 0 < 7 0
0 0 0 < 4 0 0 0 0
0 > 0 > 0 0 0 0 0 0
0 > 0 0 0 0 0 0 < 6
0 0 0 0 0 < 4 > 0 0
0 0 0 > 0 0 0 0 3
0 0 0 0 0 0 0 0
结果Futoshiki:
8 7 > 6 > 5 1 < 2 < 3 < 4
4 < 8 2 3 6 5 < 7 1
7 2 3 < 4 8 1 6 5
6 > 3 > 1 2 4 7 5 8
5 > 4 8 1 7 3 2 < 6
2 6 5 8 3 < 4 > 1 7
1 5 7 > 6 2 8 4 3
3 1 4 7 5 6 8 2
运行总时间: 86.18 ms
搜索的节点数: 1922次
0.744170
每个节点平均GAC时间: 0.74417ms
heze@HeZes-MacBook-Pro …/大三上/人工智能实验/P02_CSP_KRR
23:09:58

- 5（我使用的是txt中的测例而不是PDF中的）：


heze@HeZes-MacBook-Pro:~/Library/Mobile Documents/com~apple~CloudD...
⌥⌘1
初始Futoshiki:
6 0 0 0 0 0 > 0 0 < 0
0 0 0 > 0 0 < 0 0 0 0
0 0 0 < 0 0 0 > 0 0 < 0
0 0 0 0 0 0 < 0 7 0
0 0 > 0 0 0 < 0 < 0 0 > 0
0 < 0 0 0 0 < 0 > 0 > 0 0
0 0 0 0 0 0 > 0 5 8
0 0 0 0 0 0 < 0 < 8 0
0 > 0 > 0 0 > 0 0 0 2 4
结果Futoshiki:
6 5 8 4 2 7 > 1 3 < 9
7 2 6 > 3 4 < 5 8 9 1
3 9 1 < 2 5 8 > 6 4 < 7
1 8 2 6 9 4 < 5 7 3
9 7 > 4 8 1 < 2 < 3 6 > 5
2 < 3 7 5 8 < 9 > 4 > 1 6
4 1 3 9 7 6 > 2 5 8
5 4 9 1 6 3 < 7 < 8 2
8 > 6 > 5 7 > 3 1 9 2 4
运行总时间： 117964 ms
搜索的节点数： 2250002次
1.442479
每个节点平均GAC时间： 1.44248ms
heze@HeZes-MacBook-Pro
…/大三上/人工智能实验/P02_CSP_KRR
23:12:45

(6)

> Run the FC algorithm you implemented in E04 and the GAC algorithm you implemented in Task 3 on the 5 test cases, and fill in the following table. In the table, "Total Time" means the total time the algorithm uses to solve the test case, "Number of Nodes Searched" means the total number of nodes traversed by the algorithm, and "Average Inference Time Per Node" means the average time for constraint propagation (inference) used in each node (note that this time is not equal to the total time divided by the number of nodes searched). Analyse the reasons behind the experimental results, and write them in your report.

Test Case	Algorithm	Total Time	Nodes Searched	Time Per Node
1	FC GAC	1.861ms 1.163ms	46 31	0.0658ms 0.04213ms
2	FC GAC	8.572ms 5.818ms	173 131	0.722ms 0.4022ms
3	FC GAC	3685.4ms 2542.65ms	280174 81556	0.9254ms 0.670962ms
4	FC GAC	121.025ms 86.18ms	3914 1922	1.2141ms 0.74417ms
5	FC GAC	2036584ms 117964ms	51938902 2250002	2.0563ms 1.44248ms

可以看到，整体而言GAC运行时间、单个节点时间都要比FC要快，搜索的节点个数也更少，而且随着问题复杂度的提升，GAC的优势也更加明显，这与预期的结果是相符的。

II Resolution

1

> Implement the MGU algorithm.

The code is shown below:

```
def fingmgu(cl1, cl2):
    for i in cl1:
        for j in cl2:
            if i[0] == inver(j[0]):
                flag = 0
                for num in range(1, len(i)):
                    l1 = len(i[num])
                    l2 = len(j[num])
                    if l1 > 1 and l2 > 1 and i[num] != j[num]:
                        flag = 1
                if flag == 1:
                    break
                for num in range(1, len(i)):
                    l1 = len(i[num])
                    l2 = len(j[num])
                    if l1 == 1 and l2 > 1:
                        for ii in cl1:
                            if ii == i:
                                continue
                            for jj in ii:
                                if jj == i[num]:
                                    cl1[cl1.index(ii)]
[ii.index(jj)] = j[num]
                for num in range(1, len(i)):
                    l1 = len(i[num])
                    l2 = len(j[num])
                    if l1 > 1 and l2 == 1:
                        for ii in cl2:
                            if ii == j:
                                continue
                            for jj in ii:
                                if jj == j[num]:
                                    cl2[cl2.index(ii)]
[ii.index(jj)] = i[num]
                n1, n2 = cl1.index(i), cl2.index(j)
```

```
                cl1.remove(i)
                cl2.remove(j)
                if cl2 != []:
                    cl1.extend(cl2)
                return n1, n2
    return None
```

2

> Using the MGU algorithm, implement a system to decide via resolution if a set of first-order clauses is satisfiable. The input of your system is a file containing a set of first-order clauses. In case of unsatisfiability, the output of your system is a derivation of the empty clause where each line is in the form of "R[8a,12c]clause". Only include those clauses that are useful in the derivation.

Input clauses (provided by TA):

```
clauses = []
num = int(input())
for i in range(0, num):
    clause = []
    for item in re.findall(r'¬*[a-zA-Z]+\([a-zA-Z,\s]*\)',
input()):
        items = re.findall(r'[¬a-zA-Z]+', item)
        clause.append(items)
    clauses.append(clause)
for ii in clauses:
    print(ii)
```

The `dfs` search with a limitation of search depth:

```
def dfs(dep, thcl, deptop):
    global flag
    global ans
    global cls
    if (dep > deptop):
```

```
        return
    if (thcl == []):
        flag = 1
    if (flag == 1):
        return
    for i in cls:
        cl1 = copy.deepcopy(thcl)
        cl2 = copy.deepcopy(i)
        n1, n2 = fingmgu(cl1, cl2)
        if n1 == -1:
            continue
        ans.append(i)
        cls.append(cl1)
        dfs(dep + 1, cl1, deptop)
        cls.remove(cl1)
        if flag:
            return
        ans.pop()
```

Output the form of "R[8a,12c]clause":

```
for i in ans:
    n1, n2 = fingmgu(dst, copy.deepcopy(i))
    n1 = n1+97
    n2 = n2+97
    output = "R[" + str(lstid) + chr(n1) + "," +
str(newid[nowset.index(i)]) + chr(n2) + "] : "
    idx = idx + 1
    lstid = idx
    for j in dst:
        for k in range(len(j)):
            if k == 0:
                output += j[k]
                output += '('
            elif k == len(j) - 1:
                output += j[k]
                output += ')'
            else:
                output += j[k]
```

```
                output += ','
        if j != dst[-1]:
            output += ' , '
    print(idx, output)
```

3

> Explain any ideas you use to improve the search efficiency.

I set a limitation of search depth. Because DFS will continue to search until it is very deep, so I manually limit the depth to reduce the search.

4

> Run your system on the examples of hardworker(sue), 3-blocks, Alpine Club. Include your input and output files in your report.

I input the test case by manual input, and the results are as follows:

- AIpine Club

```
[['A', 'tony']]
[['A', 'mike']]
[['A', 'john']]
[['L', 'tony', 'rain']]
[['L', 'tony', 'snow']]
[['¬A', 'x'], ['S', 'x'], ['C', 'x']]
[['¬C', 'y'], ['¬L', 'y', 'rain']]
[['L', 'z', 'snow'], ['¬S', 'z']]
[['¬L', 'tony', 'u'], ['¬L', 'mike', 'u']]
[['L', 'tony', 'v'], ['L', 'mike', 'v']]
[['¬A', 'w'], ['¬C', 'w'], ['S', 'w']]
12 R[6a,2a] : S(mike) , C(mike)
```

```
13 R[9a,5a] : ¬L(mike,snow)
14 R[12a,8b] : C(mike) , L(mike,snow)
15 R[13a,8a] : ¬S(mike)
16 R[11a,2a] : ¬C(mike) , S(mike)
17 R[16a,14a] : S(mike) , L(mike,snow)
18 R[17a,15a] : L(mike,snow)
19 R[18a,13a] :
```

- hardworker(sue)

```
[['GradStudent', 'sue']]
[['¬GradStudent', 'x'], ['Student', 'x']]
[['¬Student', 'x'], ['HardWorker', 'x']]
[['¬HardWorker', 'sue']]
5 R[4a,3b] : ¬Student(sue)
6 R[5a,2b] : ¬GradStudent(sue)
7 R[6a,1a] :
```

- 3-blocks

```
[['On', 'aa', 'bb']]
[['On', 'bb', 'cc']]
[['Green', 'aa']]
[['¬Green', 'cc']]
[['¬On', 'x', 'y'], ['¬Green', 'x'], ['Green', 'y']]
6 R[5a,2a] : ¬Green(bb) , Green(cc)
7 R[5a,1a] : ¬Green(aa) , Green(bb)
8 R[7a,3a] : Green(bb)
9 R[8a,6a] : Green(cc)
10 R[9a,4a] :
```

5

What do you think are the main problems for using resolution to check for satisfiability for a set of first-order clauses? Explain.

First, check for satisfiability is a NP complete problem. This means that satisfiability is believed by most people to be unsolvable in polynomial time. The more data you have, the slower it is. Using SAT to determine satisfiability appear to work much better in practice than resolution.

III 实验总结

通过完成这次的项目，首先更加深入地了解了GAC检测以及和FC检测的区别与差距，其次更熟悉了基于归结的推理的过程。