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

# P02 CSP and KRR

18364066 Yanzuo Lu

October 28, 2020

## Contents

# 1 Futoshiki (GAC, C++/Python)

## 1.1 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 ($4 \times 4$ 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/`.

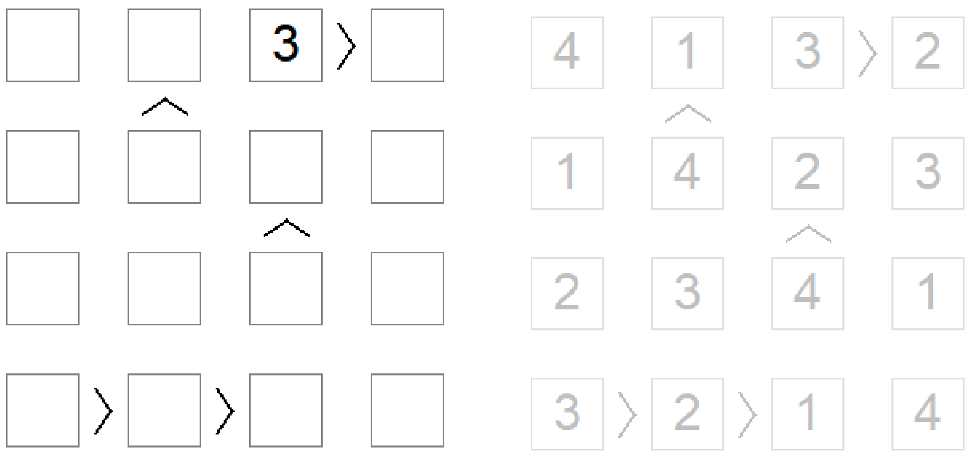



Figure 1: Futoshiki Puzzles

## 1.2 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. (10 points)

2. The GAC_Enforce procedure from class acts as follows: when removing d from CurDom[V], push all constraints $C'$ such that $V \in \mathit{scope}(C')$ and $C' \notin$ GACQueue onto GACQueue. What's the reason behind this operation? Can it be improved and how? (20 points)

3. Use the GAC algorithm to implement a Futoshiki solver by **C++** or **Python**. (20 points)

4. Explain any ideas you use to speed up the implementation. (10 points)

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. (20 points)

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. (20 points)

| Test Case | Algorithm | Total Time | Number of Nodes Searched | Average Inference Time Per Node |
|---|---|---|---|---|
| 1 | FC | | | |
| | GAC | | | |
| 2 | FC | | | |
| | GAC | | | |
| 3 | FC | | | |
| | GAC | | | |
| 4 | FC | | | |
| | GAC | | | |
| 5 | FC | | | |
| | GAC | | | |


Figure 2: Futoshiki Test Case 1

Figure 3: Futoshiki Test Case 2

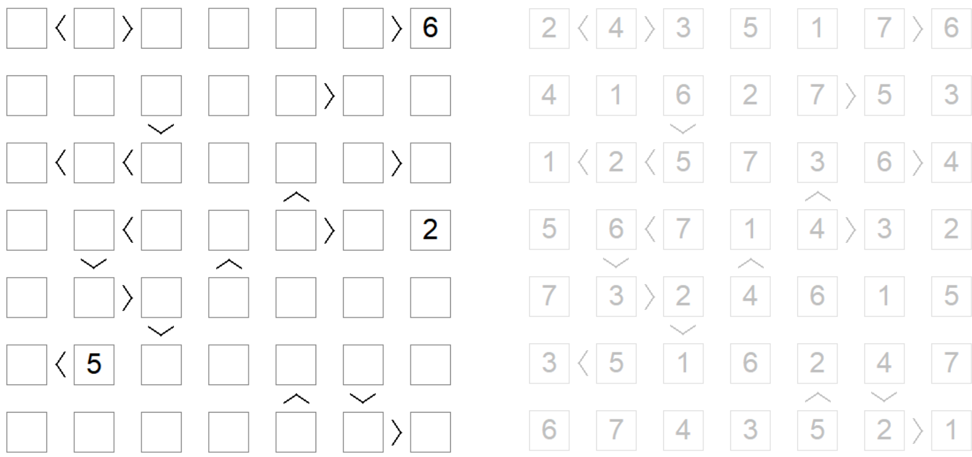


Figure 4: Futoshiki Test Case 3

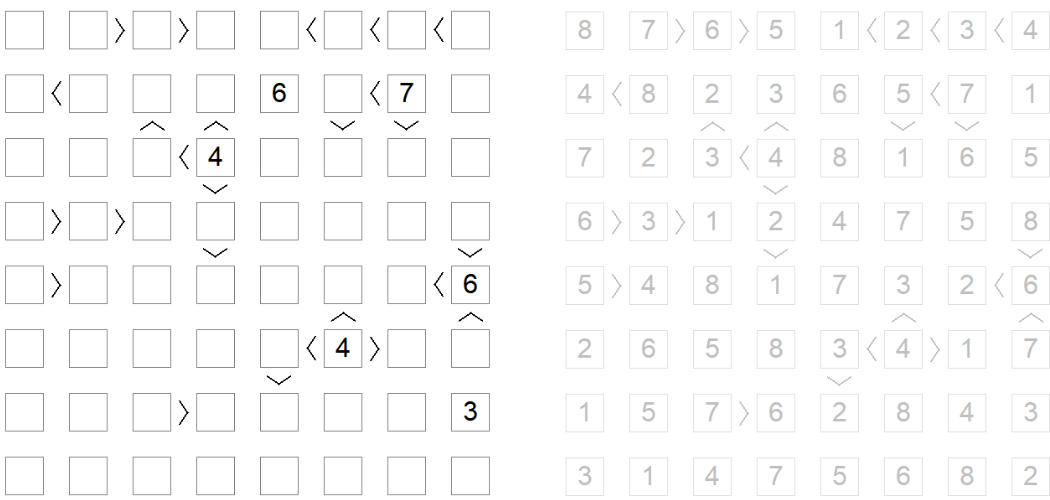


Figure 5: Futoshiki Test Case 4

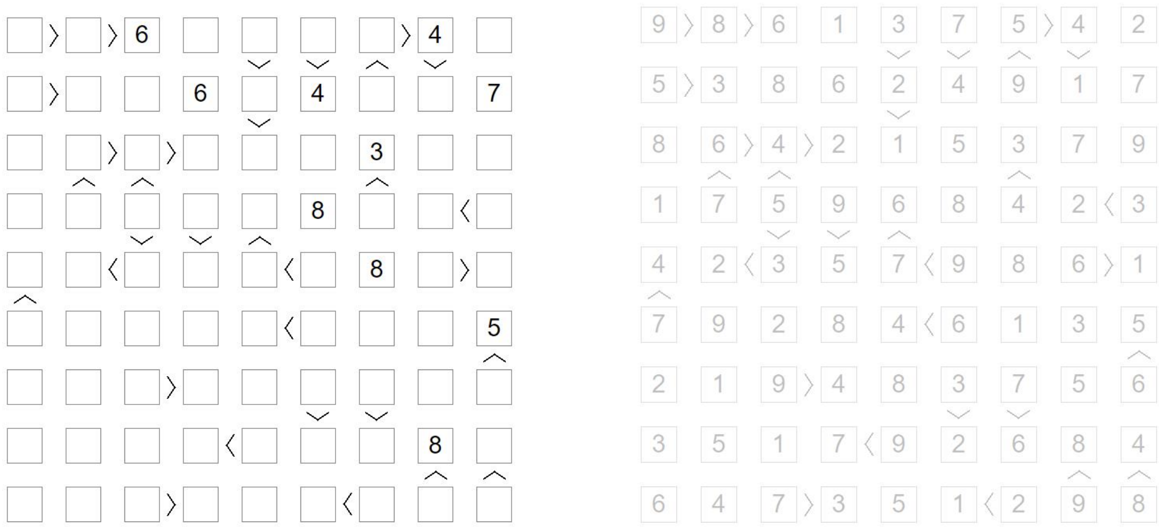


Figure 6: Futoshiki Test Case 5

# 2 Resolution

1. Implement the MGU algorithm. (10 points)
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. (10 points)
3. Explain any ideas you use to improve the search efficiency. (5 points)
4. Run your system on the examples of hardworker(sue), 3-blocks, Alpine Club. Include your input and output files in your report. (15 points)
5. What do you think are the main problems for using resolution to check for satisfiability for a set of first-order clauses? Explain. (10 points)

# 3 Results

## 3.1 Task-1.1

### 3.1.1 Main ideas of the GAC algorithm

- A variable $x$ is generalized arc consistent (GAC) with a constraint if every value of the variable can be extended to all the other variables of the constraint in such a way the constraint is satisfied.

- GAC algorithm enforces full arc-consistency on all uninstantiated variables following each tentative value assignment to the current variable.
- If a variable's domain becomes empty during the process of enforcing arc-consistency, then the current candidate value is rejected.
- Removing a value from a domain may trigger further inconsistency, so we have to repeat the procedure until everything is consistent.

#### 3.1.2 Main differences between the GAC and the FC algorithm

- The FC algorithm only checks a single unassigned variable at time for consistency, while the GAC algorithm checks pairs of unassigned variables for mutual consistency.
- The GAC algorithm may be more time-consuming but may produce better results.

### 3.2 Task-1.2

#### 3.2.1 Reason for Enforce GAC

- The main reason is that removing values from the domain of a variable may cause other variables to become no longer arc-consistent with it. Therefore, we have to check again for those constraints which contain the same variable.
- The operation supports that we can avoid having to search throuth all possible assignments to the other variables for a satisfying assignment, which is time-consuming and backtracking required.
- Rather than search for a satisfying assignment to constraint C containing V=d, we check to see if the current support is still valid: i.e., all values it assigns still lie in the variable's current domains, with this operation.

#### 3.2.2 Improvement for Enforce GAC

We can try to convert some of constraints into several common function constraints. The function constraints allow us to check each arc only once and reduce judgment times, improving the situation that we need to check again for those constraints which contain the same variable.

### 3.3 Task-1.3

"futoshiki.py"

```python
import time
import numpy as np

# 矩阵维度，矩阵，小于约束，可行域
N = 0
board = []
lessthan_constraints = []
current_domain = []

# 判断当前board是否全部放满
def is_allassigned():
    global N
    global board

    for row in range(N):
        for col in range(N):
            if board[row][col] == 0:
                return False

    return True

# 返回当前坐标涉及到的小于约束
def get_less_constraints(position):
    global lessthan_constraints

    # 目标结果
    constraints = []

    # 遍历所有小于约束
    for constraint in lessthan_constraints:
```

```
        if position == constraint[0] or position == constraint[1]:
            constraints.append(constraint)

    return constraints

# 根据小于约束从两个位置可行域中删除变量
def remove_variable_from_constraint(constraint):
    global current_domain

    less_position = constraint[0]
    position = constraint[1]

    # 保留被删除的变量用于回溯
    backup = []

    remove_list = []
    # 小于号左边位置的可行域
    for less_variable in current_domain[less_position[0]][less_position[1]]:
        remove_flag = True
        for variable in current_domain[position[0]][position[1]]:
            if variable > less_variable:
                remove_flag = False
                break
        if remove_flag:
            remove_list.append(less_variable)
    for remove_variable in remove_list:
        current_domain[less_position[0]][less_position[1]].remove(remove_variable)
        backup.append((less_position[0], less_position[1], remove_variable))
        if len(current_domain[less_position[0]][less_position[1]]) == 0:
            # DWO
```

```
            return backup, False

    remove_list.clear()
    # 小于号右边位置的可行域
    for variable in current_domain[position[0]][position[1]]:
        remove_flag = True
        for less_variable in current_domain[less_position[0]][
            less_position[1]]:
            if variable > less_variable:
                remove_flag = False
                break
        if remove_flag:
            remove_list.append(variable)
    for remove_variable in remove_list:
        current_domain[position[0]][position[1]].remove(remove_variable)
        backup.append((position[0], position[1], remove_variable))
        if len(current_domain[position[0]][position[1]]) == 0:
            # DWO
            return backup, False

    return backup, True

# 根据alldiff从一个位置的同行同列位置可行域中删除变量
def remove_variable_from_alldiff(position):
    global N
    global board
    global current_domain

    x = position[0]
    y = position[1]

    # 保留被删除的变量用于回溯
    backup = []
```

```python

    # 同列
    for index in range(N):
        if index == x:
            continue
        if board[x][y] in current_domain[index][y]:
            current_domain[index][y].remove(board[x][y])
            backup.append((index, y, board[x][y]))
            if len(current_domain[index][y]) == 0:
                # DWO
                return backup, False

    # 同行
    for index in range(N):
        if index == y:
            continue
        if board[x][y] in current_domain[x][index]:
            current_domain[x][index].remove(board[x][y])
            backup.append((x, index, board[x][y]))
            if len(current_domain[x][index]) == 0:
                # DWO
                return backup, False

    return backup, True

# 初始化可行域
def init():
    global N
    global board
    global lessthan_constraints
    global current_domain

    # 每一格均初始化为全部数字
```

```
    for row in range(N):
        row_current_domain = []
        for col in range(N):
            row_current_domain.append(list(range(N + 1))[1:])
        current_domain.append(row_current_domain)

    # 处理已赋值的位置：设置可行域为单个元素，并去除同行同列位置可行域中的该元素
    for row in range(N):
        for col in range(N):
            if board[row][col] != 0:
                # 已赋值
                current_domain[row][col] = [board[row][col]]
                remove_variable_from_alldiff((row, col))

    # 处理小于约束
    for constraint in lessthan_constraints:
        remove_variable_from_constraint(constraint)

# 打印结果
def print_result():
    global N
    global board

    for row in range(N):
        print("\t\t\t", board[row])

# 挑选一个unassigned的位置
def pick_unassigned_variable():
    global N
    global board

    for row in range(N):
```

```
            for col in range(N):
                if board[row][col] == 0:
                    return (row, col)

    return None

# 回溯
def BackTrace(backup):
    for x, y, digit in backup:
        current_domain[x][y].append(digit)

# FC记录信息
node_searched = 0
inference_total_time = 0.0

# FC recursive
def fc_recursive():
    global N
    global board
    global lessthan_constraints
    global current_domain
    global node_searched
    global inference_total_time

    # allassigned 直接return
    if is_allassigned():
        return True

    # 选出一个unassigned位置
    position = pick_unassigned_variable()
    variable_list = current_domain[position[0]][position[1]].copy()
    for variable in variable_list:
        # 遍历到一个节点
```

```
        inference_start_time = time.time()
        node_searched += 1
        board[position[0]][position[1]] = variable
        current_domain[position[0]][position[1]] = [variable]

        # 保存被删除的变量
        backup = []
        for temp_variable in variable_list:
            if temp_variable != variable:
                backup.append((position[0], position[1], temp_variable))

        # 处理约束
        ret = remove_variable_from_alldiff(position)
        backup.extend(ret[0])
        if not ret[1]:
            # DWO
            BackTrace(backup)
            board[position[0]][position[1]] = 0
            inference_total_time += time.time() - inference_start_time
            continue

        complete_flag = True
        constraints = get_less_constraints(position)
        for constraint in constraints:
            ret = remove_variable_from_constraint(constraint)
            backup.extend(ret[0])
            if not ret[1]:
                # DWO
                BackTrace(backup)
                board[position[0]][position[1]] = 0
                complete_flag = False
                break
        if not complete_flag:
```

```python
            inference_total_time += time.time() - inference_start_time
            continue

        # 下一轮
        inference_total_time += time.time() - inference_start_time
        if fc_recursive():
            return True
        else:
            BackTrace(backup)
            board[position[0]][position[1]] = 0

    return False

# FC algorithm
def ForwardChecking():
    global node_searched
    global inference_total_time

    fc_start_time = time.time()
    fc_recursive()
    fc_end_time = time.time()

    print("ForwardChecking Algorithm Total Time: {} seconds".format(fc_end_time - fc_start_time))
    print("            Number of Nodes Searched: {}".format(node_searched))
    print("     Average Inference Time Per Node: {}".format(inference_total_time / node_searched))

# 生成GAC需要的约束序列
def generate_constraint_queue():
    global lessthan_constraints
    global N
```

```python

    constraint_queue = []
    for constraint in lessthan_constraints:
        constraint_queue.append((constraint[0], constraint[1], "lt"))

    for row in range(N):
        constraint_queue.append((row, "row", "all-diff"))
    for col in range(N):
        constraint_queue.append((col, "col", "all-diff"))

    return constraint_queue

# 获取特定位置的所有未在队列内的有关约束
def generate_specific_constraint(position, constraint_queue):
    global N

    # 小于约束
    ret = get_less_constraints(position)
    for constraint in ret:
        if constraint not in constraint_queue:
            # print((constraint[0], constraint[1], "lt"))
            constraint_queue.append((constraint[0], constraint[1], "lt")
                )

    # 不等约束
    if (position[0], "row", "all-diff") not in constraint_queue:
        constraint_queue.append((position[0], "row", "all-diff"))
    if (position[1], "col", "all-diff") not in constraint_queue:
        constraint_queue.append((position[1], "col", "all-diff"))

    # exit()

# 检查all-diff约束是否满足
```

```python
def check_alldiff(domain):
    global N

    selected_number = []
    for index in range(N):
        if len(domain[index]) == 1:
            # 已赋值
            if domain[index][0] not in selected_number:
                selected_number.append((domain[index][0], index))

    while True:
        break_flag = True
        for number, number_index in selected_number:
            # print(number, number_index)
            for index in range(N):
                if (number in domain[index]) and (index != number_index):
                    domain[index].remove(number)
                    break_flag = False
                    if len(domain[index]) == 0:
                        return False
                    elif len(domain[index]) == 1:
                        if domain[index][0] not in selected_number:
                            selected_number.append((domain[index][0], index))
        if break_flag:
            break

    if len(selected_number) == N:
        return True

    for index in range(N):
        if len(domain[index]) > 1:
```

```
            # pickup one unassigned
            for variable in domain[index]:
                temp_domain = []
                for element in domain:
                    temp_domain.append(element.copy())
                temp_domain[index] = [variable]
                if check_alldiff(temp_domain):
                    return True
            # all False
            return False

# GAC algorithm
def GeneralizedArcConsistency():
    global N
    global board
    global lessthan_constraints
    global current_domain
    global node_searched
    global inference_total_time

    # 约束队列
    constraint_queue = generate_constraint_queue()

    gac_start_time = time.time()

    # 运行直至队列为空
    while len(constraint_queue) != 0:
        # 循环一次算一个节点
        node_searched += 1
        inference_start_time = time.time()

        constraint = constraint_queue[0]
        constraint_queue.remove(constraint_queue[0])
```

```
        # print(constraint)
        # print(current_domain)

        if constraint[2] == 'lt':
            # 小于约束
            x1, y1 = constraint[0]
            x2, y2 = constraint[1]

            push_flag1 = False
            push_flag2 = False

            remove_list = []
            for less_variable in current_domain[x1][y1]:
                remove_flag = True
                for variable in current_domain[x2][y2]:
                    if less_variable < variable:
                        remove_flag = False
                        break
                if remove_flag:
                    push_flag1 = True
                    remove_list.append(less_variable)
            for less_variable in remove_list:
                current_domain[x1][y1].remove(less_variable)

            remove_list.clear()
            for variable in current_domain[x2][y2]:
                remove_flag = True
                for less_variable in current_domain[x1][y1]:
                    if less_variable < variable:
                        remove_flag = False
                        break
                if remove_flag:
```

```python
                    push_flag2 = True
                    remove_list.append(variable)
            for variable in remove_list:
                current_domain[x2][y2].remove(variable)

            if push_flag1:
                generate_specific_constraint(constraint[0], constraint_queue)
            if push_flag2:
                generate_specific_constraint(constraint[1], constraint_queue)

        elif constraint[2] == 'all-diff':
            # 不等约束
            for index in range(N):
                domain = []
                if constraint[1] == 'row':
                    for index1 in range(N):
                        domain.append(current_domain[constraint[0]][index1].copy())
                elif constraint[1] == 'col':
                    for index1 in range(N):
                        domain.append(current_domain[index1][constraint[0]].copy())

                remove_list = []
                for variable in domain[index]:
                    temp_domain = []
                    for element in domain:
                        temp_domain.append(element.copy())
                    temp_domain[index] = [variable]
                    # print(temp_domain)
                    ret = check_alldiff(temp_domain)
```

```
                    # print(temp_domain, ret)
                    if not ret:
                        remove_list.append(variable)
                for variable in remove_list:
                    if constraint[1] == 'row':
                        current_domain[constraint[0]][index].remove(variable)
                        generate_specific_constraint((constraint[0], index), constraint_queue)
                    elif constraint[1] == 'col':
                        current_domain[index][constraint[0]].remove(variable)
                        generate_specific_constraint((index, constraint[0]), constraint_queue)

        inference_total_time += time.time() - inference_start_time

    gac_end_time = time.time()

    for row in range(N):
        for col in range(N):
            # print(current_domain[row][col])
            board[row][col] = current_domain[row][col][0]

    print("GeneralizedArcConsistency Algorithm Time Consuming: {} seconds".format(gac_end_time - gac_start_time))
    print("                  Number of Nodes Searched: {}".format(node_searched))
    print("       Average Inference Time Per Node: {}".format(inference_total_time / node_searched))

if __name__ == '__main__':

```

```python
    # 处理输入文件
    data_filedir = input("Please input source data file directory: ")
    data_filedir = "TestCase/data" + data_filedir + ".txt"

    with open(data_filedir, 'r') as data_file:
        for index, line in enumerate(data_file.readlines()):
            line = line.strip()
            if not line:
                continue
            # 第一行计算出矩阵维度
            if index == 0:
                N = len(line.split(' '))
                board = []

            line_split = line.split(' ')
            # 输入行要么是矩阵要么是小于约束
            if index < N:
                board.append([])
                for col in range(N):
                    board[index].append(int(line_split[col]))
            else:
                lessthan_constraints.append(((int(line_split[0]), int(
                    line_split[1])), (int(line_split[2]), int(line_split
                    [3]))))

    # 初始化可行域
    init()

    # 选择算法
    algorithm = input("Please input the algorithm to use: ")
    if algorithm == 'FC':
        ForwardChecking()
    elif algorithm == 'GAC':
```

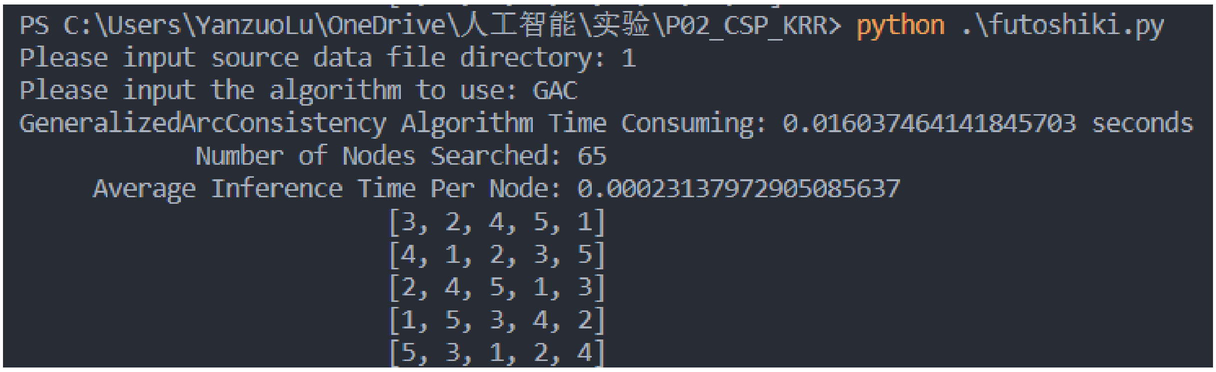
```
        GeneralizedArcConsistency()

    print_result()
```

### 3.4 Task-1.4

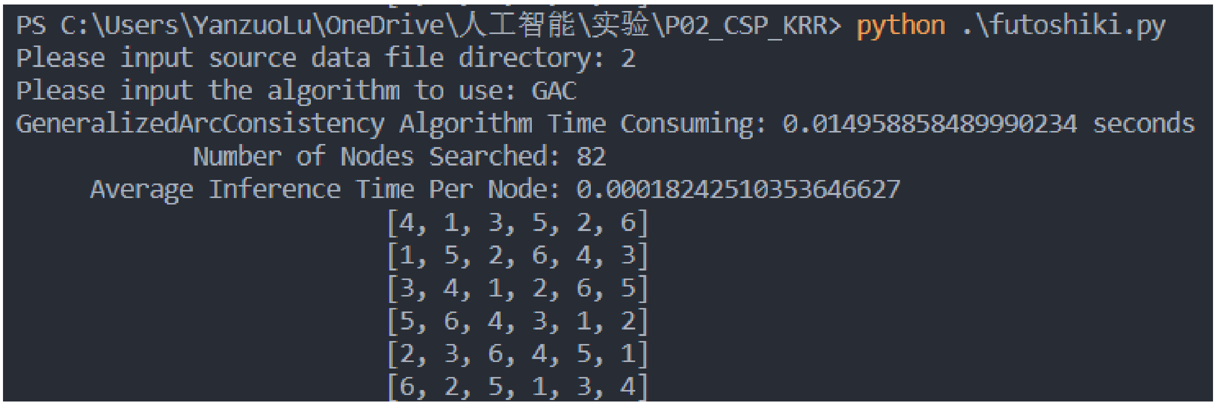
在 Task-1.3 中我用 Python 重新实现了 FC 和 GAC 两种算法，主要是为了对比更加方便，消除 C++ 和 Python 两种不同语言的差距，并编写了若干辅助函数便于调用和理解主要思想。在实现过程中，我注意到初始化的不同对结果有着重大的影响，因此我使用了一个小 trick 也就是在初始化的时候就对每个位置涉及到的约束都进行一次可行域 current_domain 削减，可以大幅降低所谓的搜索空间。经过实验发现，前后初始化的不同会导致推理时间大幅增长 (以 test5.txt 文件以及 GAC 算法为例，运行时间分别是是 600ms 和 840ms)，可见在初始搜索空间较大的情况下优化效果是十分明显的。

### 3.5 Task-1.5



```
PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\P02_CSP_KRR> python .\futoshiki.py
Please input source data file directory: 1
Please input the algorithm to use: GAC
GeneralizedArcConsistency Algorithm Time Consuming: 0.016037464141845703 seconds
                Number of Nodes Searched: 65
      Average Inference Time Per Node: 0.00023137972905085637
                    [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]
```

Figure 7: data1.txt



```
PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\P02_CSP_KRR> python .\futoshiki.py
Please input source data file directory: 2
Please input the algorithm to use: GAC
GeneralizedArcConsistency Algorithm Time Consuming: 0.014958858489990234 seconds
                Number of Nodes Searched: 82
      Average Inference Time Per Node: 0.00018242510353646627
                    [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]
```

Figure 8: data2.txt

```
PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\P02_CSP_KRR> python .\futoshiki.py
Please input source data file directory: 3
Please input the algorithm to use: GAC
GeneralizedArcConsistency Algorithm Time Consuming: 0.13877010345458984 seconds
            Number of Nodes Searched: 275
     Average Inference Time Per Node: 0.0005046185580166903
                         [2, 4, 3, 5, 1, 7, 6]
                         [4, 1, 6, 2, 7, 5, 3]
                         [1, 2, 5, 7, 3, 6, 4]
                         [5, 6, 7, 1, 4, 3, 2]
                         [7, 3, 2, 4, 6, 1, 5]
                         [3, 5, 1, 6, 2, 4, 7]
                         [6, 7, 4, 3, 5, 2, 1]
```

Figure 9: data3.txt

```
PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\P02_CSP_KRR> python .\futoshiki.py
Please input source data file directory: 4
Please input the algorithm to use: GAC
GeneralizedArcConsistency Algorithm Time Consuming: 0.1776282787322998 seconds
            Number of Nodes Searched: 268
     Average Inference Time Per Node: 0.0006627920848220142
                         [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]
```

Figure 10: data4.txt

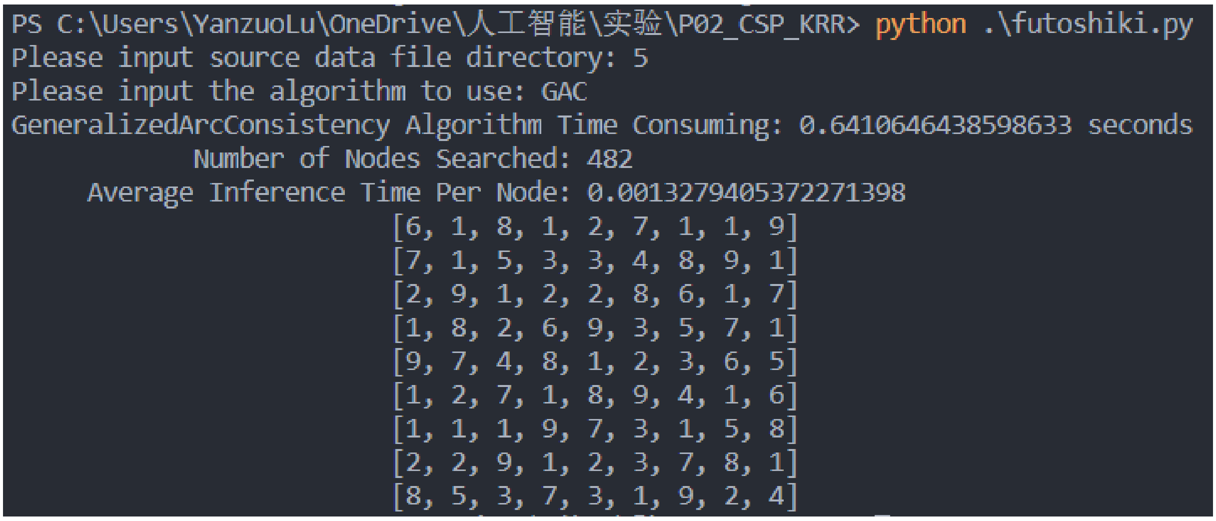



Figure 11: data5.txt

### 3.6 Task-1.6

| Test Case | Algorithm | Total Time | Number of Nodes Searched | Average Inference Time Per Node |
|---|---|---|---|---|
| 1 | FC | 0.004s | 113 | 3.58e-05s |
| | GAC | 0.021s | 65 | 3.30e-04s |
| 2 | FC | 0.006s | 48 | 1.36e-04s |
| | GAC | 0.016s | 82 | 1.95e-05s |
| 3 | FC | 3.50s | 168689 | 1.33e-05s |
| | GAC | 0.176s | 275 | 6.42e-04s |
| 4 | FC | 2.38s | 100986 | 1.66e-05s |
| | GAC | 0.233s | 268 | 8.70e-04s |
| 5 | FC | - | - | - |
| | GAC | 0.77s | 482 | 1.59e-03s |

### 3.7 Task-2.1

"MGU algorithm"

```
# 合一
def unifier(parser1, parser2):
    unifier_parser = []
    for index1, item1 in enumerate(parser1):
        for index2, item2 in enumerate(parser2):
```

```
        # print(item1, item2)
        not_flag1 = int('¬' in item1)
        not_flag2 = int('¬' in item2)
        if not_flag1 + not_flag2 != 1:
            continue
        # print(not_flag1, not_flag2)
        items1 = re.findall(r'[¬a-zA-Z]+', item1)
        items2 = re.findall(r'[¬a-zA-Z]+', item2)
        # print(items1, items2)
        predicate1 = items1[0][1:] if not_flag1 else items1[0]
        predicate2 = items2[0][1:] if not_flag2 else items2[0]
        # print(predicate1, predicate2)
        if predicate1 != predicate2:
            continue
        if len(items1) != len(items2):
            print("Error: Same predicate with different argument!")
            exit()

        check_flag = True
        unifier_result = []
        for argument_index in range(1, len(items1)):
            argument1 = items1[argument_index]
            argument2 = items2[argument_index]
            # print(argument1, argument2)
            # 单字母均认为是变量
            variable_flag1 = 1 if len(argument1) == 1 else 0
            variable_flag2 = 1 if len(argument2) == 1 else 0
            # print(variable_flag1, variable_flag2)
            if variable_flag1 + variable_flag2 == 2:
                check_flag = False
                break
            if variable_flag1 + variable_flag2 == 0:
                if argument1 != argument2:
```

```
                        check_flag = False
                        break
                    continue

                if variable_flag1:
                    unifier_result.append(argument1 + "=" + argument2)
                else:
                    unifier_result.append(argument2 + "=" + argument1)
                # print(unifier_result)

            if check_flag:
                for i in range(len(parser1)):
                    if i != index1 and parser1[i] not in unifier_parser:
                        unifier_parser.append(parser1[i])
                for j in range(len(parser2)):
                    if j != index2 and parser2[j] not in unifier_parser:
                        unifier_parser.append(parser2[j])
                assign(unifier_parser, unifier_result)
                unifier_parser = list(set(unifier_parser))
                unifier_pick = (chr(97+index1), chr(97+index2))
                return True, unifier_parser, unifier_result,
                    unifier_pick

    return False, None, None, None
```

该函数输入两个由 parser 解析的子句列表，输出分别是：两个子句能否合并、合并后的子句列表、合一字符串和被合并的子句序号。

### 3.8 Task-2.2

”mgu.py”

```
import re
import queue

```

```python
# 解析输入语句
def parser(clause):
    parser_result = []
    for item in re.findall(r'¬*[a-zA-Z]+\([a-zA-Z,\s]*\)', clause):
        parser_result.append(item)
    return parser_result

# parser赋值
def assign(uparser, uresult):
    assignment = []
    for index in range(len(uresult)):
        pos = uresult[index].find('=')
        variable = uresult[index][:pos]
        constant = uresult[index][pos+1:]
        assignment.append((variable, constant))
    # print(assignment)
    for index in range(len(uparser)):
        items = re.findall(r'[¬a-zA-Z]+', uparser[index])
        # print(items)
        for i, item in enumerate(items):
            if i == 0:
                continue
            for variable, constant in assignment:
                if item == variable:
                    items[i] = constant
                    break
        uparser[index] = items[0] + '(' + items[1]
        for item in items[2:]:
            uparser[index] += ", " + item
        uparser[index] += ')'
        # print(uparser[index])

#  合一
```

```python
def unifier(parser1, parser2):
    unifier_parser = []
    for index1, item1 in enumerate(parser1):
        for index2, item2 in enumerate(parser2):
            # print(item1, item2)
            not_flag1 = int('¬' in item1)
            not_flag2 = int('¬' in item2)
            if not_flag1 + not_flag2 != 1:
                continue
            # print(not_flag1, not_flag2)
            items1 = re.findall(r'[¬a-zA-Z]+', item1)
            items2 = re.findall(r'[¬a-zA-Z]+', item2)
            # print(items1, items2)
            predicate1 = items1[0][1:] if not_flag1 else items1[0]
            predicate2 = items2[0][1:] if not_flag2 else items2[0]
            # print(predicate1, predicate2)
            if predicate1 != predicate2:
                continue
            if len(items1) != len(items2):
                print("Error: Same predicate with different argument!")
                exit()

            check_flag = True
            unifier_result = []
            for argument_index in range(1, len(items1)):
                argument1 = items1[argument_index]
                argument2 = items2[argument_index]
                # print(argument1, argument2)
                # 单字母均认为是变量
                variable_flag1 = 1 if len(argument1) == 1 else 0
                variable_flag2 = 1 if len(argument2) == 1 else 0
                # print(variable_flag1, variable_flag2)
                if variable_flag1 + variable_flag2 == 2:
```

```
                    check_flag = False
                    break
                if variable_flag1 + variable_flag2 == 0:
                    if argument1 != argument2:
                        check_flag = False
                        break
                    continue

                if variable_flag1:
                    unifier_result.append(argument1 + "=" + argument2)
                else:
                    unifier_result.append(argument2 + "=" + argument1)
                # print(unifier_result)

            if check_flag:
                for i in range(len(parser1)):
                    if i != index1 and parser1[i] not in unifier_parser:
                        unifier_parser.append(parser1[i])
                for j in range(len(parser2)):
                    if j != index2 and parser2[j] not in unifier_parser:
                        unifier_parser.append(parser2[j])
                assign(unifier_parser, unifier_result)
                unifier_parser = list(set(unifier_parser))
                unifier_pick = (chr(97+index1), chr(97+index2))
                return True, unifier_parser, unifier_result,
                    unifier_pick

    return False, None, None, None

# 深搜
def dfs(clauses, results, records):
    # print(clauses)
    if clauses[-1] == "()":
```

```
        return clauses, results, records, True

    for i in range(len(clauses) - 1, -1, -1):
        for j in range(i - 1, -1, -1):
            if (i, j) in records:
                continue
            parser1 = parser(clauses[i])
            parser2 = parser(clauses[j])
            ret = unifier(parser1, parser2)

            if ret[0]:
                copy_clauses = clauses.copy()
                copy_results = results.copy()
                copy_records = records.copy()

                result = "R[" + str(i + 1)
                if len(parser1) != 1:
                    result += ret[3][0]
                result += "," + str(j + 1)
                if len(parser2) != 1:
                    result += ret[3][1]
                result += "]"
                if len(ret[2]) >= 1:
                    result += "(" + ret[2][0]
                for k in range(1, len(ret[2])):
                    result += "," + ret[2][k]
                if len(ret[2]) >= 1:
                    result += ")"

                clause = ret[1]
                # print(result, clause)
                if len(clause) > 1:
                    join_clause = "(" + ','.join(clause) + ")"
```

```
                elif len(clause) == 1:
                    join_clause = clause[0]
                elif len(clause) == 0:
                    join_clause = "()"

                if join_clause in copy_clauses:
                    continue

                copy_clauses.append(join_clause)
                copy_results.append(result)
                copy_records.append((i, j))
                # print(copy_clauses, copy_results, copy_records)

                dfs_ret = dfs(copy_clauses, copy_results, copy_records)
                # print(dfs_ret)
                if dfs_ret[3]:
                    return dfs_ret

    return clauses, results, records, False

def heuristic_function(clause):
    return len(parser(clause))

class Node(object):
    def __init__(self, clauses, results, records):
        self.clauses = clauses.copy()
        self.results = results.copy()
        self.records = records.copy()
        self.value = len(self.records) + heuristic_function(clauses[-1])
    def __lt__(self, other):
        return self.value < other.value

def astar(clauses, results, records):
```

```python
    pq = queue.PriorityQueue()
    pq.put(Node(clauses, results, records))
    while not pq.empty():
        node = pq.get()
        if node.clauses[-1] == "()":
            return node.clauses, node.results, node.records, True
        for i in range(len(node.clauses)):
            for j in range(i+1, len(node.clauses)):
                if (i, j) in records:
                    continue

                parser1 = parser(node.clauses[i])
                parser2 = parser(node.clauses[j])
                ret = unifier(parser1, parser2)

                if ret[0]:
                    copy_clauses = node.clauses.copy()
                    copy_results = node.results.copy()
                    copy_records = node.records.copy()

                    result = "R[" + str(i + 1)
                    if len(parser1) != 1:
                        result += ret[3][0]
                    result += "," + str(j + 1)
                    if len(parser2) != 1:
                        result += ret[3][1]
                    result += "]"
                    if len(ret[2]) >= 1:
                        result += "(" + ret[2][0]
                    for k in range(1, len(ret[2])):
                        result += "," + ret[2][k]
                    if len(ret[2]) >= 1:
                        result += ")"
```

```
                    clause = ret[1]
                    if len(clause) > 1:
                        join_clause = "(" + ','.join(clause) + ")"
                    elif len(clause) == 1:
                        join_clause = clause[0]
                    elif len(clause) == 0:
                        join_clause = "()"

                    if join_clause in copy_clauses:
                        continue

                    copy_clauses.append(join_clause)
                    copy_results.append(result)
                    copy_records.append((i, j))

                    pq.put(Node(copy_clauses, copy_results, copy_records
                        ))

    return clauses, results, records, False



if __name__ == "__main__":

    clauses = []
    results = []
    records = []

    clause_num = int(input("Please input the number of clauses: "))
    for i in range(clause_num):
        results.append("")
        clauses.append(input())
```

```python

    # clauses = ["GradStudent(sue)", "(¬GradStudent(x), Student(x))", "(¬Student(x), HardWorker(x))", "¬HardWorker(sue)"]
    # results = ["", "", "", ""]

    ret = astar(clauses, results, records)
    # ret = dfs(clauses, results, records)
    count = 1
    # print(clauses, results, records)
    for result, clause in zip(ret[1], ret[0]):
        print(str(count) + '.' + result + clause)
        count += 1
```

以上是包含了 mgu 算法函数的完整系统代码，输入是一组子句，输出是算法执行过程，格式和课堂要求是一致的。除此之外，我实现了 dfs 和 A* 两种算法，二者各有优劣，在下一部分讲述。

### 3.9 Task-2.3

起初我采取的 dfs 是比较直接的递归，从头到尾搜索当前的子句列表，若两两子句没有被合并过就尝试合并进入下一轮，但是这样带来的一个问题是如果输入子句过多或过于复杂，搜索空间极大，搜索到的第一个解中间会存在很多无用步骤，包含很多无用合并。因此我在搜索过程中使用了一个较为简单的转换，那就是从尾部向头部子句进行搜索，这是基于新合并的子句比原有子句更加可能逼近最优解的一个假设，实验结果来看也大幅改善了最终结果。

但是这还不够，我们依然能够发现最终结果尤其是最后一个例子会输出长度为 38 的解析过程，仍然包含许多无用的步骤，这时候就需要引入启发式函数来缩小搜索空间，我们可以想象到一个包含 N 个谓词的子句至少也要 N 个合并才能得到空子句，于是我的启发式函数就设计为最后一次得到子句的其中谓词的个数，事实证明我这个设计是比较成功的，输出的结果几乎逼近了最优解，但是缺点还是存在的，我们保存当前节点信息的代价是比较昂贵的，包括了子句、合并结果和合并记录三部分内容，这也导致即使 A* 能够找到比较优秀的解，但是耗时相对 DFS 来说是大得多的。

下面给出每一组例子的 DFS 和 A* 算法不同的输出结果，需要特别注意的是 A* 算法在第三个例子中时间相对来说是比较难以接受的，运算将近 15 秒钟，但 DFS 算法是几乎瞬间出结果。

### 3.10 Task-2.4

"hardworker-input"

```
Please input the number of clauses: 4
GradStudent(sue)
(¬GradStudent(x), Student(x))
(¬Student(x), HardWorker(x))
¬HardWorker(sue)
```

"hardworker-output-DFS"

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

"hardworker-output-A*"

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

"3block-input"

```
Please input the number of clauses: 5
On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(x,y), ¬Green(x), Green(y))
```

"3block-output-DFS"

```
1.On(aa,bb)
2.On(bb,cc)
3.Green(aa)
4.¬Green(cc)
5.(¬On(x,y), ¬Green(x), Green(y))
6.R[5c,4](y=cc)(¬On(x, cc),¬Green(x))
7.R[6b,3](x=aa)¬On(aa, cc)
8.R[6a,2](x=bb)¬Green(bb)
9.R[8,5c](y=bb)(¬On(x, bb),¬Green(x))
10.R[9b,3](x=aa)¬On(aa, bb)
11.R[10,1]()
```

"3block-output-A*"

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

"AlpineClub-input"

```
Please input the number of clauses: 11
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))
```

”AlpineClub-output-DFS”

```
1.A(tony)
2.A(mike)
3.A(john)
4.L(tony, rain)
5.L(tony, snow)
6.(¬A(x), S(x), C(x))
7.(¬C(y), ¬L(y, rain))
8.(L(z, snow), ¬S(z))
9.(¬L(tony, u), ¬L(mike, u))
10.(L(tony, v), L(mike, v))
11.(¬A(w), ¬C(w), S(w))
12.R[11a,3](w=john)(¬C(john),S(john))
13.R[12b,8b](z=john)(¬C(john),L(john, snow))
14.R[13a,6c](x=john)(S(john),¬A(john),L(john, snow))
15.R[14a,8b](z=john)(¬A(john),L(john, snow))
16.R[15a,3]L(john, snow)
17.R[14b,3](S(john),L(john, snow))
18.R[12a,6c](x=john)(S(john),¬A(john))
19.R[18b,3]S(john)
20.R[11a,2](w=mike)(¬C(mike),S(mike))
21.R[20b,8b](z=mike)(¬C(mike),L(mike, snow))
22.R[21b,9b](u=snow)(¬C(mike),¬L(tony, snow))
23.R[22b,8a](z=tony)(¬C(mike),¬S(tony))
24.R[23b,11c](w=tony)(¬C(mike),¬C(tony),¬A(tony))
25.R[24a,6c](x=mike)(¬A(mike),¬C(tony),S(mike),¬A(tony))
26.R[25c,8b](z=mike)(¬A(mike),L(mike, snow),¬C(tony),¬A(tony))
27.R[26b,9b](u=snow)(¬A(mike),¬L(tony, snow),¬C(tony),¬A(tony))
```

```
28.R[27b,8a](z=tony)(¬A(mike),¬S(tony),¬C(tony),¬A(tony))
29.R[28b,11c](w=tony)(¬A(mike),¬C(tony),¬A(tony))
30.R[29b,6c](x=tony)(¬A(mike),S(tony),¬A(tony))
31.R[30b,23b](¬A(mike),¬C(mike),¬A(tony))
32.R[31b,6c](x=mike)(¬A(mike),S(mike),¬A(tony))
33.R[32b,8b](z=mike)(¬A(mike),L(mike, snow),¬A(tony))
34.R[33b,9b](u=snow)(¬A(mike),¬L(tony, snow),¬A(tony))
35.R[34b,8a](z=tony)(¬A(mike),¬S(tony),¬A(tony))
36.R[35b,30b](¬A(mike),¬A(tony))
37.R[36a,2]¬A(tony)
38.R[37,1]()
```

”AlpineClub-output-A*”

```
1.A(tony)
2.A(mike)
3.A(john)
4.L(tony, rain)
5.L(tony, snow)
6.(¬A(x), S(x), C(x))
7.(¬C(y), ¬L(y, rain))
8.(L(z, snow), ¬S(z))
9.(¬L(tony, u), ¬L(mike, u))
10.(L(tony, v), L(mike, v))
11.(¬A(w), ¬C(w), S(w))
12.R[2,11a](w=mike)(S(mike),¬C(mike))
13.R[5,9a](u=snow)¬L(mike, snow)
14.R[6c,12b](x=mike)(S(mike),¬A(mike))
15.R[2,14b]S(mike)
16.R[8b,15](z=mike)L(mike, snow)
17.R[13,16]()
```

### 3.11 Task-2.5

我认为在做 resolution 过程中遇到的一个需要做出抉择的一个地方是，我们追求的到底是最优解还是最优速度，就以我的实现来看（只关注第三个例子，比较具有代表性）。DFS 当然是一种解决方法，因为无论怎么去做这个合并过程，只要我做了环检测，确保不会出现相同的合并，那我只要搜的够深几乎是一定能出结果的，因为每一步都是合法的，最后总归能归出一个空子句来，但也正是因为这个点，合并出了大量的无用子句，即使我可以去对最终结果进行优化去除那些无用子句，最后解也不太可能是最优解。

相反地，在 Astar 算法中我们引入了启发式函数这一概念，预估某个子句到空子句所需的操作数，其实这就是基于 BFS 的一种算法，最后输出的解和最优解是十分逼近的，只要启发式函数足够优秀，可采纳性和一致性都满足，甚至可以保证就是最优解，但这也就带来一个问题，我们的搜索空间在例子中太大了，遍历同一层的所有节点十分耗费时间，时间复杂度和空间复杂度都大幅上升，更不用说维护一个优先队列所需要的资源。

综上所述，在实际运用中我认为我们需要根据问题的需要来选择不同的算法，上述实现未必完美，但我认为是足够说明我提出的两个问题的，或许我们可以尝试在两者中找一个折衷的方案，使得最终解不至于太差，运算时间也不会太长。