

P02 CSP and KRR

18364066 Yanzuo Lu

October 28, 2020

Contents

1	Futoshiki (GAC, C++/Python)	3
1.1	Description	3
1.2	Tasks	3
2	Resolution	6
3	Results	6
3.1	Task-1.1	6
3.1.1	Main ideas of the GAC algorithm	6
3.1.2	Main differences between the GAC and the FC algorithm	7
3.2	Task-1.2	7
3.2.1	Reason for Enforce GAC	7
3.2.2	Improvement for Enforce GAC	7
3.3	Task-1.3	8
3.4	Task-1.4	23
3.5	Task-1.5	23
3.6	Task-1.6	25

3.7	Task-2.1	25
3.8	Task-2.2	27
3.9	Task-2.3	35
3.10	Task-2.4	35
3.11	Task-2.5	40

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 (4×4 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 $\text{CurDom}[V]$, push all constraints C' such that $V \in \text{scope}(C')$ and $C' \notin \text{GACQueue}$ 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

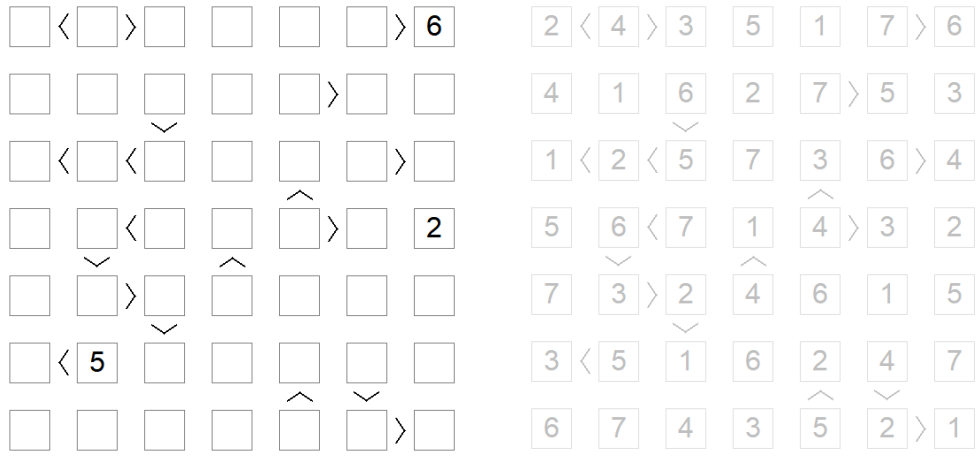


Figure 4: Futoshiki Test Case 3

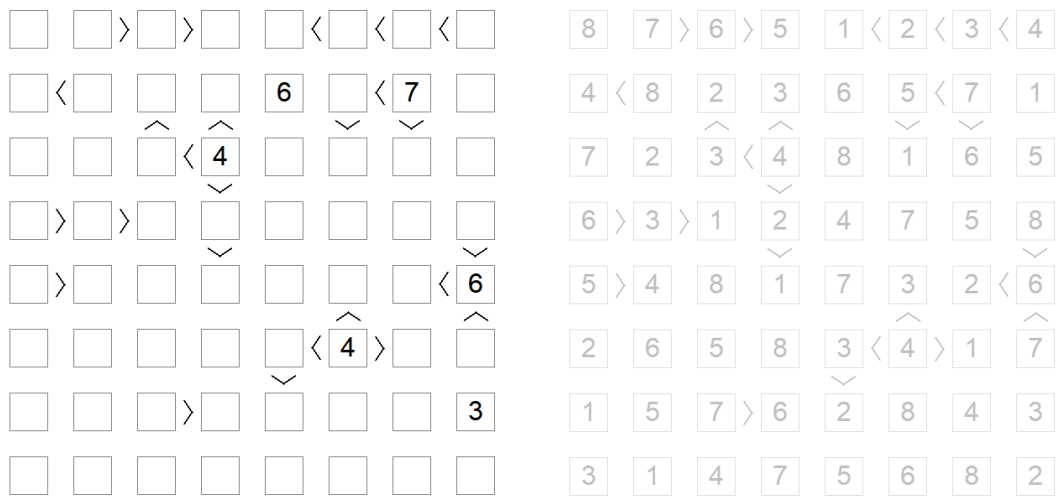


Figure 5: Futoshiki Test Case 4



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 through 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"

```
1 import time
2 import numpy as np
3
4 # 矩阵维度, 矩阵, 小于约束, 可行域
5 N = 0
6 board = []
7 lessthan_constraints = []
8 current_domain = []
9
10 # 判断当前board是否全部放满
11 def is_assigned():
12     global N
13     global board
14
15     for row in range(N):
16         for col in range(N):
17             if board[row][col] == 0:
18                 return False
19
20     return True
21
22 # 返回当前坐标涉及到的小于约束
23 def get_less_constraints(position):
24     global lessthan_constraints
25
26     # 目标结果
27     constraints = []
28
29     # 遍历所有小于约束
30     for constraint in lessthan_constraints:
```



```

31         if position == constraint[0] or position == constraint[1]:
32             constraints.append(constraint)
33
34     return constraints
35
36 # 根据小于约束从两个位置可行域中删除变量
37 def remove_variable_from_constraint(constraint):
38     global current_domain
39
40     less_position = constraint[0]
41     position = constraint[1]
42
43     # 保留被删除的变量用于回溯
44     backup = []
45
46     remove_list = []
47     # 小于号左边位置的可行域
48     for less_variable in current_domain[less_position[0]][less_position
49         [1]]:
50         remove_flag = True
51         for variable in current_domain[position[0]][position[1]]:
52             if variable > less_variable:
53                 remove_flag = False
54                 break
55         if remove_flag:
56             remove_list.append(less_variable)
57     for remove_variable in remove_list:
58         current_domain[less_position[0]][less_position[1]].remove(
59             remove_variable)
60         backup.append((less_position[0], less_position[1],
61             remove_variable))
62     if len(current_domain[less_position[0]][less_position[1]]) == 0:
63         # DWO

```

```

61         return backup, False
62
63     remove_list.clear()
64     # 小于号右边位置的可行域
65     for variable in current_domain[position[0]][position[1]]:
66         remove_flag = True
67         for less_variable in current_domain[less_position[0]][
68             less_position[1]]:
69             if variable > less_variable:
70                 remove_flag = False
71                 break
72         if remove_flag:
73             remove_list.append(variable)
74     for remove_variable in remove_list:
75         current_domain[position[0]][position[1]].remove(remove_variable)
76         backup.append((position[0], position[1], remove_variable))
77         if len(current_domain[position[0]][position[1]]) == 0:
78             # DWO
79             return backup, False
80     return backup, True
81
82 # 根据 alldiff 从一个位置的同行同列位置可行域中删除变量
83 def remove_variable_from_alldiff(position):
84     global N
85     global board
86     global current_domain
87
88     x = position[0]
89     y = position[1]
90
91     # 保留被删除的变量用于回溯
92     backup = []

```

```

93
94     # 同列
95     for index in range(N):
96         if index == x:
97             continue
98         if board[x][y] in current_domain[index][y]:
99             current_domain[index][y].remove(board[x][y])
100             backup.append((index, y, board[x][y]))
101             if len(current_domain[index][y]) == 0:
102                 # DWO
103                 return backup, False
104
105     # 同行
106     for index in range(N):
107         if index == y:
108             continue
109         if board[x][y] in current_domain[x][index]:
110             current_domain[x][index].remove(board[x][y])
111             backup.append((x, index, board[x][y]))
112             if len(current_domain[x][index]) == 0:
113                 # DWO
114                 return backup, False
115
116     return backup, True
117
118 # 初始化可行域
119 def init():
120     global N
121     global board
122     global lessthan_constraints
123     global current_domain
124
125     # 每一格均初始化为全部数字

```

```

126     for row in range(N):
127         row_current_domain = []
128         for col in range(N):
129             row_current_domain.append(list(range(N + 1))[1:])
130             current_domain.append(row_current_domain)
131
132     # 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
133     for row in range(N):
134         for col in range(N):
135             if board[row][col] != 0:
136                 # 已赋值
137                 current_domain[row][col] = [board[row][col]]
138                 remove_variable_from_alldiff((row, col))
139
140     # 处理小于约束
141     for constraint in lessthan_constraints:
142         remove_variable_from_constraint(constraint)
143
144 # 打印结果
145 def print_result():
146     global N
147     global board
148
149     for row in range(N):
150         print("\t\t\t", board[row])
151
152 # 挑选一个 unassigned 的位置
153 def pick_unassigned_variable():
154     global N
155     global board
156
157     for row in range(N):

```

```

158         for col in range(N):
159             if board[row][col] == 0:
160                 return (row, col)
161
162     return None
163
164 # 回溯
165 def BackTrace(backup):
166     for x, y, digit in backup:
167         current_domain[x][y].append(digit)
168
169 # FC记录信息
170 node_searched = 0
171 inference_total_time = 0.0
172
173 # FC recursive
174 def fc_recursive():
175     global N
176     global board
177     global lessthan_constraints
178     global current_domain
179     global node_searched
180     global inference_total_time
181
182     # allassigned 直接return
183     if is_allassigned():
184         return True
185
186     # 选出一个unassigned位置
187     position = pick_unassigned_variable()
188     variable_list = current_domain[position[0]][position[1]].copy()
189     for variable in variable_list:
190         # 遍历到一个节点

```

```

191     inference_start_time = time.time()
192     node_searched += 1
193     board[position[0]][position[1]] = variable
194     current_domain[position[0]][position[1]] = [variable]
195
196     # 保存被删除的变量
197     backup = []
198     for temp_variable in variable_list:
199         if temp_variable != variable:
200             backup.append((position[0], position[1], temp_variable))
201
202     # 处理约束
203     ret = remove_variable_from_alldiff(position)
204     backup.extend(ret[0])
205     if not ret[1]:
206         # DWO
207         BackTrace(backup)
208         board[position[0]][position[1]] = 0
209         inference_total_time += time.time() - inference_start_time
210         continue
211
212     complete_flag = True
213     constraints = get_less_constraints(position)
214     for constraint in constraints:
215         ret = remove_variable_from_constraint(constraint)
216         backup.extend(ret[0])
217         if not ret[1]:
218             # DWO
219             BackTrace(backup)
220             board[position[0]][position[1]] = 0
221             complete_flag = False
222             break
223     if not complete_flag:

```

```

224         inference_total_time += time.time() - inference_start_time
225         continue
226
227     # 下一轮
228     inference_total_time += time.time() - inference_start_time
229     if fc_recursive():
230         return True
231     else:
232         BackTrace(backup)
233         board[position[0]][position[1]] = 0
234
235     return False
236
237 # FC algorithm
238 def ForwardChecking():
239     global node_searched
240     global inference_total_time
241
242     fc_start_time = time.time()
243     fc_recursive()
244     fc_end_time = time.time()
245
246     print("ForwardChecking Algorithm Total Time: {} seconds".format(
247         fc_end_time - fc_start_time))
248     print("          Number of Nodes Searched: {}".format(
249         node_searched))
250     print("          Average Inference Time Per Node: {}".format(
251         inference_total_time / node_searched))
252
253 # 生成GAC需要的约束序列
254 def generate_constraint_queue():
255     global lessthan_constraints
256     global N

```

```

254
255     constraint_queue = []
256     for constraint in lessthan_constraints:
257         constraint_queue.append((constraint[0], constraint[1], "lt"))
258
259     for row in range(N):
260         constraint_queue.append((row, "row", "all-diff"))
261     for col in range(N):
262         constraint_queue.append((col, "col", "all-diff"))
263
264     return constraint_queue
265
266 # 获取特定位置的所有未在队列内的有关约束
267 def generate_specific_constraint(position, constraint_queue):
268     global N
269
270     # 小于约束
271     ret = get_less_constraints(position)
272     for constraint in ret:
273         if constraint not in constraint_queue:
274             # print((constraint[0], constraint[1], "lt"))
275             constraint_queue.append((constraint[0], constraint[1], "lt"))
276
277     # 不等约束
278     if (position[0], "row", "all-diff") not in constraint_queue:
279         constraint_queue.append((position[0], "row", "all-diff"))
280     if (position[1], "col", "all-diff") not in constraint_queue:
281         constraint_queue.append((position[1], "col", "all-diff"))
282
283     # exit()
284
285 # 检查 all-diff 约束是否满足

```



```

286 def check_alldiff(domain):
287     global N
288
289     selected_number = []
290     for index in range(N):
291         if len(domain[index]) == 1:
292             # 已赋值
293             if domain[index][0] not in selected_number:
294                 selected_number.append((domain[index][0], index))
295
296     while True:
297         break_flag = True
298         for number, number_index in selected_number:
299             # print(number, number_index)
300             for index in range(N):
301                 if (number in domain[index]) and (index != number_index):
302                     :
303                     domain[index].remove(number)
304                     break_flag = False
305                     if len(domain[index]) == 0:
306                         return False
307                     elif len(domain[index]) == 1:
308                         if domain[index][0] not in selected_number:
309                             selected_number.append((domain[index][0],
310                                                         index))
311
312             if break_flag:
313                 break
314
315     if len(selected_number) == N:
316         return True
317
318     for index in range(N):
319         if len(domain[index]) > 1:

```

```

317         # pickup one unassigned
318         for variable in domain[index]:
319             temp_domain = []
320             for element in domain:
321                 temp_domain.append(element.copy())
322             temp_domain[index] = [variable]
323             if check_alldiff(temp_domain):
324                 return True
325         # all False
326         return False
327
328 # GAC algorithm
329 def GeneralizedArcConsistency():
330     global N
331     global board
332     global lessthan_constraints
333     global current_domain
334     global node_searched
335     global inference_total_time
336
337     # 约束队列
338     constraint_queue = generate_constraint_queue()
339
340     gac_start_time = time.time()
341
342     # 运行直至队列为空
343     while len(constraint_queue) != 0:
344         # 循环一次算一个节点
345         node_searched += 1
346         inference_start_time = time.time()
347
348         constraint = constraint_queue[0]
349         constraint_queue.remove(constraint_queue[0])

```

```

350
351     # print(constraint)
352     # print(current_domain)
353
354     if constraint[2] == 'lt':
355         # 小于约束
356         x1, y1 = constraint[0]
357         x2, y2 = constraint[1]
358
359         push_flag1 = False
360         push_flag2 = False
361
362         remove_list = []
363         for less_variable in current_domain[x1][y1]:
364             remove_flag = True
365             for variable in current_domain[x2][y2]:
366                 if less_variable < variable:
367                     remove_flag = False
368                     break
369             if remove_flag:
370                 push_flag1 = True
371                 remove_list.append(less_variable)
372         for less_variable in remove_list:
373             current_domain[x1][y1].remove(less_variable)
374
375         remove_list.clear()
376         for variable in current_domain[x2][y2]:
377             remove_flag = True
378             for less_variable in current_domain[x1][y1]:
379                 if less_variable < variable:
380                     remove_flag = False
381                     break
382             if remove_flag:

```

```

383         push_flag2 = True
384         remove_list.append(variable)
385     for variable in remove_list:
386         current_domain[x2][y2].remove(variable)
387
388     if push_flag1:
389         generate_specific_constraint(constraint[0],
390                                     constraint_queue)
391     if push_flag2:
392         generate_specific_constraint(constraint[1],
393                                     constraint_queue)
394
395 elif constraint[2] == 'all-diff':
396     # 不等约束
397     for index in range(N):
398         domain = []
399         if constraint[1] == 'row':
400             for index1 in range(N):
401                 domain.append(current_domain[constraint[0]][
402                             index1].copy())
403         elif constraint[1] == 'col':
404             for index1 in range(N):
405                 domain.append(current_domain[index1][constraint
406                             [0]].copy())
407
408     remove_list = []
409     for variable in domain[index]:
410         temp_domain = []
411         for element in domain:
412             temp_domain.append(element.copy())
413         temp_domain[index] = [variable]
414         # print(temp_domain)
415         ret = check_alldiff(temp_domain)

```

```

412         # print(temp_domain, ret)
413         if not ret:
414             remove_list.append(variable)
415         for variable in remove_list:
416             if constraint[1] == 'row':
417                 current_domain[constraint[0]][index].remove(
418                     variable)
419                 generate_specific_constraint((constraint[0],
420                     index), constraint_queue)
421             elif constraint[1] == 'col':
422                 current_domain[index][constraint[0]].remove(
423                     variable)
424                 generate_specific_constraint((index, constraint
425                     [0]), constraint_queue)
426
427         inference_total_time += time.time() - inference_start_time
428
429     gac_end_time = time.time()
430
431     for row in range(N):
432         for col in range(N):
433             # print(current_domain[row][col])
434             board[row][col] = current_domain[row][col][0]
435
436     print("GeneralizedArcConsistency Algorithm Time Consuming: {}
437         seconds".format(gac_end_time - gac_start_time))
438     print("Number of Nodes Searched: {}".format(
439         node_searched))
440     print("Average Inference Time Per Node: {}".format(
441         inference_total_time / node_searched))
442
443 if __name__ == '__main__':
444

```

```

438 # 处理输入文件
439 data_filedir = input("Please input source data file directory: ")
440 data_filedir = "TestCase/data" + data_filedir + ".txt"
441
442 with open(data_filedir, 'r') as data_file:
443     for index, line in enumerate(data_file.readlines()):
444         line = line.strip()
445         if not line:
446             continue
447         # 第一行计算出矩阵维度
448         if index == 0:
449             N = len(line.split(' '))
450             board = []
451
452             line_split = line.split(' ')
453             # 输入行要么是矩阵要么是小于约束
454             if index < N:
455                 board.append([])
456                 for col in range(N):
457                     board[index].append(int(line_split[col]))
458             else:
459                 lessthan_constraints.append(((int(line_split[0]), int(
460                     line_split[1])), (int(line_split[2]), int(line_split
461                     [3]))))
462
463 # 初始化可行域
464 init()
465
466 # 选择算法
467 algorithm = input("Please input the algorithm to use: ")
468 if algorithm == 'FC':
469     ForwardChecking()
470 elif algorithm == 'GAC':

```

```
469         GeneralizedArcConsistency ()
470
471     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 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


```

PS C:\Users\YanzuoLu\OneDrive\人工智能\实验\p02_CSP_KRR> python .\futoshiki.py
Please input source data file directory: 5
Please input the algorithm to use: GAC
GeneralizedArcConsistency Algorithm Time Consuming: 0.6410646438598633 seconds
Number of Nodes Searched: 482
Average Inference Time Per Node: 0.0013279405372271398
[6, 1, 8, 1, 2, 7, 1, 1, 9]
[7, 1, 5, 3, 3, 4, 8, 9, 1]
[2, 9, 1, 2, 2, 8, 6, 1, 7]
[1, 8, 2, 6, 9, 3, 5, 7, 1]
[9, 7, 4, 8, 1, 2, 3, 6, 5]
[1, 2, 7, 1, 8, 9, 4, 1, 6]
[1, 1, 1, 9, 7, 3, 1, 5, 8]
[2, 2, 9, 1, 2, 3, 7, 8, 1]
[8, 5, 3, 7, 3, 1, 9, 2, 4]

```

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"

```

1 # 合一
2 def unifier(parser1, parser2):
3     unifier_parser = []
4     for index1, item1 in enumerate(parser1):
5         for index2, item2 in enumerate(parser2):

```

```

6      # print(item1, item2)
7      not_flag1 = int('¬' in item1)
8      not_flag2 = int('¬' in item2)
9      if not_flag1 + not_flag2 != 1:
10         continue
11     # print(not_flag1, not_flag2)
12     items1 = re.findall(r'[\¬a-zA-Z]+', item1)
13     items2 = re.findall(r'[\¬a-zA-Z]+', item2)
14     # print(items1, items2)
15     predicate1 = items1[0][1:] if not_flag1 else items1[0]
16     predicate2 = items2[0][1:] if not_flag2 else items2[0]
17     # print(predicate1, predicate2)
18     if predicate1 != predicate2:
19         continue
20     if len(items1) != len(items2):
21         print("Error: Same predicate with different argument!")
22         exit()
23
24     check_flag = True
25     unifier_result = []
26     for argument_index in range(1, len(items1)):
27         argument1 = items1[argument_index]
28         argument2 = items2[argument_index]
29         # print(argument1, argument2)
30         # 单字母均认为是变量
31         variable_flag1 = 1 if len(argument1) == 1 else 0
32         variable_flag2 = 1 if len(argument2) == 1 else 0
33         # print(variable_flag1, variable_flag2)
34         if variable_flag1 + variable_flag2 == 2:
35             check_flag = False
36             break
37         if variable_flag1 + variable_flag2 == 0:
38             if argument1 != argument2:

```

```

39         check_flag = False
40         break
41     continue
42
43     if variable_flag1:
44         unifier_result.append(argument1 + "=" + argument2)
45     else:
46         unifier_result.append(argument2 + "=" + argument1)
47     # print(unifier_result)
48
49     if check_flag:
50         for i in range(len(parser1)):
51             if i != index1 and parser1[i] not in unifier_parser:
52                 unifier_parser.append(parser1[i])
53         for j in range(len(parser2)):
54             if j != index2 and parser2[j] not in unifier_parser:
55                 unifier_parser.append(parser2[j])
56         assign(unifier_parser, unifier_result)
57         unifier_parser = list(set(unifier_parser))
58         unifier_pick = (chr(97+index1), chr(97+index2))
59         return True, unifier_parser, unifier_result,
60             unifier_pick
61     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"

```

1 import re
2 import queue
3

```

```

4 # 解析输入语句
5 def parser(clause):
6     parser_result = []
7     for item in re.findall(r'¬*[a-zA-Z]+\([a-zA-Z,\s]*\) ', clause):
8         parser_result.append(item)
9     return parser_result
10
11 # parser赋值
12 def assign(uparser, uresult):
13     assignment = []
14     for index in range(len(uresult)):
15         pos = uresult[index].find('=')
16         variable = uresult[index][:pos]
17         constant = uresult[index][pos+1:]
18         assignment.append((variable, constant))
19     # print(assignment)
20     for index in range(len(uparser)):
21         items = re.findall(r'¬[a-zA-Z]+', uparser[index])
22         # print(items)
23         for i, item in enumerate(items):
24             if i == 0:
25                 continue
26                 for variable, constant in assignment:
27                     if item == variable:
28                         items[i] = constant
29                         break
30         uparser[index] = items[0] + '(' + items[1]
31         for item in items[2:]:
32             uparser[index] += ", " + item
33         uparser[index] += ')'
34         # print(uparser[index])
35
36 # 合一

```

```

37 def unifier(parser1, parser2):
38     unifier_parser = []
39     for index1, item1 in enumerate(parser1):
40         for index2, item2 in enumerate(parser2):
41             # print(item1, item2)
42             not_flag1 = int('¬' in item1)
43             not_flag2 = int('¬' in item2)
44             if not_flag1 + not_flag2 != 1:
45                 continue
46             # print(not_flag1, not_flag2)
47             items1 = re.findall(r'[\¬a-zA-Z]+', item1)
48             items2 = re.findall(r'[\¬a-zA-Z]+', item2)
49             # print(items1, items2)
50             predicate1 = items1[0][1:] if not_flag1 else items1[0]
51             predicate2 = items2[0][1:] if not_flag2 else items2[0]
52             # print(predicate1, predicate2)
53             if predicate1 != predicate2:
54                 continue
55             if len(items1) != len(items2):
56                 print("Error: Same predicate with different argument!")
57                 exit()
58
59             check_flag = True
60             unifier_result = []
61             for argument_index in range(1, len(items1)):
62                 argument1 = items1[argument_index]
63                 argument2 = items2[argument_index]
64                 # print(argument1, argument2)
65                 # 单字母均认为是变量
66                 variable_flag1 = 1 if len(argument1) == 1 else 0
67                 variable_flag2 = 1 if len(argument2) == 1 else 0
68                 # print(variable_flag1, variable_flag2)
69                 if variable_flag1 + variable_flag2 == 2:

```

```

70         check_flag = False
71         break
72     if variable_flag1 + variable_flag2 == 0:
73         if argument1 != argument2:
74             check_flag = False
75             break
76         continue
77
78     if variable_flag1:
79         unifier_result.append(argument1 + "=" + argument2)
80     else:
81         unifier_result.append(argument2 + "=" + argument1)
82     # print(unifier_result)
83
84     if check_flag:
85         for i in range(len(parser1)):
86             if i != index1 and parser1[i] not in unifier_parser:
87                 unifier_parser.append(parser1[i])
88         for j in range(len(parser2)):
89             if j != index2 and parser2[j] not in unifier_parser:
90                 unifier_parser.append(parser2[j])
91         assign(unifier_parser, unifier_result)
92         unifier_parser = list(set(unifier_parser))
93         unifier_pick = (chr(97+index1), chr(97+index2))
94         return True, unifier_parser, unifier_result,
95             unifier_pick
96     return False, None, None, None
97
98 # 深搜
99 def dfs(clauses, results, records):
100     # print(clauses)
101     if clauses[-1] == "(":

```

```

102         return clauses, results, records, True
103
104     for i in range(len(clauses) - 1, -1, -1):
105         for j in range(i - 1, -1, -1):
106             if (i, j) in records:
107                 continue
108             parser1 = parser(clauses[i])
109             parser2 = parser(clauses[j])
110             ret = unifier(parser1, parser2)
111
112             if ret[0]:
113                 copy_clauses = clauses.copy()
114                 copy_results = results.copy()
115                 copy_records = records.copy()
116
117                 result = "R[" + str(i + 1)
118                 if len(parser1) != 1:
119                     result += ret[3][0]
120                 result += "," + str(j + 1)
121                 if len(parser2) != 1:
122                     result += ret[3][1]
123                 result += "]"
124                 if len(ret[2]) >= 1:
125                     result += "(" + ret[2][0]
126                     for k in range(1, len(ret[2])):
127                         result += "," + ret[2][k]
128                     if len(ret[2]) >= 1:
129                         result += ")"
130
131                 clause = ret[1]
132                 # print(result, clause)
133                 if len(clause) > 1:
134                     join_clause = "(" + ', '.join(clause) + ")"

```

```

135         elif len(clause) == 1:
136             join_clause = clause[0]
137         elif len(clause) == 0:
138             join_clause = "()"
139
140         if join_clause in copy_clauses:
141             continue
142
143         copy_clauses.append(join_clause)
144         copy_results.append(result)
145         copy_records.append((i, j))
146         # print(copy_clauses, copy_results, copy_records)
147
148         dfs_ret = dfs(copy_clauses, copy_results, copy_records)
149         # print(dfs_ret)
150         if dfs_ret[3]:
151             return dfs_ret
152
153     return clauses, results, records, False
154
155 def heuristic_function(clause):
156     return len(parser(clause))
157
158 class Node(object):
159     def __init__(self, clauses, results, records):
160         self.clauses = clauses.copy()
161         self.results = results.copy()
162         self.records = records.copy()
163         self.value = len(self.records) + heuristic_function(clauses[-1])
164     def __lt__(self, other):
165         return self.value < other.value
166
167 def astar(clauses, results, records):

```



```

168 pq = queue.PriorityQueue()
169 pq.put(Node(clauses , results , records))
170 while not pq.empty():
171     node = pq.get()
172     if node.clauses[-1] == "()":
173         return node.clauses , node.results , node.records , True
174     for i in range(len(node.clauses)):
175         for j in range(i+1, len(node.clauses)):
176             if (i, j) in records:
177                 continue
178
179             parser1 = parser(node.clauses[i])
180             parser2 = parser(node.clauses[j])
181             ret = unifier(parser1 , parser2)
182
183             if ret[0]:
184                 copy_clauses = node.clauses.copy()
185                 copy_results = node.results.copy()
186                 copy_records = node.records.copy()
187
188                 result = "R[" + str(i + 1)
189                 if len(parser1) != 1:
190                     result += ret[3][0]
191                 result += "," + str(j + 1)
192                 if len(parser2) != 1:
193                     result += ret[3][1]
194                 result += "]"
195                 if len(ret[2]) >= 1:
196                     result += "(" + ret[2][0]
197                 for k in range(1, len(ret[2])):
198                     result += "," + ret[2][k]
199                 if len(ret[2]) >= 1:
200                     result += ")"

```

```

201
202         clause = ret[1]
203         if len(clause) > 1:
204             join_clause = "(" + ', '.join(clause) + ")"
205         elif len(clause) == 1:
206             join_clause = clause[0]
207         elif len(clause) == 0:
208             join_clause = "()"
209
210         if join_clause in copy_clauses:
211             continue
212
213         copy_clauses.append(join_clause)
214         copy_results.append(result)
215         copy_records.append((i, j))
216
217         pq.put(Node(copy_clauses, copy_results, copy_records
218                     ))
219
220     return clauses, results, records, False
221
222
223 if __name__ == "__main__":
224
225     clauses = []
226     results = []
227     records = []
228
229     clause_num = int(input("Please input the number of clauses: "))
230     for i in range(clause_num):
231         results.append("")
232         clauses.append(input())

```

```

233
234     # clauses = ["GradStudent(sue)", "(¬GradStudent(x), Student(x))", "(
        ¬Student(x), HardWorker(x))", "¬HardWorker(sue)"]
235     # results = ["", "", "", ""]
236
237     ret = astar(closures, results, records)
238     # ret = dfs(closures, results, records)
239     count = 1
240     # print(closures, results, records)
241     for result, clause in zip(ret[1], ret[0]):
242         print(str(count) + '.' + result + clause)
243         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"

```

1 Please input the number of clauses: 4
2 GradStudent(sue)
3 ( $\neg$ GradStudent(x), Student(x))
4 ( $\neg$ Student(x), HardWorker(x))
5  $\neg$ HardWorker(sue)

```

"hardworker-output-DFS"

```

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

```

"hardworker-output-A*"

```

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

```

"3block-input"

```

1 Please input the number of clauses: 5
2 On(aa,bb)
3 On(bb,cc)
4 Green(aa)
5  $\neg$ Green(cc)
6 ( $\neg$ On(x,y),  $\neg$ Green(x), Green(y))

```

"3block-output-DFS"

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

"3block-output-A*"

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

"AlpineClub-input"

1	Please input the number of clauses: 11
2	A(tony)
3	A(mike)
4	A(john)
5	L(tony, rain)
6	L(tony, snow)
7	(¬A(x), S(x), C(x))
8	(¬C(y), ¬L(y, rain))

9	$(L(z, \text{snow}), \neg S(z))$
10	$(\neg L(\text{tony}, u), \neg L(\text{mike}, u))$
11	$(L(\text{tony}, v), L(\text{mike}, v))$
12	$(\neg A(w), \neg C(w), S(w))$

"AlpineClub-output-DFS"

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

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

"AlpineClub-output-A*"

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