CS 4341 Project 5 Report CSP

Kewen Gu(kgu) and **Zhaochen Ding**(zding2)

General Introduction

In this project, we developed several algorithms to solve a constraint satisfaction problem. The problem is described as thieves trying to steal bags of stuffs from a store. All bags are values and stuffs in bags are variables.

Instructions on compiling and running your program

We used Python to write this project, the application we used to write code and compile is intelliJ. To run our program, use the following command:

```
python Main.py [input file name]
```

The default method uses MRV, LCV, and foward checking. To enable these methods, go to main.py file and change any boolean of csp = CSP(True, True, True) into false when testing. The three booleans are leastConstrainingValue, minimumRemainingValue, and forwardChecking in order.

To test our program on the given 26 input files, run the bash scripts in the bash terminal by entering:

```
./csp_test.sh
```

The outputs from the script will show the status of the assignment for each input file.

Detailed description of your approach. That is, what search method and what heuristics were used, how was AC employed, etc. Please try to describe your approach in an algorithmic manner (i.e., include pseudo-code of the overall approach).

We have four main algorithms used in this project, which are backtracking search method, least constraining value, minimun remaining value, and foward checking.
For back tracking method, we based our codes on the pseudo-code on the textbook(Artifitial Intelligence, a Morden Approach 3rd edition). The pseudo-code are shown below.

```
function backtrackingSearch(csp) returns a solution or failure
return backtracking(csp) => recurse
```

```
function backtracking(csp) returns a solution or a failure
if csp is complete or variable not in csp.var return the csp
for value in domain()
      add this assignment to csp
      add forwardChecking to csp
      if csp is still valid return csp
      else remove all assignments and return failure
```

The least constraning value heuristic is based on the searching of LCV. LCV prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph in the textbook. The pseudo-code looks like following:

```
function orderDomainValues(csp) returns an arrray
if run LCV is false return unfilled bag array without doing anything
else create a list of values and the num constraint
sort the list
for all values in list
      if value exist in unassigned list add it to another list
return the ordered list
```

The foward checking heuristic is pretty straightfoward, after a variable is assigned, forward checking checks every unassigned value that has a same constraint as the assigned variable. Also we think this method can ensure arc consistency. The pseudo-code is below:

```
function inference(csp, variable, value) returns an array
if run FC is false return empty array
else create an empty list
if variable is in binary equality array
      add (binaryequality[variable], value) into list
if variable is in value of binary equality array
      for each value in binary equality array
            if the value of the index in binary equality array = variable
                  add (index, value) into list
```

Describe which tests you ran to try out your program. How did your program perform?

We run our program with backtracking, backtracking + MRV + LCV and backtracking + MRV + LCV + forward checking seperately on eight randomly chose inputs for five times each and record the average runtime in ms. The data is recorded in the table below:

Radom Input	BackTracking	BT+MRV+LCV	BT+MRV+LCT+FC
1	1.08	60.1	23.3
2	0.98	1.61	1.47
3	0.90	1.28	1.12
4	0.37	0.50	0.55
5	0.16	0.22	0.23
6	0.29	0.33	0.25
7	0.31	0.33	0.26
8	0.34	0.59	0.53

We can conclude from the above form that the advanced algorithm turns out to be faster when the problem becomes complicated. Basically, the time consumption of algorithm without forward checking is bigger than the algorithm with FC so we can safely conclude that FC actually accelerates the solution seeking, especially when problems are complicated. In some of the entries' algorithm without FC take longer maybe because the code deals with arc consistency.

Describe the strengths and the weaknesses of your program.

We think our program is robust and straight-forward. And the execution time of our program is fairly satisfactory. Our program can provide an accurate results for most input files. However, we found that for some input files, our back tracking algorithm would fall into infinite loop when solving the csp. We still don't know the cause of this situation. We would like to investigate further, but we don't have time to do it now.